

Dipartimento di / Department of

..... Informatica Sistemistica e Comunicazione

Dottorato di Ricerca in / PhD program Computer Science Ciclo / Cycle XXX

Deep Learning for Feature Representation in Natural Language Processing

Cognome / Surname Nozza Nome / Name Debora

Matricola / Registration number 717156

Tutore / Tutor: Prof. Giuseppe Vizzari

Cotutore / Co-tutor: Dr. Elisabetta Fersini

Supervisor: Prof. Enza Messina

Coordinatore / Coordinator: Prof. Stefania Bandini

ANNO ACCADEMICO / ACADEMIC YEAR 2016/2017

To my parents and grandparents

Abstract

The huge amount of textual *user-generated content* on the Web has incredibly grown in the last decade, creating new relevant opportunities for different real-world applications and domains. To overcome the difficulties of dealing with this large volume of unstructured data, the research field of *Natural Language Processing* has provided efficient solutions developing computational models able to understand and interpret human natural language without any (or almost any) human intervention. This field has gained in further computational efficiency and performance from the advent of the recent Machine Learning research lines concerned with *Deep Learning*. In particular, this thesis focuses on a specific class of Deep Learning models devoted to learning high-level and meaningful representations of input data in unsupervised settings, by computing multiple non-linear transformations of increasing complexity and abstraction. Indeed, learning expressive representations from the data is a crucial step in Natural Language Processing, because it involves the transformation from discrete symbols (e.g. characters) to a machine-readable representation as real-valued vectors, which should encode semantic and syntactic meanings of the language units.

The first research direction of this thesis is aimed at giving evidence that enhancing Natural Language Processing models with representations obtained by unsupervised Deep Learning models can significantly improve the computational abilities of *making sense* of large volume of user-generated text. In particular, this thesis addresses tasks that were considered crucial for understanding what the text is talking about, by extracting and disambiguating the named entities (Named Entity Recognition and Linking), and which opinion the user is expressing, dealing also with irony (Sentiment Analysis and Irony Detection). For each task, this thesis proposes a novel Natural Language Processing model enhanced by the data representation obtained by Deep Learning.

As second research direction, this thesis investigates the development of a novel Deep Learning model for learning a meaningful textual representation taking into account the relational structure underlying user-generated content. The inferred representation comprises both textual and relational information. Once the data representation is obtained, it could be exploited by off-the-shelf Machine Learning algorithms in order to perform different Natural Language Processing tasks.

As conclusion, the experimental investigations reveal that models able to incorporate high-level

features, obtained by Deep Learning, show significant performance and improved generalization abilities. Further improvements can be also achieved by models able to take into account the relational information in addition to the textual content.

Publications

Journal

D. Nozza, E. Fersini, E. Messina (2018). “CAGE: Constrained deep Attributed Graph Embeddings”. *Information Sciences* (Submitted).

E. Fersini, P. Manchanda, E. Messina, **D. Nozza**, M. Palmonari (2018). “LearningToAdapt with Word Embeddings: Domain Adaptation of Named Entity Recognition Systems”. *Information Processing and Management* (Submitted).

Conferences

F. Bianchi, M. Palmonari and **D. Nozza** (2018). “Towards Encoding Time in Text-Based Entity Embeddings”. *Proceedings of the 17th International Semantic Web Conference*. (To appear)

E. Fersini, P. Manchanda, E. Messina, **D. Nozza**, M. Palmonari (2018). “Adapting Named Entity Types to New Ontologies in a Microblogging Environment”. *Proceedings of the 31st International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems*.

D. Nozza, F. Ristagno, M. Palmonari, E. Fersini, P. Manchanda, E. Messina (2017). “TWINE: A real-time system for TWweet analysis via INformation Extraction”. In *Proceedings of the Software Demonstrations at the 15th Conference of the European Chapter of the Association for Computational Linguistics* (pp. 25-28). Association for Computational Linguistics.

D. Nozza, E. Fersini, E. Messina (2017). “A Multi-View Sentiment Corpus”. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers* (Vol. 1, pp. 273-280). Association for Computational Linguistics.

P. Manchanda, E. Fersini, **D. Nozza**, E. Messina, M. Palmonari (2017). “Towards Adaptation of Named Entity Classification”. In *Proceedings of the 32nd ACM Symposium on Applied Computing* (pp. 155-157). ACM.

F. M. Cecchini, E. Fersini, P. Manchanda, E. Messina, **D. Nozza**, M. Palmonari, C. Sas (2016). “UNIMIB@NEEL-IT: Named Entity Recognition and Linking of Italian Tweets”. In *Proceedings of the 3rd Italian Conference on Computational Linguistics*. CEUR-WS.

D. Nozza, E. Fersini, E. Messina (2016). “Unsupervised Irony Detection: A Probabilistic Model with Word Embeddings”. In *Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management* (pp. 68-76). SciTePress. **(Best Paper Award)**

D. Nozza, E. Fersini, E. Messina (2016). “Deep Learning and Ensemble Methods for Domain Adaptation”. In *Proceedings of the 28th IEEE International Conference on Tools with Artificial Intelligence* (pp. 184-189). IEEE.

D. Nozza, D. Maccagnola, V. Guigue, E. Messina and P. Gallinari (2014). “A latent representation model for sentiment analysis in heterogeneous social networks”. In *Proceedings of the 12th International Conference on Software Engineering and Formal Methods* (pp. 201-213). Springer International Publishing.

Contents

Abstract	i
Publications	iii
Contents	iv
List of Figures	viii
List of Tables	x
1 Introduction	1
1.1 Thesis contribution and organization	3
2 Natural Language Processing	4
2.1 Challenges	5
2.2 Language Feature Representation	8
2.2.1 Term Presence and Frequency	8
2.2.2 Words and n-grams	9
2.2.3 Complex Linguistic Descriptors	9
2.2.4 Distributional features	10
3 Deep Learning Background	12
3.1 Challenges motivating Deep Learning	14
3.2 Artificial Neural Networks	16
3.3 Neural Network Architectures	17
3.3.1 Single-Layer Feedforward Network	17
3.3.2 Multi-Layer Feedforward Network	18
3.3.3 Recurrent Neural Network	19
3.4 Activation Functions	20
3.5 Learning method	21
3.5.1 Objective functions	22
3.5.1.1 Loss functions	23
3.5.1.2 Regularization	24
3.5.2 Training algorithm	25
3.5.3 Optimization algorithms	26
3.5.3.1 Gradient descent	26
3.5.3.2 Stochastic gradient descent	27
3.5.3.3 Stochastic gradient descent with Momentum	27

3.5.3.4	Stochastic gradient descent with Nesterov Momentum	28
3.5.3.5	AdaGrad	30
3.5.3.6	RMSProp	31
3.5.3.7	Adam	31
3.5.3.8	Adadelta	33
4	Deep Learning Architectures for Textual Feature Representation	34
4.1	Neural Networks Language Model	36
4.1.1	Neural Probabilistic Language Model	38
4.1.2	Collobert and Weston	41
4.1.3	Word2vec	41
4.2	Auto-encoder	43
4.2.1	Stacked Auto-encoder	45
4.2.2	Regularized Auto-encoder	46
4.2.3	Sparse Auto-encoder	46
4.2.3.1	k -sparse Auto-encoder	47
4.2.4	Denoising Auto-encoder	47
4.2.4.1	Marginalized Stacked Denoising Auto-encoder	48
4.2.5	k -Competitive Auto-encoder	49
5	Deep Learning Representation for Making Sense of User-Generated Content	51
5.1	Named Entity Recognition and Classification	54
5.1.1	Word Embeddings Representation for Learning to Adapt Entity Classification	57
5.1.1.1	Adaptation Model: Learning to Adapt with Word Embeddings	59
5.1.1.2	Experimental Settings	61
5.1.1.3	Experimental Results	66
5.1.1.4	Related Works	73
5.2	Named Entity Linking	74
5.2.1	Word Embeddings for Named Entity Linking	75
5.2.1.1	Representation and Linking model	77
5.2.1.2	Experimental Settings	78
5.2.1.3	Experimental Results	80
5.2.1.4	Related Works	85
5.3	Sentiment Analysis	87
5.3.1	Deep Learning Representation and Ensemble Learning methods for Domain Adaptation in Sentiment Classification	88
5.3.1.1	Deep Learning Representation and Ensemble Learning model	89
5.3.1.2	Experimental Settings	90
5.3.1.3	Experimental Results	93
5.3.1.4	Related Works	96
5.4	Irony Detection	97
5.4.1	A Probabilistic Model with Word Embeddings for Unsupervised Irony Detection	98
5.4.1.1	Unsupervised Topic-Irony Model	98
5.4.1.2	Experimental Settings	101
5.4.1.3	Experimental Results	102

5.4.1.4	Topic Detection results	106
5.4.1.5	Related Works	108
6	Enhancing Textual Feature Representation including Relational Information	110
6.1	Related Works	112
6.2	Deep Attributed Graph Embeddings Model	114
6.2.1	Problem Definition and Motivation	114
6.2.2	Constrained Deep Attributed Graph Embeddings Model	117
6.2.2.1	Auto-encoder	118
6.2.2.2	Textual Attribute Embedding Model	118
6.2.2.3	Structural Graph Embedding Model	119
6.2.2.4	Optimization problem	120
6.2.2.5	Toy Example	122
6.3	Experimental Settings	122
6.3.1	Dataset	124
6.3.2	Compared Models	124
6.3.3	Evaluation Framework	125
6.4	Experimental Results	126
7	Conclusion and Future Works	130
A	TWINE: A real-time system for TWeet analysis via INformation Extraction	133
A.1	Introduction	133
A.2	TWINE system	134
A.2.1	System Architecture	134
A.2.2	User Interface	137
A.3	Conclusion	138
B	Additional Results for LearningToAdapt model	139
	Bibliography	145

List of Figures

2.1	Example of a distributional representation.	10
3.1	Venn diagram of Deep Learning context.	14
3.2	Flowchart of Deep Learning context.	14
3.3	Neuron model.	17
3.4	Single-Layer Feedforward Network.	18
3.5	Multi-Layer Feedforward Network.	19
3.6	Recurrent Neural Network.	20
3.7	Comparison between linear and nonlinear transformations.	21
3.8	Activation functions.	22
3.9	Comparison between the gradient descent with and without momentum.	28
4.1	Neural language model proposed by Bengio et al. [1].	39
4.2	CBOW and Skip-gram architectures [2].	42
4.3	Auto-encoder architecture.	44
4.4	Stacked Auto-encoder architecture.	45
5.1	Example of the process of making sense of user-generated content.	52
5.2	Proposed framework.	52
5.3	Manual mappings between two generic ontologies.	57
5.4	Graphical example of L2A.	60

5.5	Example of a sentence processed by Named Entity Linking.	75
5.6	Example of Named Entity Linking similarity computation.	76
5.7	Pipeline of the proposed Named Entity Linking framework.	78
5.8	Accuracy of ensemble methods based on Decision Tree.	92
5.9	Accuracy of ensemble methods based on Support Vector Machines.	93
5.10	Transfer losses on the Amazon benchmark of 4 domains: Kitchen (K), Electronics (E), DVDs (D) and Books (B).	94
5.11	Transfer ratio on the Amazon benchmark.	95
5.12	The in-domain ratio versus transfer ratio.	95
5.13	Graphical representation of the proposed Topic-Irony Model.	100
5.14	Comparison of TIM and TIM+WE with supervised state of the art methods on the imbalanced dataset.	105
6.1	Graphical example of an attributed graph.	114
6.2	Selection of nodes used for explaining, first-order and second-order attribute proximity.	116
6.3	Graphical representation of the Constrained deep Attributed Graph Embedding model (CAGE).	117
A.1	TWINE system overview.	134
A.2	TWINE system architecture.	135
A.3	TWINE Map View snapshot.	135
A.4	TWINE List View snapshot.	136

List of Tables

5.1	Different commercial and academic Named Entity Recognition and Classification systems with their corresponding Generic Types.	55
5.2	Type Distribution (%) according to Ritter Ontology (O_S).	63
5.3	Type Distribution (%) according to Microposts Ontology (O_T).	63
5.4	Accuracy performance of L2A model considering Word Embeddings feature space.	67
5.5	Class-Wise Accuracy Contribution (%) on #Micropost2015 of L2A model considering Word Embeddings feature space (X_E and $X_{P \sim E}$).	68
5.6	Class-Wise Accuracy Contribution (%) on #Micropost2016 of L2A model considering Word Embeddings feature space (X_E and $X_{P \sim E}$).	68
5.7	Precision, Recall, F-Measure and STMM on #Micropost2015 of L2A model considering Word Embeddings feature space (X_E and $X_{P \sim E}$).	69
5.8	Precision, Recall, F-Measure and STMM on #Micropost2016 of L2A model considering Word Embeddings feature space (X_E and $X_{P \sim E}$).	69
5.9	Class-Wise Accuracy contribution (%) on #Micropost2015 of L2A model and baselines.	69
5.10	Class-Wise Accuracy contribution (%) on #Micropost2016 of L2A model and baselines.	69
5.11	Precision, Recall, F-Measure and STMM on #Micropost2015 of L2A model and baselines.	70
5.12	Precision, Recall, F-Measure and STMM on #Micropost2016 of L2A model and baselines.	70

5.13	Capabilities performance measures on #Micropost2015 of L2A model and baselines.	72
5.14	Capabilities performance measures on #Micropost2016 of L2A model and baselines.	72
5.15	Datasets statistics.	81
5.16	Results for #Micropost2015 without preprocessing.	82
5.17	Results for #Micropost2016 without preprocessing.	82
5.18	Results for NEEL-IT 2016 without preprocessing.	82
5.19	Results for #Micropost2015 with preprocessing.	82
5.20	Results for #Micropost2016 with preprocessing.	83
5.21	Results for NEEL-IT 2016 with preprocessing.	83
5.22	Results for #Micropost2015 with preprocessing and considering entity types.	83
5.23	Results for #Micropost2016 with preprocessing and considering entity types.	83
5.24	Results for NEEL-IT 2016 with preprocessing and considering entity types.	83
5.25	Comparison for #Micropost2015 sorted by F-measure.	85
5.26	Comparison for #Micropost2016 sorted by F-measure.	85
5.27	Comparison for NEEL-IT 2016.	85
5.28	Accuracy of the baseline and gold standard.	91
5.29	Results compared to a supervised state of the art method for each binary problem (O).	102
5.30	Results of the proposed models (TIM and TIM+WE) for each binary problem (O) distinguishing between ironic (+) and not ironic (-).	103
5.31	Results in terms of F-Measure of the proposed models (TIM and TIM+WE) against state of the art approaches.	103
5.32	Results of the proposed models (TIM and TIM+WE) for each binary problem (S) distinguishing between ironic (+) and not ironic (-).	104
5.33	Results of the proposed models (TIM and TIM+WE) on the imbalanced dataset (O) distinguishing between ironic (+) and not ironic (-).	106

5.34	Results of the proposed models (TIM and TIM+WE) on the imbalanced dataset (S) distinguishing between ironic (+) and not ironic (-).	106
5.35	Topic-related words are reported in bold, while the irony-related ones are underlined. These results are related to TIM in the original scenario (O) and the balanced settings.	107
5.36	Topic-related words are reported in bold, while the irony-related ones are underlined. These results are related to TIM+WE in the simulated scenario (S) and the balanced settings.	108
6.1	Toy example showing the impact of different proximity measures.	123
6.2	Comparison of CAGE vs S models on DBLP using 50% of labeled data.	126
6.3	Comparison of CAGE vs S models on CiteSeer-M10 using 50% of labeled data.	126
6.4	Comparison of CAGE vs T and S+T models on DBLP using 50% of labeled data.	127
6.5	Comparison of CAGE vs T and S+T models on CiteSeer-M10 using 50% of labeled data.	127
6.6	Comparison of CAGE vs S models in terms of F-measure _{micro} on DBLP using different % of labeled data.	128
6.7	Comparison of CAGE vs S models in terms of F-measure _{micro} on CiteSeer-M10 using different % of labeled data.	128
6.8	Comparison of CAGE vs T and S+T models in terms of F-measure _{micro} on DBLP using different % of labeled data.	129
6.9	Comparison of CAGE vs T and S+T models in terms of F-measure _{micro} on CiteSeer-M10 using different % of labeled data.	129
B.1	Accuracy performance on #Micropost2015 and #Micropost2016 Test set considering Word Embeddings feature space.	139
B.2	Class-Wise Accuracy Contribution (%) on #Micropost2015 Test set considering Word Embeddings feature space (X_E and $X_{P \sim E}$).	140
B.3	Class-Wise Accuracy Contribution (%) on #Micropost2016 Test set considering Word Embeddings feature space (X_E and $X_{P \sim E}$).	140
B.4	Precision, Recall, F-Measure and STMM on #Micropost2015 Test set considering Word Embeddings feature space (X_E and $X_{P \sim E}$).	140

B.5	Precision, Recall, F-Measure and STMM on #Micropost2016 Test set considering Word Embeddings feature space (X_E and $X_{P \sim E}$).	140
B.6	Class-Wise Accuracy Contribution (%) on #Micropost2015 Test set of L2A model and baselines.	141
B.7	Class-Wise Accuracy Contribution (%) on #Micropost2016 Test set of L2A model and baselines.	141
B.8	Precision, Recall, F-Measure and STMM on #Micropost2015 Test set of L2A model and baselines.	141
B.9	Precision, Recall, F-Measure and STMM on #Micropost2016 Test set of L2A model and baselines.	141
B.10	Capabilities performance measures on #Micropost2015 Test set of L2A model and baselines.	141
B.11	Capabilities performance measures on #Micropost2016 Test set of L2A model and baselines.	141
B.12	Accuracy performance on #Micropost2015 and #Micropost2016 Dev set considering Word Embeddings feature space.	142
B.13	Class Wise Accuracy Contribution (%) on #Micropost2015 Dev set considering Word Embeddings feature space (X_E and $X_{P \sim E}$).	142
B.14	Class Wise Accuracy Contribution (%) on #Micropost2015 Dev set considering Word Embeddings feature space (X_E and $X_{P \sim E}$).	142
B.15	Class Wise Accuracy Contribution (%) on #Micropost2015 Dev set of L2A model and baselines.	143
B.16	Class Wise Accuracy Contribution (%) on #Micropost2016 Dev set of L2A model and baselines.	143
B.17	Precision, Recall, F-Measure and STMM on #Micropost2015 Dev set considering Word Embeddings feature space (X_E and $X_{P \sim E}$).	143
B.18	Precision, Recall, F-Measure and STMM on #Micropost2016 Dev set considering Word Embeddings feature space (X_E and $X_{P \sim E}$).	143
B.19	Precision, Recall, F-Measure and STMM on #Micropost2015 Dev set of L2A model and baselines.	143

B.20 Precision, Recall, F-Measure and STMM on #Micropost2016 Dev set of L2A model and baselines.	143
B.21 Capabilities performance measures on #Micropost2015 Dev set of L2A model and baselines.. . . .	144
B.22 Capabilities performance measures on #Micropost2016 Dev set of L2A model and baselines.	144

Chapter 1

Introduction

With the continuous and fast evolution of the Internet and the advent of Social Web, or Web 2.0, the amount of **unstructured textual** data produced by the social interactions between people has become an immense hidden treasure of knowledge. It has been estimated that unstructured data, which come in the textual form of logs, emails, social media content, and customer comments, represents the 80% of the available data of the world. This means that, more often than not, we know little about data, causing people, organizations and institutions to lack of potentially highly valuable information.

Besides the problem of managing large volume of heterogeneous data, that cannot be performed by a human, the process of understanding and extracting meaningful information from natural language text is an extremely complex and difficult task. Each day we solve ambiguities, we infer intentions or emotions not explicitly stated, and we correct grammar mistakes without being consciously aware. Teaching computers to perform these tasks, even imperfectly, would deeply influence our lives: companies would improve their consumer satisfaction and engagement, doctors would provide more leading-edge and accurate treatments in a faster way, intelligence analysts would detect and monitor suspicious conversations easier, people in general would be able to obtain more interesting and engaging information when searching for restaurants or holiday destinations.

Natural Language Processing (NLP), which is the research field concerned with the process of understanding and interpreting human natural language, can contribute to these challenging goals. The major difficulties lie in the fact that human language is ambiguous, highly variable and constantly changing and evolving. Moreover, natural language is expressed in the form of discrete symbols, e.g. characters, that combined together can denote objects, concepts and actions. However, the mental representation that two words evoke in our mind has nothing to do with the symbols used to represent them, e.g. “beach” and “sun” do not have any common letter while their concepts are related.

A crucial step in Natural Language Processing, and in Machine Learning in general, is to find the right technique to represent the text in a form that machines can understand. The input representation can considerably influence the performance of Machine Learning models, in particular for NLP tasks, depending on its ability to disentangle and discover explanatory factors of variations behind the data given as input. **Representation Learning** has become an important research field aimed at learning transformations of data in order to simplify the extraction of useful information when building classifiers. In the last ten years, this task became even more important with the advent of **Deep Learning**. In Natural Language Processing, Deep Learning has provided efficient and performing computational models able to obtain a high-level and meaningful representation of the data by learning to represent knowledge as a nested hierarchy composed of multiple nonlinear transformations of increasing complexity and abstraction. The main advantage of Deep Learning for textual representation regards its ability to map discrete symbols, such as characters or words, to continuous real-valued vectors which encode semantic and syntactical meanings of the corresponding language units. An additional benefit of Deep Learning is that it is able to efficiently take advantage of large amount of unlabeled data for obtaining meaningful and abstract representations. This is particularly useful in the context of Web 2.0, where the huge amount of user-generated content exponentially increases everyday, generating the need of Machine Learning models, named **unsupervised**, able to extract valuable knowledge from data that have not been manually annotated.

While a consistent part of these user-generated content is created by individuals, the great majority is developed through collaborative efforts of multiple users (e.g. by commenting and sharing). This information takes the structure of an **attributed graph**, where the users with their generated content and interactions among users can be seen as nodes with attributes and edges respectively. In this context, this structure can be a great source of information, providing insights that are not possible to extract focusing only on the textual data, e.g. two users may show that they have similar opinions by commenting each other posts or by performing some approval interactions (e.g. like or share). In this context, most of the state of the art works are aimed at efficiently learning good representations for either textual or relational information. In particular, Representation Learning models have been defined for dealing only with text, disregarding the relational structure, or on the other side they have been developed for tackling relationships without taking into account the textual content.

However, jointly considering textual and relational information can provide effective improvements on Natural Language Processing tasks.

1.1 Thesis contribution and organization

The main problem addressed in this thesis is concerned with the extraction of meaningful information from user-generated content available on Social Media, exploiting high-level representations of text and relational structure. In order to address this issue, two main contributions have been provided.

First, this thesis gives evidence that joining the strength of Natural Language Processing and Deep Learning can contribute to the extraction of a meaningful text representation for different goals. With the aim of **making sense** of user-generated content, the investigated problems lead to better understand what the text is talking about, by extracting and disambiguating the named entities (Named Entity Recognition and Linking), and which opinion the user is expressing, dealing also with irony (Sentiment Analysis and Irony Detection).

Secondly, this thesis investigates the development of a novel Deep Learning model for learning meaningful textual representations considering an underlying relational structure. Once the representation is obtained, it could be exploited by off-the-shelf Machine Learning algorithms in order to perform different text classification tasks.

The thesis is organized as follows. In Chapter 2, a brief overview to the field of Natural Language Processing is presented where particular attention has been given to the current challenges on interpreting and understanding natural language text generated via Web 2.0 channels. Chapter 3 provides the background needed for the understanding of Deep Learning models, starting from the neural network architectures to the learning methods. Among all the Deep Learning approaches, this thesis focuses on a specific segment regarding unsupervised models for textual Representation Learning. Chapter 4 presents how to learn representations of textual data with unsupervised models (neural network language models and Auto-encoders). In Chapter 5, the first contributions related to the task of making sense of user-generated content is presented. Chapter 6 introduces the second contribution on Representation Learning considering both textual and relational information. Finally, in Chapter 7 several conclusions are derived and some future works are highlighted.

Chapter 2

Natural Language Processing

Since the beginning of the Digital Age, that started with the advent of World Wide Web and later on exploded with Web 2.0 or Social Web, the amount of digital data created and consumed has exponentially grown, reaching around 2800 Zettabyte distributed worldwide. It has been estimated that the majority of this huge amount of data comes from **unstructured** sources, as the impact of Social Media provided the users new time-efficient and interactive ways to communicate their ideas and opinions in disparate heterogeneous forms.

Among all these precious and valuable data, commonly known as **user-generated content**, only a small percentage is explored for obtaining its analytical value, despite nowadays it is in order for companies and institutions to ensure maximum profits and efficiency. The limited exploitation of the user-generated content resides on one side on the huge amount of data, too large to be handled by humans, and on the other side on the hidden semantics conveyed by the text, that is still a challenge for computational machines.

To this extent, different scientific communities (i.e. linguistics, psycholinguistics, computational linguistics, philosophy, statistics, and computer science) have contributed to the field of **Natural Language Processing (NLP)**, with the aim of obtaining computational models able to understand and interpret human natural language. The research areas in this field can be roughly distinguished in [3]:

- *Text-oriented*: information extraction, text summarization, machine translation, etc.;
- *Dialogue-oriented*: learning systems, question answering systems, etc.;
- *Speech-oriented*: speech recognition, speech segmentation, etc..

This thesis focuses on the first research area, meaning that the input data are assumed to be in textual form.

In [4], Jurafsky and Martin distinguished Natural Language Processing from other data processing systems because of the requirement to employ a *knowledge of language*. Considering, for example, the sentence:

I didn't enjoy the final GOT's episode, it was boring.

As first concern, NLP system should have the knowledge of **morphology**, the way words break down into component parts that carry meanings. In the sentence, it is important to understand contractions as *didn't* and to recognize that *GOT's* is a genitive. Then, the **syntactical** knowledge helps to understand the structure of the sentence (e.g. the word *GOT* can be identified as proper noun and not as auxiliary verb by considering the preceding adjective word *final*). In order to understand the meaning of the example sentence, NLP systems should know something about **lexical semantics**, the meaning of the words (e.g. *GOT*) as well as **compositional semantics** (e.g. the connection between *GOT* and *episode* or the meaning of using the adjective *final* related to *episode*). A **pragmatic** knowledge is needed when it can be relevant to understand the intention of the user. As in the example, it seems that the user is expressing a personal sentiment about the episode and suggesting an interchange of opinions. Moreover, a **discourse** knowledge is required in order to have some insights about linguistic units larger than a single utterance. The second part of the example sentence contains the pronoun *it*, that it is not trivially attributable to the other words. The next Section will first present the main challenges that researchers in the field of Natural Language Processing have to deal with since it represents a difficult and largely unsolved task. Then, Section 2.2 will be devoted to detail how textual data should be represented in order to be machine-readable and subsequently processed.

2.1 Challenges

Natural Language Processing is quickly evolving and the Deep Learning revolution has brought to light systems able to reproduce similar results to the ones obtained by humans. However, representing and understanding natural language is a very challenging problem and computational approaches are still a long way off to find effective and efficient solutions. Following, the main characteristic challenges on understanding and producing natural language text using computers are provided.

Word distribution An interesting property of the distribution of words in a text was described in [5], where the authors analysed the text of Mark Twain's Tom Sawyer and showed that, while the most common words accounted for slightly over half of the words in the text, there was a significant number (the other half) of the words that appeared only once in the corpus. This

means that the distribution of word frequency of occurrence in a text follows the Zipf's law [5]:

$$f \propto \frac{1}{r}. \quad (2.1)$$

where f is the frequency of the word and r is its position in the list of words sorted in descending order by their frequency. Therefore, there will be few very common words, a middling number of medium frequency words and many low-frequency words. As a result, training data will include a sufficient number of examples only for a restricted number of words, while the information available for the others will be exceedingly sparse. Consequently, it is hard to predict the behavior of words that are never or barely ever observed in the corpus. As the size of the data increases, the size of vocabulary increases. This means that, independently of the size of the training data, there will always be rare words difficult to analyse.

Word representation As introduced in Chapter 1, natural language text is expressed in the form of **discrete** symbols (e.g. characters) that combined together can denote concepts, objects and actions. While for colors it is possible to provide a mathematical representation and consequently to compute mathematical operations, such as the difference, for words it is not possible to apply the same reasoning, as there is not a simple operation that can provide the difference between the words “pink” and “red” without using a large lookup table or a dictionary [6]. Chapter 4 will present several Deep Learning approaches that provide efficient and effective solutions to this problem.

Ambiguity The biggest challenge in NLP involves the **ambiguity** problem. For natural language text, ambiguity can be referred at multiple levels: word sense, word category, syntactic structure, and semantic scope. Considering one of the most used example sentence *I made her duck*, Jurafsky and Martin [4] provided several possible meanings, each of which solves ambiguities at a different level:

1. I cooked waterfowl for her.
2. I cooked waterfowl belonging to her.
3. I created the (plaster?) duck she owns.
4. I caused her to quickly lower her head or body.
5. I waved my magic wand and turned her into undifferentiated waterfowl.

As a first consideration about ambiguity, the words *duck* and *her* are morphologically and syntactically ambiguous. Additionally, the verb *make* is semantically (it can mean *create* or *cook*)

and syntactically (it can be a transitive or intransitive verb) ambiguous. Thus, from this example it is possible to comprehend the difficulties that a computation machine can encounter on fully understanding and disambiguating natural language text, since it is a complex problem for humans too.

Language of Web 2.0 With the exponential growth of Social Media and Web 2.0 in general, the language register is substantially changed from well-formed documents to colloquial text. Consequently, there are still several open issues to properly tackle the real nature of these communications [7]:

- **Social Context:** The register on Web 2.0 is strongly related to the authors and media used. The textual contents are written from users of all ages, genders, culture orientations and nationalities (although English is the dominant language, as in every web-domain [8]). Therefore, these distinctive features can strongly influence the meaning and the use of specific language forms, arising additional issues related to the adaptation to different social contexts.
- **Short and Noisy Content:** Communications on social media platforms typically consist of very short text, where the users tend to convey complex meanings in few words or symbols. This short nature has led to the frequent use of abbreviations (e.g. *2morrw*), slang (e.g. *LOL*) and to the presence of a lot of grammar, syntactic and semantic errors, because the users often write very quickly. In general, the adopted language is similar to an oral and informal discussion, where it is not important to re-read and control the written sentences. The need of a short and incisive way to communicate has brought to the increasing use of visual language, the so-called *emojis*, as a standard for online communication [9]. It has been estimated that emojis cover over 10% of Twitter posts and over 50% of textual content on Instagram [10]. As outlined in [11], there are many possible challenging investigations on these visual features that would lead to an increasing understanding of social media communication and human-computer interaction. It is important to consider individual and communicative contexts for the computational modeling of the emojis, as it has been shown that people do not interpret them in the same way [12] and that their role is closely related to the context where they appear [13]. Other interesting issues regard the use of multimodal [14] and multilanguage approaches [11] that will considerably improve the global interpretation of emojis.
- **Dynamics:** Social Media are characterized by strong temporal dynamics that are commonly related to the continuous evolution of trending topics. The largely popular expressions used in social media platforms to track these topics are hashtags, a particular form of tag marked with a # symbol that includes one or more concatenated words. Although hashtags are most commonly recognized as topic-markers [15], different studies

proved that these symbols are more versatile from a linguistic point of view, resulting in a pragmatic function used to create and feed the evolution of communities [16, 17] and to support the visibility or participation to social causes [18, 19]. Currently, it is an interesting research direction to understand how hashtags can be used to search for information, track conversations and summarize the dynamics of discussions over time.

2.2 Language Feature Representation

Since Machine Learning approaches became the most adopted methods for understanding and interpreting text expressed in natural language, the way in which the data are represented should adapt to the conventions required by these computational methods. Many Machine Learning and Deep Learning approaches require the input data to be expressed as real-valued vectors, in order to compute mathematical functions and transformations. Therefore, the first step before applying these methods is to convert a natural language text to its vector representation, that is a major challenge per se, given the symbolic and discrete nature of language [6]. The mapping from textual to real-valued representation is called *feature extraction* or *feature representation*, where each feature is a measurable property that can be observed.

The process of extracting the set of features from the input data is called *feature engineering* and, as a crucial step, it can strongly influence the performance of Machine Learning models. Deep Learning methods alleviate this responsibility by providing end-to-end systems that are able to elaborate more meaningful representations and perform Machine Learning tasks all at once, as it will be presented in Chapter 3.

In the next sections, several feature representation techniques are presented and detailed.

2.2.1 Term Presence and Frequency

Traditionally, the input text (sentence, paragraph, or documents) is represented as a feature vector wherein the entries correspond to individual terms. In NLP, the most common feature representation is called **bag-of-words** (BOW): a binary-valued feature vectors in which the entries indicate whether an individual word occurs (value 1) or not (value 0).

One of the most immediate drawbacks of this representation consists on considering each word as equally important, while it may be useful to weigh the words that are more frequent. Consider a document $d \in D$ as composition of words w . The feature representation that reflects this need is called **Term Frequency** (TF) which is the raw frequency of a term w in a document d and can be defined as $TF(w, d) = \frac{\#w \in d}{|d|}$ where $|d|$ is the number of words in the document. While the

frequency is a good estimate of the word importance, it should be noted that there are usually common words that have a high count but do not carry any meaning (e.g. determiners and prepositions as *a*, *in*, *to*, or words that are strongly related to the document's topic as *ball* or *team* in a football match report). For obtaining the best trade-off, the term frequency is usually weighted by a term called **Inverse Document Frequency**, $IDF(w, D) = \log \frac{|D|}{|d_w \in d|}$. Finally, the *TF-IDF* measure can be expressed as:

$$TF-IDF(w, d, D) = TF(w, d) \times IDF(w, D)$$

2.2.2 Words and n-grams

Another level of interest for feature representation regards the choice of considering words or consecutive pairs or triples of words, so-called bigrams or trigrams. More generally, a feature representation composed of n consecutive words is called **n-gram**.

For $n = 1$ the feature is called unigram. For $n > 1$ the feature codes a list of consecutive words, $n = 2$ is called bigram, $n = 3$ trigram and so on. Whether higher-order n-grams are useful features compared to unigrams appears to be a matter of some debate. However, a crucial problem with n-grams is the combinatorial complexity of the dictionary construction, which increases exponentially with n . Even if the increasing dimensionality carries more useful information, it inevitably introduces considerable noise. The growth of available data implies then more difficulties, because of the introduced noise and the necessity of additional memory resources. It is important to notice that, while n-grams with $n = 4$ or $n = 5$ are sometimes used for letters, they are rarely used for words due to the increasing sparseness.

2.2.3 Complex Linguistic Descriptors

Since language is characterized by many other aspects beyond the directly observable presence or frequency of words, there have also been studies on incorporating more complex descriptors within the feature set. The most common set of descriptors regards word classes, or **part-of-speech** tags (POS). Their aim is to define the grammatical/lexical class of a word in a sentence (nouns, adjectives, adverbs, etc.). Considering these complex linguistic descriptors can provide benefits on different NLP tasks [20, 21]. It has also been studied the **semantic role** of a word, i.e. the underlying relationship of a word with the main verb in a clause (e.g. agent, instrument, goal). These relations between constituents can be valuable, for example, for identifying the arguments in a sentence. When the input data is a sentence, it is also possible to consider its **dependency tree**, which expresses the dependency between words in the sentence by child-parent relationships of nodes. Since each node corresponding to a word is connected by a

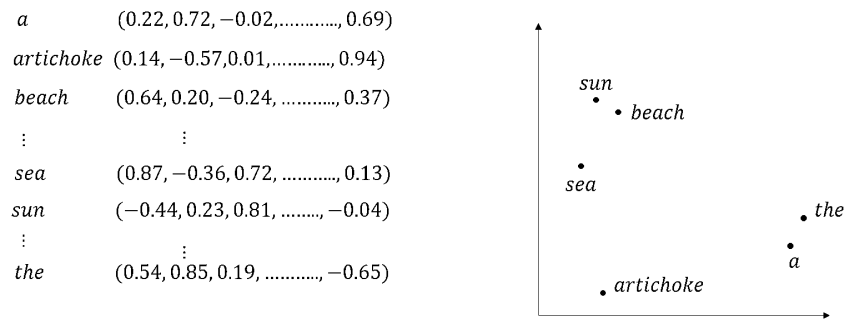


FIGURE 2.1: Example of a distributional representation as dense real-valued vectors and its graphical projection.

branch, a dependency subtree is able to give richer syntactic information. Considering the word dependencies can be very helpful, for example, for machine translation tasks.[22]

2.2.4 Distributional features

Since one of the main challenges in Natural Language Processing is related to the symbolic and discrete nature of text, several studies have been performed in order to address this issue and in particular to reflect the meaning of words according to the context where they appear. A fundamental contribution in the state of the art was proposed by Firth [23] and Harris [24], where the *distributional hypothesis of language* has been presented. According to this hypothesis, words that occur in the same contexts tend to have similar meanings. For instance, the words *sun* and *beach* are more likely to have a more similar meaning with respect to the word *artichoke* than it is commonly used in a completely different context.

Over the years, two main research directions have been investigated to model the distributional hypothesis of language: (1) **clustering-based** methods, where similar words are assigned to the same cluster and each word is then represented by its cluster membership [25, 26], (2) **embedding-based** methods, or *Word Embeddings*, which represent each word as a real-valued vector such that similar words (words appearing in the same context) have similar vectors [27, 28].

Figure 2.1 summarizes an example of distributional features that can be obtained by an embedding method. On the left side, an example of the dense real-valued vectors are reported, while the chart on the right side represents their graphical projection. As presented, similar words tend to be mapped next to each other (*sea* and *beach*) and distant from unrelated words (*artichoke*).

Given the impressive results and remarkable properties, Word Embeddings are the most popular approach in Deep Learning for NLP. For this reason, these methods are the most investigated

approach in this thesis and their principles and architectures will be explained more in details in Chapter 4.

Chapter 3

Deep Learning Background

The question “*can machines think?*” is a philosophical dilemma that has existed for decades. Beyond the fascinating conceptual issue, the answer can have a strong impact on our daily lives. To many, the ability of computers to process language as skillfully as we humans do, will signal the arrival of truly intelligent machines. The basis of this belief is the fact that the effective use of language is intertwined with our general cognitive abilities [4].

In 1956, a new field related to the building and understanding of intelligent entities was born under the name of **Artificial Intelligence** (AI) [29]. This field is constantly and actively growing and changing, involving numerous applications and many challenges, such as understanding speech or images, write poetry, and diagnose diseases. While the main goal of AI is to conceive intelligent machines able to perceive, understand and predict a complicated and boundless world, AI studies have been mostly performed in relatively sterile and controlled environments. From its earliest days, AI was very successful in solving problems that can be intellectually difficult for human beings but relatively straightforward for machines, i.e. problems that can be described by mathematical rules. The most representative event realization of this statement happened in 1997 when an AI system beat the reigning world champion at chess under tournament regulation, a game with very formal rules.

Indeed, the true challenge for AI is to perform, or approximate, tasks that are very intuitive for humans but require a huge amount of knowledge about the world that cannot be formally described. Each day, we recognize objects and people’s identities, we understand words and intentions from a speech, and we correct grammar mistakes without being consciously aware. The issue regards to the complexity on how to represent and incorporate this subjective informal knowledge into artificial intelligent systems.

The first forms of investigation into AI, called **knowledge-based systems**, reside on the explicit representation of the knowledge in the form of words and symbols. The machine expresses

information and reasons in terms of rules or cases. Despite the early successes, most of the human intelligence has remain difficult to represent [30], revealing the need of AI systems able to build up their own knowledge model, based on observations and experience. These systems are known as **Machine Learning** models. Using this approach, machines can perform tasks and make decisions by learning from past experience as humans do, e.g. they can recognize faces in images, they can infer the emotion beyond a text or a speech, etc.

However, the performance of these Machine Learning algorithms heavily depends on the representation of the data given as input. Each piece of information included in the representation is known as a *feature*. Indeed, the Machine Learning models strongly depend on how features are selected and engineered.

One solution to the problem of choosing the right set of features is to use Machine Learning to learn a suitable feature representation, this approach is known as **Representation Learning**. Representation Learning methods aim at efficiently obtaining good representations that allow AI systems to rapidly adapt to new tasks and improving performance with respect to hand-crafted features.

According to Yoshua Bengio and Vincent [31], the main goal of data representation, or algorithms for Representation Learning, is to identify and disentangle the underlying explanatory factors hidden in the observed data. In [32], *factors* are defined as separate sources of influence that are often not quantities that can be directly observed, but unobservable objects or forces that influence the observable quantities. These factors can be viewed as concepts or abstractions that help us make sense of the rich variability in the data. For example, the concept of word gender is a factor of variation underlying words, e.g. “man”-“woman”, “uncle”-“aunt” or “king”-“queen”. Identifying these latent factors is a crucial issue for AI systems because they fully influence the observed data in a covert way. For instance, when extracting the users’ opinion from a text, understanding that the adjectives “malfunctioning” and “reliable” for kitchen appliances have the same polarity of “boring” and “hilarious” respectively for DVDs can provide very powerful models. For the same task, it is crucial to interpret words respect to their contexts, e.g. an “unpredictable book” is a book that keeps its readers interested, while an “unpredictable car” suggests a car that has unexpected and dangerous behaviors.

The extraction of these hidden factors from raw data can be very complex. Their identification seems to be possible only using sophisticated, nearly human-level understanding processes of the data. **Deep Learning** solves this central problem in Representation Learning by introducing representations that are expressed in terms of other, simpler representations. It achieves great power and flexibility by learning to represent the world as a nested hierarchy, where higher level, more abstract concepts, can be learned from the lower level ones. Hence, Deep Learning helps to disentangle these abstractions and efficiently extracts the right set of features that are useful

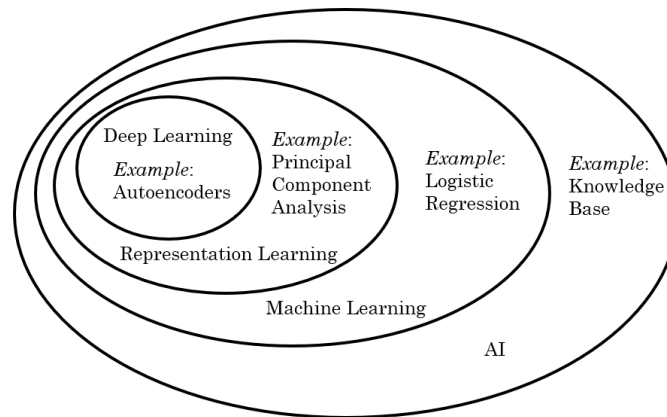


FIGURE 3.1: Venn diagram showing the relationships between different AI disciplines. Each section of the Venn diagram includes an example of an AI technology [32].

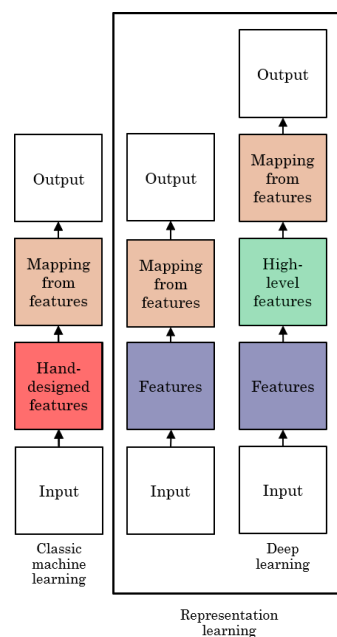


FIGURE 3.2: Flowcharts showing the relationships between different AI disciplines [32].

for improving performance. Figures 3.1 and 3.2 represent the diagrams proposed by Yann LeCun and Hinton [32] that provide an effective visual representation and summarization of the relationships between AI, Machine Learning, Representation Learning, and Deep Learning.

3.1 Challenges motivating Deep Learning

While Deep Learning dates back to the 1950s, this field has gained a lot of attention and popularity only in the last ten years. The motivations behind this renaissance were firstly presented by Geoffrey Hinton in a talk, called “Deep Learning”, to the Royal Society in 2016. Following,

this section investigates the factors that disruptively changed the perspectives and interests in this field.

Data availability Considering the evolution of the Internet and the recent increasing digitization of the society, the number of online users and the content that they generate is exploding. This has led to a proliferation of large resources that considerably help the learning phase of Deep Learning models. While huge datasets are one of the key factors of the success of the current Deep Learning approaches, it is often not possible to perform supervised classification tasks because of the enormous human effort of data labeling. In this context, interesting studies have been proposed in order to take advantage of unlabeled examples when small training sets are available [33, 34] or to perform unsupervised learning [35–37].

Computational power Deep Learning algorithms need a considerable computational power (both memory and processor) to efficiently run for solving several tasks such as Natural Language Processing and image object recognition. Differently from the 1950s, researchers and people in general have the possibility to access to more powerful computational resources. In particular, the exploitation of GPU computing arises after NVIDIA released the high-level language CUDA in 2006. GPUs provided many orders of magnitude of computing power more than CPUs, they perform matrix operations for back-propagation more efficiently and they make use of massively parallel graphics processors to accelerate computations.

Improved learning algorithms In 2006, Geoffrey E. Hinton and Teh [38] and Bengio et al. [39] published breakthrough works on a faster learning algorithm for neural networks. They proposed a greedy layer-wise unsupervised pretraining followed by a supervised fine-tuning. Each layer is pretrained with an unsupervised learning algorithm, learning a nonlinear transformation of its input (the output of the previous layer) that captures its main underlying variations [40].

Beyond this breakthrough approach, novel training improvements have been proposed in the last few years for solving problems better and providing improved performance: more efficient activation functions (Sec. 3.4), regularization techniques able to prevent overfitting (Sec. 3.5.1) and more robust optimizers (Sec. 3.5.3).

Real-world impact Another key success factor of Deep Learning regards the impressive advancement in various tasks such as object recognition [41], pedestrian detection [42], speech recognition [43, 44], robotics [45], machine translation [46, 47], etc.

Deep Learning algorithms gained popularity as massively used by many world's largest technology companies including Google, Microsoft, Facebook, IBM, Apple, and NVIDIA Corporation. Moreover, the software infrastructure have strongly influenced the advancement in this field: Theano [48, 49], Torch [50], Caffe [51], TensorFlow [52], etc.

As final analysis, Deep Learning is providing innovative, efficient and technical advanced solutions for several problems and domains.

None of that would have been possible without the advancement in the fundamental model family of neural networks. Indeed, the next sections give an introduction of Artificial Neural Networks (Sec. 3.2) and the most popular employed architectures to design them (Sec. 3.3). Afterwards, the basic operations of artificial neurons and the commonly adopted training methods are described in Section 3.4 and 3.5 respectively.

3.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are the most dominant model family in Deep Learning. An Artificial Neural Network is an information-processing system developed as generalizations of mathematical models of neural biology. In this system, the *neuron* (Figure 3.3) is the fundamental unit block to process information and it is composed of several elements:

- The *input nodes* are associated to elements of the input signal expressed as n -dimensional vector $\mathbf{x} \in \mathbb{R}^n$.
- The signal flows over the *connection links*, or synapses. Each link has an associated weight, which typically multiplies the transmitted signal. The set of weights is defined as a real-valued vector $\mathbf{w} \in \mathbb{R}^n$, that includes negative as well as positive values.
- A supplementary signal, called *bias*, is added to the input. This parameter $b \in \mathbb{R}$, that is generally permanently set to 1, is important because it can increase or lower the neuron value, a crucial effect during the learning phase.
- The *adder* component computes a weighted sum of all the input signals, weighted by the respective connection strength. In addition to summing, the adder component can select the minimum, maximum, majority, product or several normalizing algorithms.
- Finally, each neuron applies an *activation function* $a : \mathbb{R} \rightarrow \mathbb{R}$, usually nonlinear, to the input weighted sum to determine the final output signal \hat{y} :

$$\hat{y} = a \left(\sum_{i=1}^n w_i x_i + b \right) = a(\mathbf{W}\mathbf{x} + b) \quad (3.1)$$

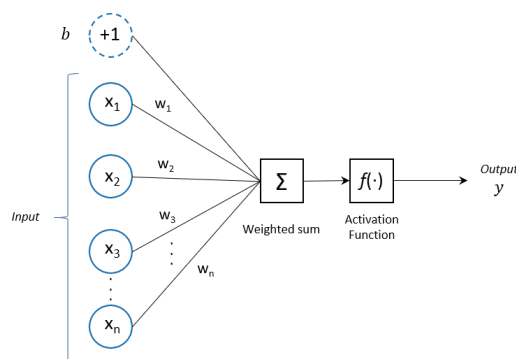


FIGURE 3.3: Model of a neuron.

The activation function can also scale the output or control its value via thresholds.

To summarize, an ANN is a network of artificial neurons, which receive an input, change their internal state according to the input, and produce an output with respect to the activation function. An ANN is composed by the connection of artificial neurons, denoting a directed and weighted graph, where the nodes are the neurons and the weights' edges are the connection links between neurons. The weights are randomly initialized and adjusted during the learning phase.

In the next sections, several building blocks of Artificial Neural Networks are presented: the *architecture* that defines the inter-connection between neurons, the *activation functions*, and the *learning method* to determine the connection weights.

3.3 Neural Network Architectures

A key design consideration for neural networks lies in determining their *architecture*, i.e. the overall structure of the network. For convenience, most neural networks are organized into groups of units called *layers*. Within each layer, neurons usually have the same activation function and the same pattern of connections to other neurons. The network can be either fully connected when every node in each layer is connected to every node in the adjacent forward layer, or partially connected, when some connections are missing from the network.

Indeed, the design choices concern the dimensions and amount of layers and how they are connected to each other.

3.3.1 Single-Layer Feedforward Network

The simplest form of a layered network, called *single-layer feedforward network*, has one layer of connection weights. The designation “single-layer” refers to the presence of only one layer

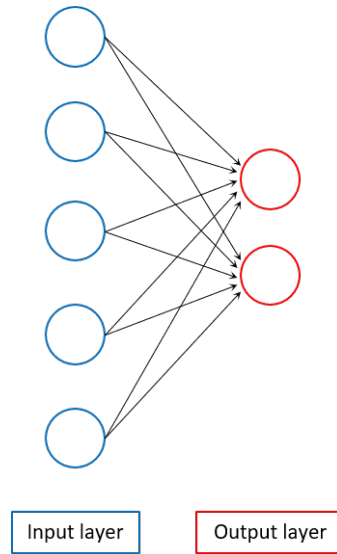


FIGURE 3.4: Single-layer feedforward network where the input and output layers are composed of five and two nodes respectively.

of computational nodes, i.e. the output neurons. While the term “feedforward” is related to the direction of the information flow from the input to the output layer and not vice-versa. The involved neuron group units are the *input layer* that receives the signals, and the *output layer* that computes and transforms the input signal. Figure 3.4 illustrates the typical single-layer feedforward network where the input layer is fully-connected with the output one.

Denoting \mathbf{h} as the activation of a layer, it is possible to define a single-layer feedforward network as:

$$\mathbf{h} = a(\mathbf{W}\mathbf{x} + b) \quad (3.2)$$

where, in this case, \mathbf{h} refers only to the output layer because no computations are performed in the input layer.

3.3.2 Multi-Layer Feedforward Network

When more complicated input transformations should be expressed, several layers can be introduced in the network resulting in an architecture called *multi-layer feedforward network*. Differently from the previous one, this network has one or more *hidden layers* or *activation layers* between the input and the output. By the introduction of these hidden layers, the network is enabled to extract latent factors of variations from its input. Moreover, the network acquires the ability to capture a global perspective despite its local connectivity, due to the additional set of synaptic connections and the increased neural interactions [53].

According to this architecture, the input signal vector is fed to the input layer, consequently the computational nodes of the second layer (i.e., the first hidden layer) provide the activation

pattern onto the input layer nodes. Then, the output signals of the second layer are used as inputs to the third one, and so on for the rest of the network. The nodes of a layer receive as input signal the output signals of the preceding layer only. The final output signals of the last layer constitute the response of the network to the first (input) layer. The architectural graph in Figure 3.5 shows the layout of a fully-connected multi-layer feedforward network with one hidden layer.

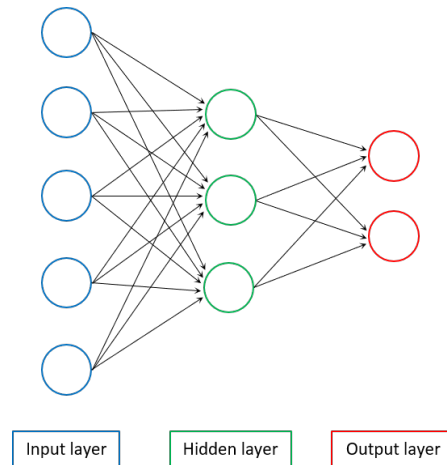


FIGURE 3.5: Multi-layer feedforward network where the input, hidden and output layers are composed of five, three and two nodes respectively. In this example, the number of hidden neurons is less than the number of input neurons, but the opposite situation is also possible.

A multi-layer feedforward network can be formally defined as a stack of u hidden layers characterized by a weight matrix $\mathbf{W} \in \mathbb{R}^{u \times n}$ and a bias vector $\mathbf{b} \in \mathbb{R}^u$. The complete set of parameters for the neural network can be defined as $\theta = (\mathbf{W}, \mathbf{b})$. To include multiple layers in the model, it is possible to rewrite Equation 3.2 as follows:

$$\begin{aligned} \mathbf{h}_0 &= a_0(\mathbf{W}\mathbf{x} + \mathbf{b}_0) \\ \mathbf{h}_l &= a_l(\mathbf{W}\mathbf{h}_{l-1} + \mathbf{b}_l) \end{aligned} \tag{3.3}$$

where the subscript l corresponds to the sequence position of the layer in the network, starting at $l = 1$. The parameter set θ refers to the parameters of network, while θ_l is the specific parameter set of the l -th layer \mathbf{h}_l .

3.3.3 Recurrent Neural Network

A *recurrent neural network* [54] differs from a feedforward neural network for the presence of at least one *feedback* loop. The feedback connections are used to feed back the output of the model into itself. This kind of neural network are specialized for processing a sequence of values $x^{(1)}, \dots, x^{(n)}$. The intuition behind the transition from multi-layer network to recurrent network is the idea of sharing parameters across different parts of the model, in this case across several

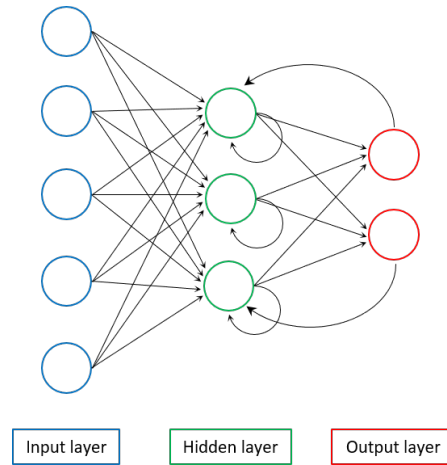


FIGURE 3.6: Recurrent neural network model.

time steps. Figure 3.6 shows the architecture of a recurrent neural network, where feedback connections are originated from hidden neurons as well as from the output neurons.

More details for a formal definition of recurrent networks are provided in [32].

3.4 Activation Functions

The basic operation of an artificial neuron involves summing its weighted input signals and applying an *activation function*. Typically, the same activation function is used for all neurons within each layer in the network. For more realistic results, nonlinear activation functions are commonly applied. These nonlinear transformations can considerably help when the input data are not linearly separable in the input space, providing a new representation space in which the transformed data can be linearly separated.

Figure 3.7 shows a simple example on a dataset composed of two curves on a plane referred to two different classes (blue and red). The simplest configuration of a one-layer neural network (Fig. 3.7(a)) will trivially classify the input space with a linear separation line. Interestingly, the addition of a hidden layer, that computes nonlinear functions, transforms the input signal generating a linearly separable space (Fig. 3.7(b)). Indeed, the resulting model is able to capture nonlinear relationships between the input and the output data, helping to discover latent factors in the data.

A crucial property of the activation functions is that they must be differential and therefore continuous. This property is necessary to enable the error correction during the training phase. In particular, the computation of the derivative is needed during the gradient-based optimization learning process, as it will be detailed in Section 3.5. In Figure 3.8, the most adopted activation functions in neural networks are listed.

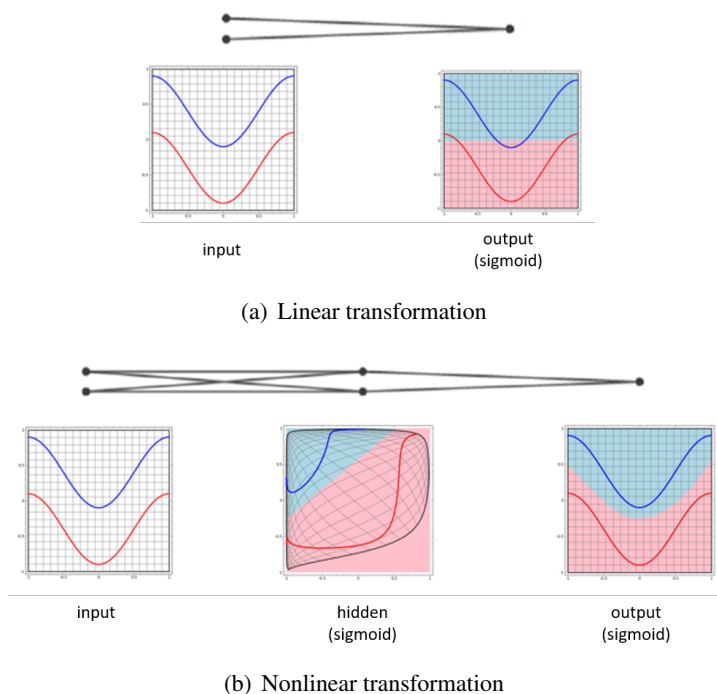


FIGURE 3.7: Comparison between linear and nonlinear transformations.
<http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/>

For neural networks, the rectified linear activation function (*ReLU*) [55, 56] is generally recommended because it is usually more efficient than other traditional activation functions such as *sigmoid* or *tanh*. While the transformations applied by *ReLU* are nonlinear, it can be represented by a piecewise linear function with two linear pieces. This property will consequently lead to a lower computation complexity of the training algorithm and higher generalization abilities of the Artificial Neural Networks. It will also help to solve the problem of vanishing gradient, i.e. value exceedingly close to 0, which causes instability. Figure 3.8 shows two extensions of *ReLU*: the parameterised rectified linear activation function (*PReLU*) [57] that adaptively learns the parameters of the rectifiers, and the exponential linear units (*ELU*) [58] which aims to make the mean activations closer to zero speeding up the learning. In 2000, Dugas et al. [59] introduced a smooth version of the rectifier, called *Softplus*. Despite this function is differentiable everywhere, Glorot et al. [56] proved that the use of rectifier linear units leads to better results and more intuitive hidden transformations.

3.5 Learning method

As it was presented so far, neural networks are models able to disentangle latent factors of input data, by means of an architecture composed of several fully-connected nonlinear transformations layers. The introduction of these additional hidden layers motivates the neurons to represent even more abstract and ultimately unintelligible concepts. The neural networks models will then










Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

FIGURE 3.8: List of activation functions and their derivatives (https://en.wikipedia.org/wiki/Activation_function).

be very hard to interpret. For this reason, Deep Learning is often associated with the term “black box” model, where it is not possible to see the inner computations or they are not examined but just executed. Another problem related to complexity consists of the increasing dimension of the search space, that grows exponentially with the number of activation layers. This motivates the learning process to be more complex and computationally expensive.

This section outlines the commonly adopted *objective functions*, the most used training algorithm for neural networks, *back-propagation*, and the most popular *optimization algorithms* adopted to solve the afore-mentioned issues.

3.5.1 Objective functions

Most Machine Learning models, and Deep Learning algorithms too, involve an optimization phase. Generally, optimization refers to the task of either minimizing or maximizing some function $\mathcal{L}(\theta)$. Usually, an optimization problem seeks to minimize the objective function, taking the name to **loss function**, cost function or error function. In case of maximization, it suffices to minimize $-\mathcal{L}(\theta)$.

The objective function $\mathcal{L}(\theta)$ typically returns a metric that quantifies the distance between the neural network output $\hat{\mathbf{y}}$ computed by passing the input data through the nonlinear transformation layers and the expected response \mathbf{y} , usually provided by the training data. The aim of the optimization process is to find the best set of parameters θ that minimizes this distance.

As follows, the case of a supervised problem is used as a concrete example for giving a more formal explanation. Given a training set of n instances (\mathbf{x}, \mathbf{y}) , where \mathbf{x} are the input data and \mathbf{y} the corresponding labels, a per-instance loss function L , a parameterized function $f_{\theta}(\mathbf{x})$ that denotes the neural network model which returns the output $\hat{\mathbf{y}}$, the corpus-wide loss function can be computed as the average loss over all the training instances:

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n L(f_{\theta}(x_i), y_i) \quad (3.4)$$

As previously stated, the principal component that determines the loss value is the parameter set. The aim of the training process is then to find the parameters θ that minimize the value of \mathcal{L} :

$$\hat{\theta} = \arg \min_{\theta} \mathcal{L}(\theta) = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n L(f_{\theta}(x_i), y_i) \quad (3.5)$$

When performing optimization over the training set, the computed loss tends to be very specific for the given dataset and consequently not sufficiently generalizable for new unknown inputs. To overcome this issue, called *overfitting*, it is possible to employ **regularization** methods [60], which aim to smoothly restrict the loss function reducing the error on new data, possibly at the expense of increased training error. Essentially, a regularization term $\mathcal{R}(\theta)$, balanced by the hyperparameter λ , is added to the objective function in order to minimize the complexity of the parameters:

$$\mathcal{L}(\theta) = \left(\overbrace{\frac{1}{n} \sum_{i=1}^n L(f_{\theta}(x_i), y_i)}^{\text{Loss}} + \overbrace{\lambda \mathcal{R}(\theta)}^{\text{Regularization}} \right) \quad (3.6)$$

Several loss functions and regularization methods have been proposed and their combination can lead to different learning algorithms. Following, this section introduces the most commonly employed loss functions and regularization techniques.

3.5.1.1 Loss functions

The most widely adopted loss functions used for training Artificial Neural Networks for NLP tasks are presented in the following. For each input instance x_i , the per-instance loss function L is expressed in terms of the label y_i and the neural network prediction $f_{\theta}(x_i) = \hat{y}_i$.

Mean Squared Error (MSE) requires as output real values $y_i \in \mathbb{R}$. The prediction error is squared and averaged over all the training set instances, as shown in the following equation:

$$L_{MSE}(\hat{y}_i, y_i) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (3.7)$$

The main drawback of this loss is that it often leads to poor results when used with gradient-based optimization [60].

Hinge loss is the loss function usually adopted for binary classification problems $y_i \in \{-1, +1\}$. The classification is correct if the predicted \hat{y}_i and the output y_i share the same sign, meaning that $\hat{y}_i y_i > 0$. The hinge loss is defined as:

$$L_{hinge}(\hat{y}_i, y_i) = \max(0, 1 - \hat{y}_i y_i) \quad (3.8)$$

The loss value is 0 if the network output and the desired output share the same sign. Eventually, the hinge loss can be extended for multi-classification purposes [61].

Log loss displays a similar convergence rate to the hinge loss function and it is continuous. This loss can be seen as a soft variation of the hinge loss:

$$L_{log}(\hat{y}_i, y_i) = \log(1 + e^{-\hat{y}_i y_i}) \quad (3.9)$$

Binary cross-entropy loss [62] is usually adopted for binary classification with conditional probability outputs. The cross-entropy loss is defined as:

$$L_{cross-entropy}(\hat{y}_i, y_i) = -y_i \log \hat{y}_i - (1 - y_i) \log (1 - \hat{y}_i) \quad (3.10)$$

This loss is convex and closely related to the Kullback-Leibler divergence [63] between the empirical distribution and the predicted distribution.

3.5.1.2 Regularization

As presented in Equation 3.6, the regularization term takes as input the neural network parameters and returns a score representing their complexity. The aim of the training algorithm is then to both minimize the loss function and the complexity of the parameters. The hyperparameter $\lambda \in [0, \text{inf})$ weights the relative contribution of the regularization term: a large value favors simple models to a lower loss, while a low value increases the importance of having the minimum loss at the cost of a higher complexity. Usually, a regularizer \mathcal{R} measures the parameter matrices norms and aims to return solutions with low norms. As in the state of the art, this thesis applies regularization only to the weight matrix \mathbf{W} and leaves the biases \mathbf{b} unregularized. This decision is due to the fact that biases typically require less data to fit accurately than the weights and their regularization can introduce a significant amount of underfitting [56].

The most commonly adopted regularizers are presented as follows.

L_2 regularization [64], also called as weight decay or ridge regression, is one of the simplest parameter norm penalty. This regularization term can be written as the sum of squares of the network weights:

$$\mathcal{R}_{L_2}(\mathbf{W}) = \|\mathbf{W}\|_2^2 = \sum_{i,j} (W_{ij})^2 \quad (3.11)$$

It should be noted that L_2 regularizers are severely penalized for high values of parameter weights, but once the values are close enough to zero, their effect becomes imperceptible [6].

L_1 regularization [65], commonly known as lasso, is written as the sum of absolute values of the network weights:

$$\mathcal{R}_{L_1}(\mathbf{W}) = \|\mathbf{W}\|_1 = \sum_{i,j} |W_{ij}| \quad (3.12)$$

Differently from L_2 , L_1 regularization term is particularly recommended for sparse solutions, i.e. models where many parameters have an optimal value of zero.

Elastic net [66] is the combination of L_1 and L_2 regularizers:

$$\mathcal{R}_{elastic.net}(\mathbf{W}) = \lambda_1 \mathcal{R}_{L_1}(\mathbf{W}) + \lambda_2 \mathcal{R}_{L_2}(\mathbf{W}) \quad (3.13)$$

where the hyperparameters λ_1 and λ_2 control the contribution of the two regularization terms.

Dropout [67, 68] is a regularization technique proposed for addressing the problem of overfitting in deep neural networks. The core idea is to randomly drop units (and their connections) from the neural network during the training phase. The learned model will then significantly reduce the generalization error compared to the other regularization methods. Although the use of dropout regularization generally causes a higher computational training time because of slow convergence, this method has the advantage of generally leading to significantly improved results.

3.5.2 Training algorithm

A highly popular method for training neural networks is the **back-propagation** algorithm [69–71]. The training process consists of two phases that perform different computations by passing through the network into two different directions: a forward phase and a backward phase.

In the *forward* pass, the input signal \mathbf{x} is propagated through the network layers until it reaches the output. During this phase, the weights of the connection links between neurons are fixed. The output $\hat{\mathbf{y}}$ computed by the last layer of the neural network is then compared with the desired

response \mathbf{y} using a loss function (Sec. 3.5.1), the resulting value is used as error signal in the *backward* pass. This error signal is propagated into the network layer-by-layer, in the opposite direction of the input: from the last to the first layer. The weights of the connection links are then adjusted in order to minimize the error, so that the network output would become more similar to the desired response. The computations of the adjustments for the output layer are straightforward, but they become much more challenging for the hidden layers.

More technically, the back-propagation algorithm calculates the gradient of the loss function for each layer in an iterative way, by using the chain-rule of derivatives. The gradient is then fed to the optimization method which uses it to update the weights of the artificial neurons, in an attempt to minimize the loss function. The optimization algorithms actively adopted for training neural network models are presented in the next section.

3.5.3 Optimization algorithms

The most difficult optimization problem in neural networks regards the training. Even the training of a single instance can take days or months of time on hundreds of machines. Given the importance of this problem, several specialized optimization techniques have been proposed in order to solve the issues of increasing complexity and computational costs.

The most effective modern optimization algorithms are based on *gradient descent* and include Stochastic Gradient Descent, AdaGrad, RMSProp, Adam and Adadelta, eventually associated with momentum or Nesterov momentum.

3.5.3.1 Gradient descent

Gradient descent is the core gradient-based optimization algorithm for training neural networks. This method can be easily applied, especially for convex functions, to compute the minimum of the objective function $\mathcal{L}(\theta)$ (Eq. 3.4). At each step of the iterative process, the model parameters θ are updated in the direction of the gradient of the objective function. Indeed, the update step presented in Equation 3.4 is:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \left(\frac{1}{n} \sum_{i=1}^n L(f_{\theta}(x_i), y_i) \right) \quad (3.14)$$

where η is the *learning rate*, i.e. a hyperparameter that controls how much the training algorithm is adjusting the weights of the neural network with respect to the loss gradient.

The update procedure in Equation 3.14 takes into consideration the sum of the gradients of the loss function computed over the n instances of the training set, this approach is known as **batch**

learning. Although batch gradient descent has shown impressive convergence property, it is rarely used because of the enormous computational cost for large datasets. A solution to tackle this issue is to compute the gradient over a small sample of m_b training instances, assuming that all samples are independent and identically distributed. When $m_b = 1$, the algorithm is called **online** gradient descent, while for $m_b > 1$ the algorithm is called **minibatch** gradient descent [72].

3.5.3.2 Stochastic gradient descent

Stochastic gradient descent (SGD) [73, 74] is a stochastic approximation of the gradient descent optimization, which follows an iterative procedure for minimizing the objective function. The SGD method is presented in Algorithm 1.

Algorithm 1 Stochastic gradient descent

```

1: Require: Learning rate  $\eta$ 
2: Require: Initial set of parameters  $\theta$ 
3: while Stopping criterion not met do
4:   Sample a minibatch of  $m_b$  examples from the training set  $\{x_i, \dots, x_{m_b}\}$ .
5:   Set  $g = 0$ 
6:   for  $i = 1$  to  $m_b$  do
7:     Compute gradient estimate:

```

$$g \leftarrow g + \nabla_{\theta} L(f_{\theta}(x_i), y_i).$$

```

8:   end for
9:   Apply update:  $\theta \leftarrow \theta - \eta g$ 
10: end while

```

Given a learning rate η and an initial set of parameters θ , the algorithm computes the gradient of the loss function with respect to the parameters over a minibatch of m_b instances. The parameters θ are subsequently updated and moved in the opposite direction of the gradient, scaled by the hyperparameter η .

SGD guarantees to converge at a global optimum when the function is convex and the learning rate is properly instantiated. When the function is non-convex, that is the case of multi-layer neural networks, SGD proved to be robust and to perform well although it does not guarantee the global optimum detection [6].

3.5.3.3 Stochastic gradient descent with Momentum

The learning stage with stochastic gradient descent can sometimes be highly time-consuming. An effective approach for speeding up the training process is the **momentum** method [75]. Its

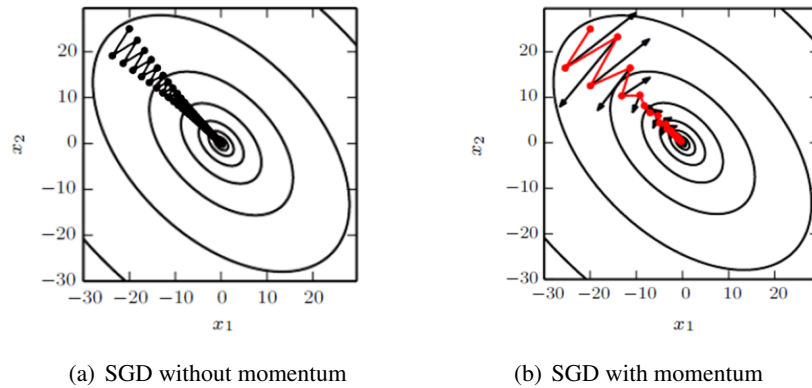


FIGURE 3.9: Comparison between the gradient descent with and without momentum. The momentum gradient path (red line) clearly provides a faster and less oscillating convergence to the optimum with respect to the common gradient descent path (black line) [32].

core principle, beyond the optimization process, is inspired by a physical interpretation. The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction [32] (Fig. 3.9). As effect of its application, the optimization will accelerate along the dimensions where the gradient points in the same direction and will decelerate along the dimensions in which the gradient trajectory is unstable, resulting in a faster convergence and reduced oscillation.

Formally, the momentum algorithm introduces the velocity variable v , that denotes the direction and speed at which the parameters move through the parameter space. It is possible to think of the optimization algorithm as the process of simulating the parameter vectors rolling on the landscape with a certain velocity. In the momentum update, the parameter $\rho \in [0, 1)$ will denote how quickly the gradient will exponentially decay:

$$\begin{aligned}
 v &\leftarrow \rho v - \eta \nabla_{\theta} \left(\frac{1}{n} \sum_{i=1}^n L(f_{\theta}(x_i), y_i) \right) \\
 \theta &\leftarrow \theta + v
 \end{aligned} \tag{3.15}$$

The SGD algorithm with momentum is described in Algorithm 2.

The momentum can provide significant improvements over SGD, especially in terms of computational time. With this simple method, it is possible to speed up the training phase avoiding the unstable oscillatory effect caused by setting high values of learning rates.

3.5.3.4 Stochastic gradient descent with Nesterov Momentum

In 2013, Sutskever et al. [76] proposed an improvement of the momentum method, called **Nesterov momentum**, inspired by Nesterov's accelerated gradient method [77]. Differently from

Algorithm 2 Stochastic gradient descent with momentum

-
- 1: Require: Learning rate η , momentum parameter ρ
 - 2: Require: Initial set of parameters θ
 - 3: **while** Stopping criterion not met **do**
 - 4: Sample a minibatch of m_b examples from the training set $\{x_1, \dots, x_{m_b}\}$.
 - 5: Set $g = 0$
 - 6: **for** $i = 1$ to m_b **do**
 - 7: Compute gradient estimate:

$$g \leftarrow g + \nabla_{\theta} L(f_{\theta}(x_i), y_i).$$

- 8: **end for**
 - 9: Compute gradient estimate: $v \leftarrow \rho v - \eta g$
 - 10: Apply update: $\theta \leftarrow \theta + v$
 - 11: **end while**
-

the standard momentum approach, it evaluates the gradient after the velocity is applied. This sort of correction prevents the model from proceeding too fast and increasing the responsiveness. The resulting update step is defined as follows:

$$v \leftarrow \rho v - \eta \nabla_{\theta} \left(\frac{1}{n} \sum_{i=1}^n L(f_{\theta}(x_i + \rho v), y_i) \right) \quad (3.16)$$

$$\theta \leftarrow \theta + v$$

The SGD algorithm with Nesterov momentum is presented in Algorithm 3.

Algorithm 3 Stochastic gradient descent with Nesterov momentum

-
- 1: Require: Learning rate η , momentum parameter ρ
 - 2: Require: Initial set of parameters θ , initial velocity v
 - 3: **while** Stopping criterion not met **do**
 - 4: Sample a minibatch of m_b examples from the training set $\{x_1, \dots, x_{m_b}\}$
 - 5: Apply intermediate update: $\theta \leftarrow \theta + \rho v$
 - 6: Set $g = 0$
 - 7: **for** $i = 1$ to m_b **do**
 - 8: Compute gradient estimate (at intermediate point):

$$g \leftarrow g + \nabla_{\theta} L(f_{\theta}(x_i), y_i).$$

- 9: **end for**
 - 10: Compute gradient estimate: $v \leftarrow \rho v - \eta g$
 - 11: Apply update: $\theta \leftarrow \theta + v$
 - 12: **end while**
-

While the Nesterov momentum have proved to improve the rate of convergence for convex batch gradient loss functions, it does not guarantee the same behavior for stochastic gradient descent models.

3.5.3.5 AdaGrad

The previous sections (Sec. 3.5.3.3 and 3.5.3.4) presented two different techniques for adapting the update procedure during the gradient computation and consequently speed up the computational time. Following the same concept of adaptation, several methods have been proposed to provide better learning rates during the training phase.

Adagrad [78] was the first adaptive gradient descent model to be proposed in the state of the art for learning rates. It performs learning rate updates depending on the parameters: the more the parameters are frequent, the larger the learning rate update will be, and vice-versa. The algorithm, presented in Algorithm 8.4, adapts the learning rates by scaling them inversely proportional to the square root of the sum of all of their historical squared values. The update rule is defined as:

$$\begin{aligned}\varphi &\leftarrow \varphi + g^2 \\ \theta &\leftarrow \theta - \frac{\eta}{\sqrt{\varphi}}g\end{aligned}\tag{3.17}$$

Differently from the previously presented gradient-optimization approaches, Adagrad considers a different learning rate for each step. Indeed, the update procedures in Algorithm 4 are performed element-wise.

Algorithm 4 AdaGrad

- 1: Require: Global learning rate η
 - 2: Require: Initial set of parameters θ
 - 3: Initialize gradient accumulation variable $\varphi = 0$
 - 4: **while** Stopping criterion not met **do**
 - 5: Sample a minibatch of m_b examples from the training set $\{x_1, \dots, x_{m_b}\}$
 - 6: Apply intermediate update: $\theta \leftarrow \theta + \rho v$
 - 7: Set $g = 0$
 - 8: **for** $i = 1$ to m_b **do**
 - 9: Compute gradient:

$$g \leftarrow g + \nabla_{\theta} L(f_{\theta}(x_i), y_i).$$
 - 10: **end for**
 - 11: Accumulate gradient: $\varphi \leftarrow \varphi + g^2$ (square is applied element-wise)
 - 12: Compute gradient estimate: $\Delta\theta \leftarrow -\frac{\eta}{\sqrt{\varphi}}g$ ($\frac{1}{\sqrt{\varphi}}$ is applied element-wise)
 - 13: Apply update: $\theta \leftarrow \theta + \Delta\theta$
 - 14: **end while**
-

Adagrad shows interesting results for convex optimization problems. However, it has been empirically demonstrated that, for non-convex problems as deep neural networks, the accumulation of squared gradients can result in a premature and excessive decrease of the effective learning rate [56].

3.5.3.6 RMSProp

The main weakness of Adagrad is the accumulation of squared gradients, that keeps growing during the training phase. This behavior can lead the learning rate to assume infinitesimal values, at which point the algorithm is no longer able to acquire additional knowledge [72].

Tieleman and Hinton [79] proposed the **RMSProp** method to solve this issue and consequently perform better on non-convex optimization problems as deep neural network. The key strategy is to use an exponentially decaying average to discard historical values from the extreme past. The RMSProp algorithm, shown in Algorithm 5, introduces the hyperparameter ρ responsible of the length scale of the moving average. The update procedure will change as follows:

$$\begin{aligned}\varphi &\leftarrow \rho\varphi + (1 - \rho)g^2 \\ \theta &\leftarrow \theta - \frac{\eta}{\sqrt{\varphi}}g\end{aligned}\tag{3.18}$$

Algorithm 5 RMSprop

- 1: Require: Global learning rate η , decay rate ρ
 - 2: Require: Initial set of parameters θ
 - 3: Initialize accumulation variable $\varphi = 0$
 - 4: **while** Stopping criterion not met **do**
 - 5: Sample a minibatch of m_b examples from the training set $\{x_1, \dots, x_{m_b}\}$
 - 6: Set $g = 0$
 - 7: **for** $i = 1$ to m_b **do**
 - 8: Compute gradient:

$$g \leftarrow g + \nabla_{\theta} L(f_{\theta}(x_i), y_i).$$
 - 9: **end for**
 - 10: Accumulate gradient: $\varphi \leftarrow \rho\varphi + (1 - \rho)g^2$
 - 11: Compute parameter update: $\Delta\theta \leftarrow -\frac{\eta}{\sqrt{\varphi}}g$ ($\frac{1}{\sqrt{\varphi}}$ is applied element-wise)
 - 12: Apply update: $\theta \leftarrow \theta + \Delta\theta$
 - 13: **end while**
-

RMSProp is one of the most effective gradient-based optimization algorithms for deep neural networks.

3.5.3.7 Adam

An additional method for computing adaptive learning rates for each parameter is called Adaptive Moment Estimation (**Adam**) [80]. This method can be viewed as a combination of RMSProp and momentum, where the exponentially decaying average is computed both on the past gradients (s) and their squares (r). Hence, s is the estimation of the first moment (mean) of the

gradients and r estimates the second moment (uncentered variance) of the gradients:

$$\begin{aligned}\tau &\leftarrow \rho_1 \tau + (1 - \rho_1)g \\ \varphi &\leftarrow \rho_2 \varphi + (1 - \rho_2)g^2\end{aligned}\tag{3.19}$$

where ρ_1 and ρ_2 are decay rates. Since τ and φ are initialized equal to 0, their values can be biased towards 0 especially during the initial training phase. For this reason, Adam algorithm comprehends a bias-correction of the estimates:

$$\begin{aligned}\hat{\tau} &\leftarrow \frac{\tau}{1 - \rho_1} \\ \hat{\varphi} &\leftarrow \frac{\varphi}{1 - \rho_2}\end{aligned}\tag{3.20}$$

After the computation of the bias-corrected first and second moments, the update procedure is derived as follows:

$$\theta \leftarrow \theta - \frac{\eta \hat{\tau}}{\sqrt{\hat{\varphi} + \varepsilon}} g\tag{3.21}$$

where ε is an infinitesimal constant that avoids division by zero.

The Adam method (Algorithm 6) results as a robust response to the choice of hyperparameters, preventing from high biases.

Algorithm 6 Adam

- 1: Require: Step size η
 - 2: Require: Decay rates ρ_1 and ρ_2 , constant ε
 - 3: Require: Initial set of parameters θ
 - 4: Initialize 1st and 2nd moment variables $\tau = 0, \varphi = 0$
 - 5: Initialize timestep $t = 0$
 - 6: **while** Stopping criterion not met **do**
 - 7: Sample a minibatch of m_b examples from the training set $\{x_1, \dots, x_{m_b}\}$
 - 8: Set $g = 0$
 - 9: **for** $i = 1$ to m_b **do**
 - 10: Compute gradient:

$$g \leftarrow g + \nabla_{\theta} L(f_{\theta}(x_i), y_i).$$
 - 11: **end for**
 - 12: $t \leftarrow t + 1$
 - 13: Get biased first moment: $\tau \leftarrow \rho_1 \tau + (1 - \rho_1)g$
 - 14: Get biased second moment: $\varphi \leftarrow \rho_2 \tau + (1 - \rho_2)g^2$
 - 15: Compute biased-corrected first moment: $\hat{\tau} \leftarrow \frac{\tau}{1 - \rho_1}$
 - 16: Compute biased-corrected second moment: $\hat{\varphi} \leftarrow \frac{\varphi}{1 - \rho_2}$
 - 17: Compute update: $\Delta\theta \leftarrow -\frac{\eta \hat{\tau}}{\sqrt{\hat{\varphi} + \varepsilon}} g$ (operation applied element-wise)
 - 18: Apply update: $\theta \leftarrow \theta + \Delta\theta$
 - 19: **end while**
-

3.5.3.8 Adadelta

In 2012, Zeiler [81] proposed an extension of the Adagrad model, called **Adadelta**. As first improvement, Adadelta solves the issue of continual decay of learning rates throughout the training phase. As RMSProp, it considers a fixed window of accumulated past gradients instead of all the historical gradient values, resulting in a better local estimate of the gradient (Eq. 3.18). One of the main drawbacks of the afore-mentioned gradient descent model is the need for a manually selected global learning rate. As it is possible to see from Algorithm 7, Adadelta overcomes this requirement by replacing the learning rate η with the root mean squared error of parameter updates:

$$\begin{aligned}\tau &\leftarrow \rho\tau + (1 - \rho)(\Delta\theta)^2 \\ \theta &\leftarrow \theta - \frac{\sqrt{\tau + \varepsilon}}{\sqrt{\varphi + \varepsilon}}g\end{aligned}\tag{3.22}$$

Algorithm 7 AdaDelta

- 1: Require: Decay rate ρ , constant ε
 - 2: Require: Initial set of parameters θ
 - 3: Initialize accumulation variables $\tau = 0, \varphi = 0$
 - 4: **while** Stopping criterion not met **do**
 - 5: Sample a minibatch of m_b examples from the training set $\{x_1, \dots, x_{m_b}\}$
 - 6: Set $g = 0$
 - 7: **for** $i = 1$ to m_b **do**
 - 8: Compute gradient:

$$g \leftarrow g + \nabla_{\theta} L(f_{\theta}(x_i), y_i).$$
 - 9: **end for**
 - 10: Accumulate gradient: $\varphi \leftarrow \rho\varphi + (1 - \rho)g^2$
 - 11: Compute update: $\Delta\theta \leftarrow -\frac{\sqrt{\tau + \varepsilon}}{\sqrt{\varphi + \varepsilon}}g$ (operation applied element-wise)
 - 12: Accumulate update: $\tau \leftarrow \rho\tau + (1 - \rho)(\Delta\theta)^2$
 - 13: Apply update: $\theta \leftarrow \theta + \Delta\theta$
 - 14: **end while**
-

Adadelta appears robust to a wide variety of configuration choices, such as model architectures, input data types and nonlinearities, demonstrating a good predisposition to be used as optimization algorithm for training neural networks.

Chapter 4

Deep Learning Architectures for Textual Feature Representation

As presented in previous Chapters 2 and 3, Deep Learning models have become incredibly popular in Natural Language Processing, mainly because of their remarkable results in this field and their ability to obtain meaningful distributional representations from symbolic and discrete language units.

This thesis focuses on a specific segment of the research lines in Deep Learning: **unsupervised models for textual Representation Learning**. The motivations beyond this decision have been briefly mentioned in the previous chapters and can be summarized into three aspects presented as follows.

Unsupervised models As highlighted by the historical trends, people are witnessing a new digital era where user-generated content is produced in abundance over disparate Social Media channels. Leveraging this information can be of a great value for companies and institutions. Indeed, there is the need of Machine Learning models, named **unsupervised**, able to extract valuable knowledge from data that have not been manually annotated (a time and effort consuming task that became impossible with the exponential increase of available data). Deep Learning provides several models that can efficiently take advantage of this large amount of data to identify and disentangle their underlying explanatory factors.

Representation Learning The choice of the right set of features used to represent the data is a crucial part of every Machine Learning process, as it can strongly influence the performance both in terms of time and accuracy. Involving human experts in this process is not always efficient due to the time-consuming effort and because sometimes the real explanatory features of

the data cannot be easily identified. Among the various techniques of learning representations, Deep Learning methods proved to be particularly capable of obtaining high-level meaningful representations of the data by learning multiple nonlinear transformations of increasing complexity and abstraction. Beyond the classification performance, the representations obtained with Deep Learning proved to be also able to improve the generalization abilities of Machine Learning models, by providing better results also on unseen data not present in the training set.

Textual Distributed Representation The symbolic and discrete nature of natural language has led to great difficulties in obtaining a valid representation of textual data for solving Machine Learning problems. Since a mathematical representation cannot be directly inferred, several feature extraction procedures have been proposed. In the recent state of the art, the neural network based approaches are the most prominent solution for obtaining a low dimensional meaningful representation of words, or language units in general. The produced embeddings are then able to capture the semantic and syntactical meaning of language into their representation, resulting in better performance for several NLP tasks.

This chapter aims to present the two main methodological trends for learning how to represent textual data with unsupervised models. Section 4.1 introduces **neural networks language models**, along with the most popular **Word Embeddings** algorithms. These algorithms are neural network based models used to map words from a vocabulary to the corresponding vectors of real numbers. They have been widely studied to overcome the issue of natural language discreteness and sparsity, and also to obtain a representation easily manageable by any Machine Learning model. Although the most popular approach for Word Embeddings proposed in [2] does not have the salient characteristic of Deep Learning architectures (i.e. complexity, nonlinearity and multi-level), it is commonly associated with this class of models because it does actually have the Deep Learning intellectual heritage. The greatest insight about the experiment in [2] lies on the result that the authors have discovered that the simplest model returned the most robust and unexpectedly sensible results among many different complex and deep neural network models.

Section 4.2 details the most adopted Deep Learning models for unsupervised Representation Learning, i.e. **Auto-encoders**. These models act as feature learning methods by learning to copy its input to its output. The hidden layer between the input and output layers will then represent the latent factor of the data, as a dimensional reduction method. This representation has proved to facilitate the classification, visualization, communication and storage of data [82]. In the last years, successful results have been achieved in a variety of applications, one of which is Natural Language Processing [83–85].

4.1 Neural Networks Language Model

The most popular contributions of Deep Learning in Natural Language Processing concern **language modeling** (LM). The aim of this task is to assign a probability to any arbitrary sequence of words or other linguistic symbols (e.g., letters, parts of speech, etc.). Beyond the probability of a sequence of words, language models can also infer the likelihood of a given word to follow a sequence of words. Although the performance of LM is still far from the human-level ones, language models are often an important subprocess of many successful applications, such as text information retrieval, document classification, statistical machine translation, etc. [86].

Formally, Goldberg [6] defined the task of language modeling as the problem to assign a probability to any sequence of words $w_{1:T}$, defined as $P(w_{1:T})$. Using the chain-rule of probability, this can be written as:

$$P(w_{1:T}) = P(w_1)P(w_2 | w_1)P(w_3 | w_{1:2})P(w_4 | w_{1:3}) \dots P(w_n | w_{1:T-1}) \quad (4.1)$$

or it can be more concisely expressed as:

$$P(w_{1:T}) = \prod_{t=1}^n P(w_t | w_{1:t-1}) \quad (4.2)$$

where w_t is the t -th word. Essentially, a statistical language model can be represented by the conditional probability of the next word given all the previous ones [1]. When building LM, the difficulty on modeling entire sentences can be considerably reduced by exploiting the *markov-assumption*, which states that closer words in the word sequence are statistically more dependent, by the assumption that the future is independent of the past given the present.

More formally, a k -th order markov-assumption assumes that the next word in a sequence depends only on the last $k - 1$ words:

$$P(w_t | w_{1:t-1}) \approx P(w_t | w_{t-k+1:t-1}). \quad (4.3)$$

The estimate of the probability of a sentence then becomes:

$$P(w_{1:T}) = \prod_{t=1}^T P(w_t | w_{t-k+1:t-1}) \quad (4.4)$$

where w_{t-k+1}, \dots, w_0 are defined to be special padding symbols.

The k -th order markov assumption is the dominant approach for language modeling because it can produce good results for relatively small values of k . This thesis, and this chapter in particular, discusses k -th order language models.

Before discussing the currently most adopted neural network based language models (NNLMs), it is worthwhile to introduce the traditional frequentist-based approaches and to motivate the reason of their recent overtake.

Traditional approaches Traditional techniques for estimating LM parameters are based on corpus count. Assuming a k -order markov property, such as $P(w_t = w^* | w_{1:t-1}) \approx P(w_t = w^* | w_{t-k+1:t-1})$. The aim of LM is to estimate the likelihood of a given word w^* to follow a sequence of words, i.e. $P(w_t = w^* | w_{t-k+1:t-1})$. Let $\#w_{i:j}$ be the count of the sequence of words $w_{i:j} = (w_i, w_{i+1}, \dots, w_{j-1}, w_j)$ in a corpus. The *maximum likelihood estimate* (MLE) of $P(w_t = w^* | w_{t-k+1:t-1})$ can be written as:

$$P_{MLE}(w_t = w^* | w_{t-k+1:t-1}) = \frac{\#w_{t-k+1:t}}{\#w_{t-k+1:t-1}} \quad (4.5)$$

When the training data is not representative of the whole possible events, it can happen that an event $w_{t-k+1:t}$ is never observed in the corpus ($w_{t-k+1:t} = 0$), and consequently its probability will be 0. This will cause a domino effect, i.e. the entire corpus will assume 0 as probability value, because of the multiplicative nature of the sentence probability calculation.

A solution to the sparsity problem is provided by *smoothing techniques*. The simplest idea is to add a smoothing quantity to the counts in order to ensure a probability greater than 0 for every event. This approach is called add- α smoothing [87, 88] and can be formalized as:

$$P_{add-\alpha}(w_t = w^* | w_{t-k+1:t-1}) = \frac{\#w_{t-k+1:t} + \alpha}{\#w_{t-k+1:t-1} + \alpha |V|} \quad (4.6)$$

where $|V|$ is the vocabulary size and $0 < \alpha \leq 1$ is the smoothing parameter.

Despite researchers have contributed with several improved extensions of add- α smoothing [89, 90], this class of approaches has different important drawbacks. Smoothed MLE models cannot easily deal with long-range dependencies, because the introduction of larger k -grams significantly increases the model sparsity and also the time and space complexity. As more important issue, MLE-based language models are not able to generalize over different contexts or domains. For example, as outlined in [6], observing *black car* and *blue car* in the training data does not influence the estimates of *red car* if it has not been seen before.

In spite of their well-known weaknesses, these traditional methods remained the dominant approaches in the state of the art until the advent of neural networks and Deep Learning, which provided better language models over several standard benchmark tasks [2, 91].

Neural Network Language Models Since 2010, Deep Learning provided new solutions for solving the inherent problems of traditional approaches. With this new trend of models, it is

possible to efficiently consider longer dependencies with only a linear increase in the number of parameters and better deal with unseen events.

Bengio et al. [1] summarized the core idea of neural networks language models (NNLMs) as follows:

1. a distributed *word feature vector* or *Word Embeddings* (a real-valued vector in \mathbb{R}^m) is associated with each word in the vocabulary,
2. the joint *probability function* of word sequences is expressed in terms of the feature vectors of these words in the sequence, and
3. the Word Embeddings and the probability function parameters are learned simultaneously.

After the learning phase, the *word feature vector* obtained by NNLMs will reflect the whole aspects of the associated word, e.g. conceptual, semantic, syntactic. Its dimension is usually much smaller than the actual size of the vocabulary, resulting in what is commonly known as *low-dimensional embeddings*. Moreover, as the name neural networks language model reminds, the *probability function*, i.e. the product of conditional probabilities of the next word given the previous ones (Eq. 4.4), is estimated by a multi-layer neural network.

By performing these operations, NNLMs are then able to successfully generalize between different contexts. The core idea, inspired by the distributional hypothesis [24], is that words that appear in a similar context should have a similar vector representations. As a practical implication, by observing, for example, that the words *blue*, *green*, *red*, *black*, etc. appear in a similar context, the NNLM will associate them similar Word Embeddings. Therefore, NNLM is able to generalize from “*blue car*” to “*red car*” also if the event “*red car*” was never observed.

This approach is different from the popular state of the art models as Brown Clustering [25] or Latent Semantic Indexing [92] because (1) the features composing Word Embeddings are not deterministic, but rather real values that combined as vectors better reflect similarities between words and (2) the learning process is aimed at maximizing the probability distribution of word sequences instead of co-occurrences [1].

4.1.1 Neural Probabilistic Language Model

The fundamental neural network model described in this section has been proposed by Bengio et al. [1].

Given a training set as a sequence w_1, \dots, w_T of words $w_t \in V$, where the vocabulary V is a large but finite set. The objective is to learn a language model $f(w_t, \dots, w_{t-k+1}) = P(w_t | w_{1:t-1})$,

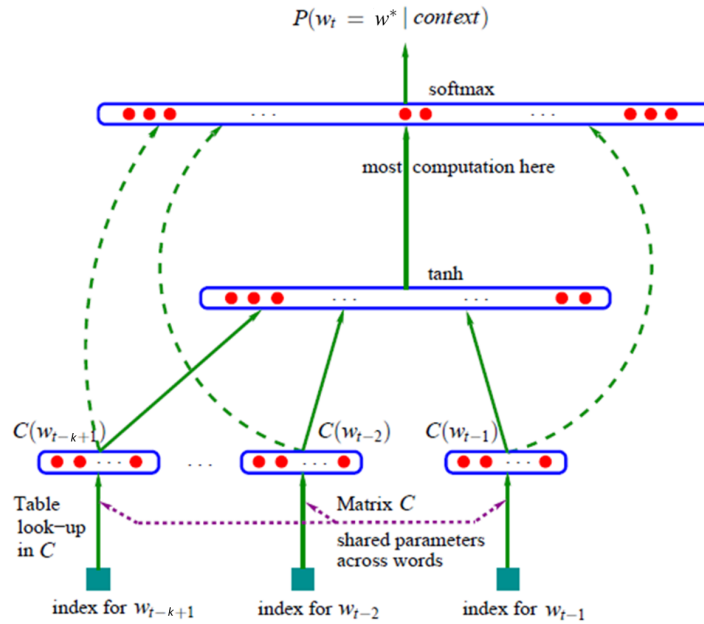


FIGURE 4.1: Neural language model proposed by Bengio et al. [1].

that can provide good estimates even without sample instances. The only constraint of the probabilistic LM is that the sum of the probabilities of a given sentence that ends with different words should sum to 1, that is for any choice of $w_{1:t-1}$, $\sum_{w^* \in V} f(w^*, w_{t-1}, \dots, w_{t-k+1}) = 1$, with $f > 0$. The model of the joint probability of word sequences can be then obtained by the product of these conditional probabilities.

The function $f(w_t, \dots, w_{t-k+1}) = P(w_t | w_{1:t-1})$ can be separately examined in two parts:

- The Word Embeddings are expressed as a mapping C that associatea to each word in the vocabulary $w^* \in V$ a real vector $C(w^*) \in \mathbb{R}^m$.
- The probability function is defined as a function $g(\cdot)$ that maps a sequence of Word Embeddings $(C(w_{t-1}), \dots, C(w_{t-k+1}))$ to a conditional probability distribution over words in the vocabulary V with the aim of predicting the next word w_t . This distribution can be practically seen as a vector where each element estimates the probability $P(w_t = w^* | w_{1:t-1})$ over different w^* :

$$f(w^*, w_{t-1}, \dots, w_{t-k+1}) = g(C(w^*), C(w_t), \dots, C(w_{t-k+1})) \quad (4.7)$$

The function g can be expressed as a neural network model (Fig. 4.1), following the notions provided in Chapter 3:

$$\begin{aligned}\mathbf{x} &= [C(w_{t-1}), \dots, C(w_{t-k+1})] \\ \mathbf{h}_1 &= \tanh(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \\ \hat{\mathbf{y}} &= \text{LM}(w_{1:T}) = \text{softmax}(\mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2)\end{aligned}\tag{4.8}$$

where C , \mathbf{W}_1 , \mathbf{W}_2 , \mathbf{b}_1 and \mathbf{b}_2 are the model parameters and \tanh and softmax are the activation functions used in the neural probabilistic language model proposed in [1].

The aim is to obtain the parameter set $\theta = (C, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2)$ that maximizes the objective function, which in this case is the *penalized log-likelihood*:

$$\mathcal{L}_{NNLM}(\theta) = \frac{1}{T} \sum_{t=1}^T \log f_{\theta}(w_t, w_{t-1}, \dots, w_{t-k+1}) + \lambda \mathcal{R}(\theta)\tag{4.9}$$

where $\mathcal{R}(\theta)$ is the regularization term and λ is the hyperparameter balancing the contribution of the regularization.

The neural model proposed by Bengio et al. [1] provided several important advantages that helped them to be considered the state of the art reference for language modeling.

Beyond the overcoming results, the afore-presented model is able to **linearly scale** with the size of vocabulary $|V|$ and also with k , allowing the model to efficiently learn from larger corpora and considering longer word dependencies. As for neural network models, the nonlinear transformations permit the presented model to **disentangle the factors of variation** of the input data. Finally, unseen words or sequences of words are a minor problem because of the good generalization abilities of the model across contexts and domains. This can sometimes be a disadvantage with respect to traditional methods, as it is not always true that similar words can be replaced in any context, although it is often the case. An example, following the previous ones, can be the probability assigned to the sequence of words “*purple horse*”, which can result as highly probable because of the similarity of “*purple*” with other words such as “*black*” or “*brown*”.

These remarkable properties, together with the unsupervised nature of the NNLM proposed by Bengio et al. were the main reasons why several successful methods took inspiration from it: the Collobert and Weston algorithm [27] and the Word2vec algorithms [2, 28]. These approaches are discussed in the next sections.

4.1.2 Collobert and Weston

In 2008, Collobert and Weston [27] developed a unified architecture able to generalize among several NLP tasks, based on neural networks language models. They improved the model introduced by Bengio et al. at two levels. First, they took advantage of the whole word context by considering the window surrounding a word (i.e. $P(w_t | w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2})$), instead of analysing only the k -gram on its left (i.e. $P(w_t | w_{t-1}, w_{t-2}, w_{t-3}, w_{t-4})$). Second, they changed the objective function by moving to a *two-class classification task*: if the word in the middle of the input window is related to its context or not. Following the revised task goal, the training data are then composed by positive examples, a sequence of words in the corpus, and by negative examples, whereby the middle word of a sequence is replaced by a random word. This simple change avoids computationally expensive normalization that is mandatory for probabilities prediction tasks.

More formally, given a target word w^* and its context $c_{1:k}$, after applying the mapping C for obtaining the Word Embeddings, the model aims to predict a score $s(w^*, c_{1:k})$ of the word-context pairs. This problem can be expressed as a neural network model as follows:

$$\begin{aligned} \mathbf{x} &= [C(w^*), C(c_1), \dots, C(c_k)] \\ \hat{\mathbf{y}} &= s(w^*, c_{1:k}) = g(\mathbf{W}\mathbf{x} + \mathbf{b}). \end{aligned} \quad (4.10)$$

Since the model is related to a classification task, the loss function corresponds to the misclassification error:

$$L_{CW}(w^*, c, w') = \max(0, 1 - (s(w^*, c_{1:k}) - s(w', c_{1:k}))), \quad (4.11)$$

where w' is a random word. This loss is then computed for each word-context pair, sampling a random word w' for each step. As the NNLM presented in the previous section, the aim is to find the best parameter set $\theta = (C, \theta_{NN})$ that minimizes the misclassification error over the training set.

The word feature vectors, or Word Embeddings, obtained by the training procedure of the Collobert and Weston's NNLM provided impressive results. For this reason, this work established Word Embeddings as a powerful tool for NLP tasks, leading to a proliferation of studies in this field.

4.1.3 Word2vec

The widely popularization of Word Embeddings was arguably due to Mikolov et al. [2, 28] that positively revolutionized the NNLM research with the introduction of **Word2vec** model. The main reason for the disruptive advent of Word2vec regards the fact that previous works

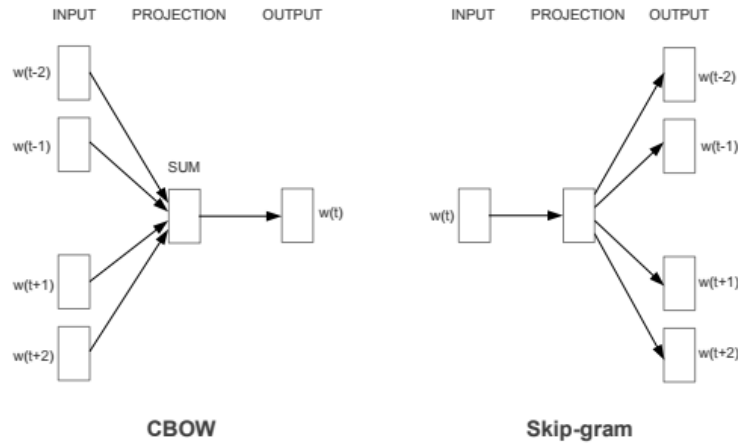


FIGURE 4.2: CBOW architecture on the left and Skip-gram architecture on the right [2].

were significantly more computationally expensive during the training phase. Word2vec model comprises two different model architectures (Figure 4.2) for learning distributed representations of words. The proposed models were constructed in order to try to minimize computational complexity. In the literature, Deep Learning has attracted a lot of attention because of its ability to capture nonlinear transformations of input data, via the composition of concepts of increasing complexity. Against the trends, Mikolov et al. decided to explore simpler models that might not be able to represent the data as precisely as deep neural networks, but can possibly be trained on much more data efficiently.

Continuous Bag-of-Words Model The first proposed architecture is called Continuous Bag-of-Words model (CBOW) and it aims to compute the conditional probability of a target word given the context words surrounding it, across a window of size k , as in [27]. The name of the model is due to the fact that, unlike the standard bag-of-words model, it uses a continuous distributed representation of the context. CBOW is implemented as a fully connected neural network with only one hidden layer. Differently from standard NNLMs, the nonlinear hidden layer is removed and the projection layer is shared for all the words. More formally, given a sequence of words $w_{1:t}$, the objective is to maximize the average log probability [28]:

$$\mathcal{L}_{CBOW} = \frac{1}{T} \sum_{t=1}^T \sum_{-k \leq j \leq k, j \neq 0} \log P(w_t | w_{t+j}) \quad (4.12)$$

where k is the size of the context window. As in traditional language models, the inclusion of longer dependencies (high value of k) can provide better results, at the expenses of a higher complexity.

Continuous Skip-gram Model The second architecture is similar to CBOW but with a completely opposite objective. Skip-gram model aims to predict the surrounding context words given the central target word [93]. Essentially, this model assumes that the elements in the context are independent from each other, treating them as different contexts. Despite this strong assumption, it has been proved that this architecture can be very effective and it has been successfully used [6].

As CBOW, the Skip-gram model objective function can be written as:

$$\mathcal{L}_{\text{Skip-gram}} = \frac{1}{T} \sum_{t=1}^T \sum_{-k \leq j \leq k, j \neq 0} \log P(w_{t+j} | w_t) \quad (4.13)$$

where k is the size of the context window.

Optimization objectives The last layer of Word2vec architectures is characterized by the computation of *softmax* functions, unfortunately this requires the normalization of word vector probabilities for each target word, that is computationally prohibitive in most real-world applications. For this reason, two different optimization objectives have been proposed in [28]: Hierarchical Softmax and Negative-Sampling.

Hierarchical Softmax [94] is a computationally efficient approximation of the full softmax function computed in Word2vec. This method makes use of a binary tree for structuring the softmax computations, where the probability of each word is computed as the product of branch selection decisions. The efficiency is obtained when computing the probability of a single word, because, instead of evaluating W output nodes, it is possible to evaluate only $\log_2(W)$ nodes.

The second optimization approach is *Negative Sampling*, which is an approximation of Noise Contrastive Estimation [95]. Similar to the Collobert and Weston [27] model, the aim is to correctly classify word-context pairs from random-generated ones. The training process is then performed by approximating the normalization in the denominator of the softmax, resulting in increasing computational efficiency.

4.2 Auto-encoder

Auto-encoders have attracted a lot of attention in recent years as a building block of Deep Learning. As efficient unsupervised models for Representation Learning, Auto-encoders became very popular also because of the incredibly large amount of unlabeled data that emerged from the World Wide Web. The core component is a neural network that tries to reconstruct its input at the output layer.

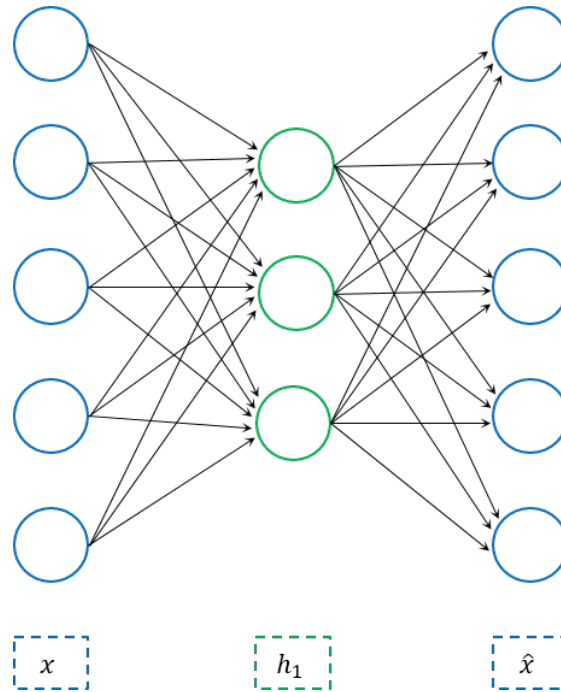


FIGURE 4.3: Auto-encoder architecture.

An Auto-encoder initially consists of a feature-extracting function in a specific parameterized closed form, called **encoder**, denoted as f_{θ} that allows the straightforward and efficient computation of a feature vector $\mathbf{h} = f_{\theta}(\mathbf{x})$ from an input \mathbf{x} . For each instance x_i from a set of data $\mathbf{X} = \{x_1, \dots, x_n\}$, the encoder function is defined as:

$$h_i = f_{\theta}(x_i) \quad (4.14)$$

where h_i will be the feature vector representation that codes the input x_i .

The other crucial component is the **decoder** function, which is a closed form parameterized function g_{θ} that maps the feature space back into input space, producing a reconstruction $\hat{\mathbf{x}}$:

$$\hat{\mathbf{x}} = g_{\theta}(\mathbf{h}) \quad (4.15)$$

The general structure of Auto-encoders is illustrated in Figure 4.3. The Auto-encoder training process consists in finding the parameter set θ that minimizes the reconstruction error:

$$\mathcal{L}_{AE}(\theta) = \frac{1}{n} \sum_{i=1}^n L(x_i, g_{\theta}(f_{\theta}(x_i))) \quad (4.16)$$

where x_i is a training example. Error minimization is usually carried out by stochastic gradient descent methods.

Indeed, Auto-encoders can be expressed as a multi-layer neural network:

$$\begin{aligned}\mathbf{h} &= f_{\theta}(\mathbf{x}) = a_f(\mathbf{W}\mathbf{x} + \mathbf{b}_f) \\ \hat{\mathbf{x}} &= g_{\theta}(\mathbf{h}) = a_g(\mathbf{W}'\mathbf{h} + \mathbf{b}_g)\end{aligned}\tag{4.17}$$

where a_f and a_g are the encoder and decoder activation functions (typically *sigmoid* or *hyperbolic tangent* for nonlinearity, or the *identity function* if staying linear). The set of parameters is $\theta = \{\mathbf{W}, \mathbf{b}_f, \mathbf{W}', \mathbf{b}_g\}$ where \mathbf{b}_f and \mathbf{b}_g are called encoder and decoder bias vectors, and \mathbf{W} and \mathbf{W}' are the encoder and decoder weight matrices. The choice of the activation functions and the loss function largely depends on the input domain nature.

4.2.1 Stacked Auto-encoder

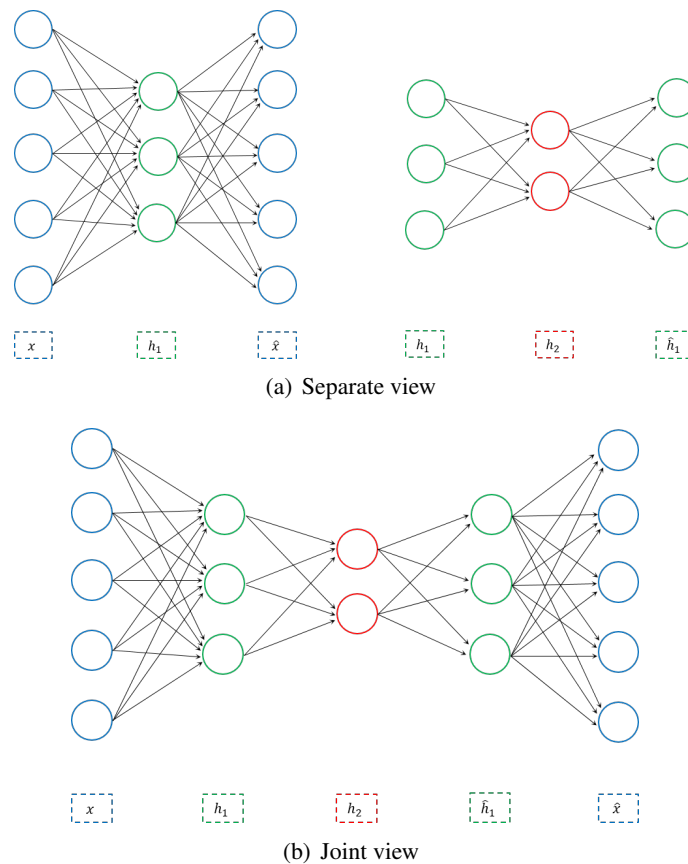


FIGURE 4.4: Stacked Auto-encoder architecture with 2 hidden layers.

As the multi-layer feedforward architecture presented in Section 3.3.2, also the Auto-encoder models can be stacked to produce the so-called **stacked Auto-encoder**. The main difference consists on which layer is given as input to the next stacked model: in a multi-layer feedforward neural network the *output* of each layer is given as input to the next one, while stacked Auto-encoders passes as input the *hidden* layer.

The stacking procedure of a 2-layers stacked Auto-encoder is shown in Figure 4.4. The first Auto-encoder is constructed as presented in the previous section, i.e. the input x is encoded in the hidden layer h_1 and then decoded for its reconstruction. Then, the activation features h_1 are given as input to the second Auto-encoder for obtaining a secondary level of features h_2 (Fig. 4.4(a)). Finally, the layers can be combined together (Fig. 4.4(b)) to form a stacked Auto-encoder with 2 hidden layers.

Each of the Auto-encoder variations that will be presented in the next sections can be used as core layers for a stacked implementation.

4.2.2 Regularized Auto-encoder

While the idea to copy the input to the output may appear pointless, the true value is in the hidden layers between the input and its reconstruction.

A common approach for obtaining meaningful feature representation, as any dimensionality reduction technique, is to use a bottleneck, i.e. to constrain the hidden layer h to have a lower dimension than the input x . In this case, the Auto-encoder takes the name of **undercomplete Auto-encoder**. Reducing the dimension of the hidden layer forces to prioritize the aspects of the input that should be copied, resulting in a more useful representation and better generalization abilities over new data. Another interesting approach is the **overcomplete Auto-encoder**, where the hidden layer dimension is greater than the input data size.

Both of these two variations can encounter a critical problem. When too much capacity is given to the encoder and decoder functions, the copying task can be perfectly performed without having extracted any meaningful information about the distribution of the data. For this reason, alternative **regularized Auto-encoders** have been proposed to solve this issue by constraining the learning process. The idea is to use a loss function to stimulate the learning of other properties besides the reconstruction ability. Possible properties can be *representation sparseness* and *noise robustness* as presented in the next sections. In conclusion, regularized Auto-encoders are models that are able to learn meaningful representations of the input data, even if the model capacity is high enough to learn a trivial identity function.

4.2.3 Sparse Auto-encoder

A **sparse Auto-encoder** (SAE) [96, 97] is a form of regularized Auto-encoder that considers a sparsity penalty $\Omega(h)$ on the code layer h in addition to the reconstruction error:

$$\mathcal{L}_{SAE}(\theta) = \frac{1}{n} \sum_{i=1}^n L(x_i, g_{\theta}(f_{\theta}(x_i))) + \Omega(h) \quad (4.18)$$

With this constraint, most of the hidden neurons will tend to produce very small activations, allowing the use of a large number of hidden units.

4.2.3.1 k -sparse Auto-encoder

Although sparse Auto-encoders have shown significant improvements in the state of the art, these methods are sometimes not guaranteed to produce sparse representations for each input. Indeed, Makhzani and Frey [98] proposed a fast and efficient improvement of sparse Auto-encoder. This model, called **k -sparse Auto-encoder (KsAE)**, is basically an Auto-encoder where only the k highest activities in the hidden layers are kept.

During the feedforward phase, after computing the hidden layer \mathbf{h} , the k largest hidden units are kept and the others are set to zero. This results in a vector of activities with the support set defined as $\Gamma = \text{supp}_k(\mathbf{h})$. This selection step acts as a regularizer that prevents the use of an excessively large number of hidden units when reconstructing the input. At testing time, instead of using the k largest neurons as features, Makhzani and Frey considered the αk largest hidden units, where $\alpha \geq 1$ is selected using validation data, as soft assignments resulted in better classification performance. Algorithm 8 summarizes the fundamental steps.

Algorithm 8 k -sparse Auto-encoder

- 1: Perform the feedforward phase and compute $\mathbf{h} = a_f(\mathbf{W}\mathbf{x} + \mathbf{b}_f)$.
- 2: Find the k largest activations of \mathbf{h} and set the rest to zero.

$$\mathbf{h}_{(\Gamma)^c} = 0 \quad \text{where} \quad \Gamma = \text{supp}_k(\mathbf{h}).$$

- 3: Compute the reconstruction error using the sparsified \mathbf{h} .

$$\mathcal{L}_{KsAE}(\theta) = \frac{1}{n} \sum_{i=1}^n L(x_i, g_{\theta}(\mathbf{h})).$$

- 4: Backpropagate the error through the k largest activations defined by Γ and iterate.

Inference

- 5: Encode input data $\mathbf{h} = a_f(\mathbf{W}\mathbf{x} + \mathbf{b}_f)$.
Find its αk largest activations and set the rest to zero

$$\mathbf{h}_{(\Gamma)^c} = 0 \quad \text{where} \quad \Gamma = \text{supp}_{\alpha k}(\mathbf{h}).$$

4.2.4 Denoising Auto-encoder

Another regularization approach for avoiding the learning of the identity function is provided by **denoising Auto-encoders (DAE)** [35]. In this case, the model learns a reconstructed input

corrupted by a form of noise. This denoising-based approach is motivated by the following two assumptions:

- a higher level representation is expected to be robust under corruptions of the input data;
- a model able to correctly perform the denoising task should extract much more informative features.

Formally, the objective function optimized by a DAE is:

$$\mathcal{L}_{DAE}(\theta) = \frac{1}{n} \sum_{i=1}^n L(x_i, g_{\theta}(f_{\theta}(\tilde{x}_i))), \quad (4.19)$$

where \tilde{x} is a corrupted copy of x . Indeed, in order to learn the copying function, denoising Auto-encoders must undo this corruption. State of the art studies demonstrate that corruption levels generate much better features, resulting in improved classification.

4.2.4.1 Marginalized Stacked Denoising Auto-encoder

Vincent et al. [37] proposed a novel stacked version of denoising Auto-encoders, called **Stacked Denoising Auto-encoder** (SDA). This layered version of the model essentially works as presented in Section 4.2.1, where each denoising Auto-encoder layer takes as input the encoded output provided by the previous one.

In order to reduce the high computational cost of SDA, the authors in [36] further developed a variant called **marginalized Stacked Denoising Auto-encoder** (mSDA). mSDA is able to preserve the strong feature learning capabilities of SDA while providing several speed-ups, i.e. fewer meta-parameters, faster model-selection and layer-wise convexity.

Given an input $\mathbf{X} = \{x_1, \dots, x_n\}$ and its corrupted version $\tilde{\mathbf{X}} = \{\tilde{x}_1, \dots, \tilde{x}_n\}$, where the features of the input space \mathbb{R}^d are randomly removed with a given probability p . The main goal is to reconstruct the corrupted input with a linear mapping $\mathbf{W} : \mathbb{R}^m \rightarrow \mathbb{R}^m$, that minimizes the objective function:

$$\mathcal{L}_{mSDA}(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n L(x_i, \mathbf{W}\tilde{x}_i), \quad (4.20)$$

where in [36], the squared reconstruction loss is used as per-instance loss function L . The solution of Equation (4.20) can be expressed as the closed-form solution for ordinary least squares [99]:

$$\mathbf{W} = \mathbf{PQ}^{-1} \quad (4.21)$$

where $\mathbf{Q} = \tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top$ and $\mathbf{P} = \mathbf{X}\tilde{\mathbf{X}}^\top$. When the denoising transformation of \mathbf{W} is computed, it would be ideal to consider all possible corruptions of all possible inputs. As n becomes very large, the matrices \mathbf{P} and \mathbf{Q} converge to their expected values $E[\mathbf{Q}]$ and $E[\mathbf{P}]$ by the weak law of large numbers. In the limit case, where $n \rightarrow \infty$, it is possible to derive their expectations and express the corresponding mapping for \mathbf{W} in closed form as:

$$\mathbf{W} = E[\mathbf{P}]E[\mathbf{Q}]^{-1} \quad (4.22)$$

The marginalized stacked denoising Auto-encoder is composed of m closed-form denoising layers.

4.2.5 k -Competitive Auto-encoder

Although all the afore-mentioned Auto-encoder models proved to be effective for Natural Language Processing tasks, they might incur in the problem of learning trivial representations. The motivations behind this issue are related to the intrinsic characteristics of natural language text (Sec. 2.1): large vocabulary and sparseness. While the vocabulary can comprise millions of words, the commonly used representations are usually characterized by only a small percentage of non-zero entries (2%). Moreover, the word distribution is not the same as any given data. Natural language text follows the Zipf's law distribution, which means that most of the occurrences is related to low-frequency words. Indeed, Chen and Zaki [100] proposed a novel **k -competitive Auto-encoder for text** (KATE) able to better deal with all the difficulties brought by natural language. The core idea consists of competitive learning among the neurons of the Auto-encoder.

In the feedforward phase, only the most competitive k neurons of each layer are considered and those k "winners" further incorporate the aggregate activation potential of the remaining inactive neurons. This reallocation process is defined as *k -competitive_layer*. In this way, each hidden neuron of each layer will become specialized in recognizing specific data patterns. After the model is trained, each hidden neuron is distinct from the others and no competition is needed at inference time. The main steps of the method are described in Algorithm 9.

The KATE model is specifically implemented with a_f as *tanh* function and a_g as *sigmoid* function. Denoting \hat{x} the output of the model, i.e. $\hat{x}_i = g_\theta(f_\theta(x_i))$, the per-instance loss function L is associated to the *binary cross-entropy*:

$$L(x_i, \hat{x}_i) = -[x_i \log(\hat{x}_i) + (1 - x_i) \log(1 - \hat{x}_i)]. \quad (4.23)$$

Algorithm 9 k -Competitive Auto-encoder

Training

- 1: Perform the feedforward phase and compute $\mathbf{h} = a_f(\mathbf{W}\mathbf{x} + \mathbf{b}_f)$.
- 2: Apply the k -competition $\hat{\mathbf{h}} = k\text{-competitive_layer}(\mathbf{h})$.
- 3: Compute the reconstruction error:

$$\mathcal{L}_{KATE}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n L(x_i, g_{\boldsymbol{\theta}}(\hat{\mathbf{h}})).$$

- 4: Backpropagate the error and iterate.

Inference

- 5: Encode input data $\mathbf{h} = a_f(\mathbf{W}\mathbf{x} + \mathbf{b}_f)$.
-

As it can be noted, the KATE model operations are very similar to the k -sparse algorithm (Sec. 4.2.3.1). However, they differ in two aspects: KsAE forces sparsity both in training and testing time, while KATE focuses on specializing each neuron during the training phase and does not perform any competition in testing; KsAE suffers from the problem of “dead” neurons for low values of k that can generate problems during back-propagation, KATE, instead, does not face this issue because it does not involve any zero-value assignments.

As a recently proposed method, KATE has shown very impressive results on a variety of methods on many different NLP tasks, demonstrating the ability to learn semantically meaningful representations of words, documents and topics.

Chapter 5

Deep Learning Representation for Making Sense of User-Generated Content

With the advent of Social Media, people saw new and more engaging opportunities for interacting and communicating. The study presented in [101] posits different motivations that can bring anyone to relate with user-generated content: by consuming, by participating, and by producing. Consuming takes place by watching, reading, or viewing contents for entertainment or for obtaining information such as opinions. Participation includes social interactions (user-to-user) and community development (user-to-content), for example by the actions of posting a comment, sharing a content, or approving the post of another user. Finally, producing implies the actual personal content production by an individual, usually for self-expression and self-actualization. With this background in mind, the big potential that **making sense of user-generated content** can provide for supporting information interpretation and decision making over large-scale, dynamic media streams is only remotely conceivable[102].

The crucial steps for making sense of user-generated content are the identification on what the text is talking about, by *extracting* and *disambiguating* the named entities involved (e.g. with respect to DBpedia), and the understanding of the *opinion* that the user is expressing about these entities, also dealing with *irony*. This is an attractive problem given the great challenges that dealing with large amount of unstructured textual data raise (Sec. 2.1) and, at the same time, it is a major opportunity for individuals, companies and institutions.

An example of the process of making sense of user-generated content is given in Figure 5.1 where the reference text is a message posted on a social media platform and states “@EmmaWatson no1 can play hermione better than u”. The main difficulties in analyzing this sentence, which is a typical social media content, are clearly evident, even at first sight. First,

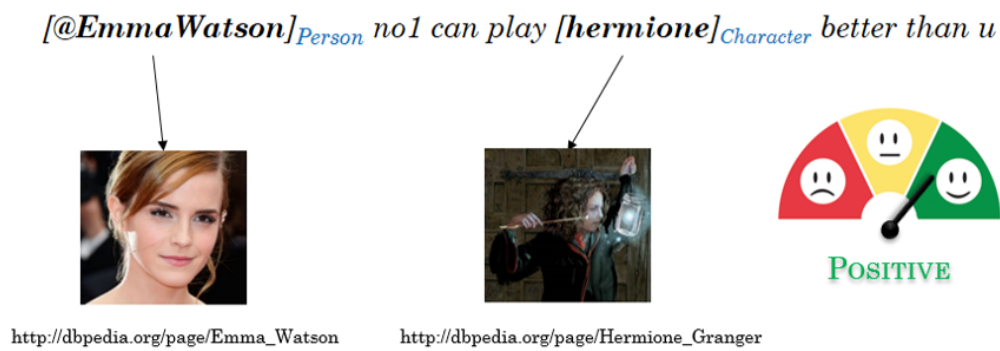


FIGURE 5.1: Example of the process of making sense of user-generated content.

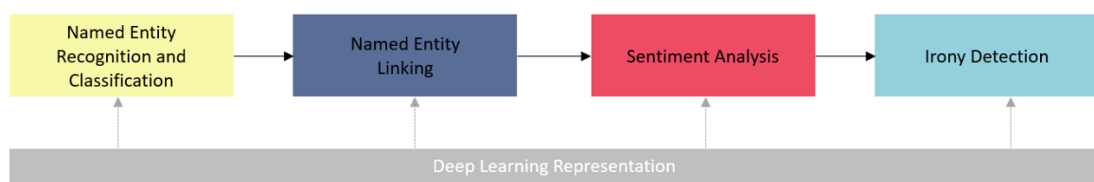


FIGURE 5.2: Proposed framework.

abbreviations and ambiguities can strongly interfere with the correct meaning understanding. While it is easier to conceive that *u* refers to the pronoun *you*, associating the abbreviation *no1* to the word *no one* instead of *number one* is more challenging. If one has no knowledge about the domain context of this sentence, as it is the case of a computational machine, the word *hermione* can assume uncounted meanings, especially if associated to the verb *play*, e.g. the title of a song, a board game, or a character in a movie. Beyond the language issues, a system should also consider the explicit symbols that appear in a text and can be platform-dependent, as hashtags and mentions. For example, by tracing back the mention *@EmmaWatson* to the user, it is possible to retrieve that, as a verified account, the user is related to an authentic person of public interest, resulting in an easier linking of the mere word with the person in the real world.

In order to address these challenges on *making sense* of user-generated content, this chapter proposes novel **Natural Language Processing** approaches exploiting innovative **Deep Learning Representation** models as depicted in Figure 5.2.

The first step consists of identifying named entities in a given text and of classifying them in a pre-defined domain ontology such as persons, organizations, locations. This process is known as **Named Entity Recognition and Classification** (NER) and it is fundamental for understanding which entities are involved in a text. Regarding the previous example, finding that *@EmmaWatson* is an entity of type *Person*, combined with the information that *hermione* is an entity of type *Character*, can have a strong impact on framing this sentence in the domain context of movies. The subsequent task is called **Named Entity Linking** (NEL), i.e. the disambiguation of the identified named entities by associating them to unambiguous units, as for example resources in

a Knowledge Base. As shown in Figure 5.1, a good NEL system should be able to link the entity mention *@EmmaWatson* to the actress named Emma Watson, and the mention *hermione* to the character that she interpreted in the Harry Potter movies. The ability of correctly linking mentions to unambiguous resources in a Knowledge Base can be of great importance, for example, when searching for information about a product or a person, it is not desirable to obtain results that are not related to the searched query (e.g. the company *Ford* is not interested in retrieving messages about the actor Harrison *Ford*).

One of the most investigated tasks, because of the great value for companies, is **Sentiment Analysis**. This task permits to understand the opinion polarity that the user is expressing in a text, usually specified in three classes: positive, negative and neutral. The given example is a case of a positive sentence with respect to the actress Emma Watson, as it expresses a compliment to her acting performance. The task of predicting the sentiment of users has become inevitably important for real-world applications, as social media platforms are viewed as new channels for marketing and for fostering trust of potential customers [103]. Beyond the mere commercial applications, analysis of social media messages can provide important insights in several domains, such as election results [104], political movements [105], and health issues [106].

When dealing with the sentiment polarity of a sentence, the most challenging factor is the presence of figurative language such as irony, or sarcasm [107, 108]. For this reason, the investigated framework includes the task of **Irony Detection**, in order to further improve and eventually refine the polarity of the opinion. The major implication of irony, or sarcasm, regards the fact that its presence (that can be expressed by just a word or a symbol) can completely revert the message polarity. For example, by adding the hashtag word *#mostannoyingcharacter* at the end of the sentence, the sentiment polarity would become negative. A more impressive example is when an ironic message expresses a negative opinion using only positive words, e.g. “*It seems like I have to study on my vacation. That’s great. Awesome. And oh, so very fun.*”.

In summary, by incorporating the Deep Learning high-level representation into the Natural Language Processing models, this thesis aims at improving the generalization abilities of the models underlying the afore-mentioned tasks by leveraging the large amount of unlabeled data over a wide variety of domains and to increase the sensibility of these models about the perception of the words as expression of multiple properties and not as flat and discrete symbols.

Following, Section 5.1 proposes a novel approach for *adapting* the classification of NER systems to a new ontology over generic types without the need of a slow re-training of the model. Then, Section 5.2 presents an *unsupervised* model for Named Entity Linking exploiting a novel heterogeneous representation space characterized by more *meaningful similarity measures* between words and named entities obtained by Word Embeddings. Section 5.3 addresses the problem of Sentiment Analysis by combining Deep Learning and Ensemble Learning where little or no training data is available in the studied domain, that nowadays is a common problem

given the great discrepancy between manually labeled data and available ones. Finally, Section 5.4 investigates the problem of Irony Detection, since it is one of the most challenging and critical steps towards the correct classification of the polarity sentiment of a text, especially if it has been created in a Social Media environment. The presented novel unsupervised irony detection model is then able to *generalize across different domains*, with the contribution of Word Embeddings representation, and to *leverage the unlabeled data* in an unsupervised way, by exploiting an unsupervised probabilistic model.

5.1 Named Entity Recognition and Classification

Named Entity Recognition and Classification is one of the key Information Extraction (IE) tasks, which is concerned with identifying **entity mentions**, which are text fragment(s) denoting real-world objects, from unstructured text and classifying them according to a given domain classification hierarchy/**ontology**. Extracting valuable information from user-generated content in the form of entities mentions, events and relations is of utmost significance for knowledge discovery from natural language text.

Given the example sentence of Figure 5.1,

“@EmmaWatson no1 can play hermione better than u”,

the process of named entity recognition will identify the named entities as:

“[@EmmaWatson] no1 can play [hermione] better than u”.

Consequently, the named entity classification process will assign a class to the entity mentions:

“[@EmmaWatson]_{Person} no1 can play [hermione]_{Character} better than u”.

In the last years, several research studies towards Information Extraction have been proposed, giving leeway to the emergence of numerous academic and commercial NER systems. In Table 5.1, several available NER systems have been listed together with their associated *generic ontologies*, i.e. ontologies aimed at capturing general knowledge about the world by providing basic notions and concepts for things [113].

Although all the reported NER systems make use of *generic ontologies*, from Table 5.1 it is evident that there are considerable differences among them. This is motivated by the fact that, because of varying application scenarios and/or requirements, different NER systems use different entity classification schemas/ontologies to classify the discovered entity mentions into entity types. From the generic ontologies listed in Table 5.1, it is possible to derive several considerations:

TABLE 5.1: Different commercial and academic Named Entity Recognition and Classification systems with their corresponding Generic Types.

NER System	Website	Generic Ontology
OSU Twitter NLP Tools [109]	https://github.com/aritter/twitter_nlp/	Band, Company, Facility, Geo-Location, Movie, Other, Person, Product, Sportsteam, TVshow
NERD [110]	http://nerd.eurecom.fr/	Thing, Amount, Animal, Event, Function, Location, Organization, Person, Product, Time
Stanford NER [111]	https://nlp.stanford.edu/software/CRF-NER.shtml	Person, Organization, Money, Percent, Location, Date, Time
DBpedia Spotlight [112]	https://www.dbpedia-spotlight.org/	Thing, CreativeWork, Event, Language, Organization, Person, Place, Product, Unknown
Dandelion API	https://dandelion.eu/	Person, Works, Organisations, Places, Events, Concepts
Google Cloud Natural Language	https://cloud.google.com/natural-language/	Person, Consumer Good, Organization, Event, Location, Other
IBM Natural Language Understanding	https://www.ibm.com/watson/services/natural-language-understanding/	Anatomy, Award, Broadcaster, Company, Crime, Drug, EmailAddress, Facility, GeographicFeature, HealthCondition, Hashtag, IPAddress, JobTitle, Location, Movie, MusicGroup, NaturalEvent, Organization, Person, PrintMedia, Quantity, Sport, SportingEvent, TelevisionShow, TwitterHandle, Vehicle
Ambiverse	https://www.ambiverse.com/natural-language-understanding-api/	Person, Location, Organization, Event, Artifact, Other, Unknown
Bitext	https://www.bitext.com/	Person name, Car license plate, Place, Phone number, Email address, Company/Brand, Organization, URL, IP address, Date, Hour, Money, Address, Twitter hashtag, Twitter user, Other alphanumeric, Generic
MeaningCloud	https://www.meaningcloud.com/	Event, ID, Living Thing, Location, Numex, Organization, Person, Process, Product, Timex, Unit, Other
Ingen.io	https://ingen.io/	Person, Organization, Location, Geo Political Entity, Misc, Event, Structure, Category, Lang, Artwork
Rosette®	https://www.rosette.com/	Location, Organization, Person, Product, Title, Nationality, Religion, Identifier, Temporal
Thomson Reuters Open Calais™	http://www.opencalais.com/	Company, Person, Geography, Industry Classifications, Topics, Social Tags, Facts, Events

- Only the type *Person* is equivalently reported in all the generic ontologies;
- There are few types (e.g. *Location*) that are present in all the ontologies but with different associated names (e.g. *Location* / *Geo-Location* / *Place* / *Geography*);
- Several types (e.g. *Product*) are equivalently used by few NER systems, while in others they are either not present, attributable to other corresponding types (e.g. *Thing* / *Artifact* / *Artwork* / etc.) or partitioned in multiple types (e.g. *Movie*, *TelevisionShow*, *Vehicle*, etc.);
- Since each generic ontology is composed of different types, the *Other* (or *Unkown*) type can assume disparate meanings depending on the other types involved.
- It is also notable the use of types that are particularly related to a language register, such as *Hashtag*, *Twitter user* or *IP address*.

Indeed, comparisons and integrations of NER systems become complex even for human experts without considering each single entity mention case.

As stated in [114], NER models can be roughly distinguished on two main approaches: (i) supervised Machine Learning models trained on large manually-labeled corpora [111]; and (ii) knowledge-based methods [110, 112]. However, both approaches suffer from two main limitations:

1. The amount of available data for accurately training a NER system according to a given target ontology can be limited, due to the costly and time-consuming labeling activity.

Many NER systems based on *Machine Learning* (e.g. Conditional Random Fields [115] and Labeled LDA [116]) assume that the training and test data must be represented in the same feature space and have the same underlying distribution. However, when a NER system needs to be adapted to a different domain ontology, this assumption may not hold. Since the target data are characterized by a different representation space and follow a different data distribution with respect to the source data, the lack of a training set can be a problem. This scenario is perfectly depicted in the #Microposts2015 Challenge [117], where the number of training instances ($\sim 3,500$) are not sufficient for inducing a NER system that is able to generalize unseen data.

2. Beyond the Machine Learning approaches, the task of Named Entity Recognition and Classification can be also performed by exploiting *Knowledge Bases* (KBs). In [118], Knowledge Bases, also referred as Knowledge Graphs, are defined as large networks of entities, their semantic types, properties, and relationships between entities. In this case, the list of all possible entity mention are extracted from a KB and the identification of these entities in a given text is performed by looking for matching parts. However, different NER systems can refer to different KBs (e.g. Wikipedia, DBpedia, Freebase, etc.) that are not guaranteed to be available and accessible at any time. For example, the system proposed by Ritter et al. [109] trains a Labeled LDA model using Freebase as underlying ontology, which was shut down in 2014. Moreover, the dimension of KBs increases rapidly due to new upcoming entities that users generate every day. In this case, it could be also expensive to frequently update the exploited knowledge.

In order to solve the problem of comparisons and integrations of NER systems, together with the challenges of the Named Entity Recognition and Classification task especially for user-generated content, the next sections will present a novel method discussed in [119, 120], called **Learning To Adapt** (L2A). The proposed approach is intended to adapt the entity classification of generic types on user-generated content to a given generic target ontology¹. By performing this operation, not only the model will adapt to a new ontology without the need of the slow task of retrieving and injecting external knowledge, but it will be also able to correctly map entity mentions that have been misclassified from the source NER system, for example because of the ambiguity. The concrete improvement of L2A consists of the use of **Word Embeddings** representation as input to the adaptation model. Involving Word Embeddings results in better classification performance, as the word vector representation includes several properties of the word (semantic, syntactic, etc.) that can help in dealing with non-local dependencies and ambiguity.

¹In the following, generic ontologies are simply referred as ontologies.

5.1.1 Word Embeddings Representation for Learning to Adapt Entity Classification

As Natural Language Processing in general (Sec. 2.1), the main challenge of Named Entity Recognition and Classification models applied to *user-generated content* regards the noisy and dynamic nature of the Web 2.0 language. While excellent accuracy performance have been obtained on well-formed text, the state of the art studies on ill-formed text, such as microblog posts, do still need strong improvements [121].

In addition to the intrinsic language difficulty of entity recognition and classification, it is often the case that different NER systems use different ontologies for entity classification, although it can commonly happen that a system using one source ontology may need to be adapted to the use of a different target ontology according to application requirements.

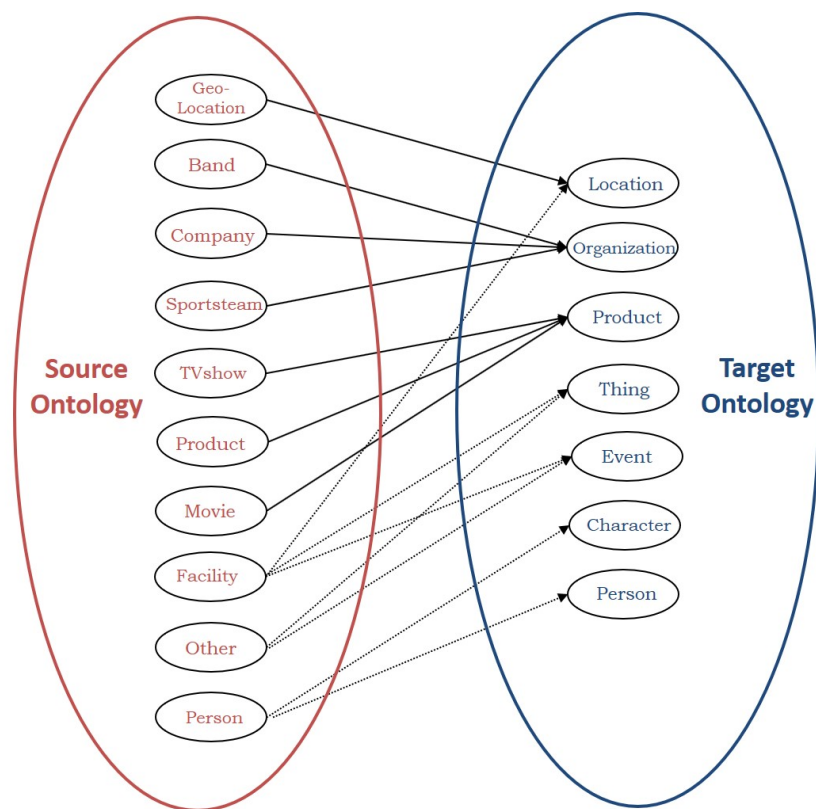


FIGURE 5.3: Manual mappings between two generic ontologies.

A first investigation aimed at dealing with this issue has been presented in [110], where a manual mapping between generic ontologies has been defined. As represented in Figure 5.3, a manual mapping is a deterministic mapping from the source ontology types to the target ontology types manually defined by a human expert. Although this study represents a first step towards the definition of cross-ontology NER systems, there is still the need of automatic mapping methods that can be used for dealing with any ontologies without the need of human intervention. In order to enable automatic mappings, some open problems need to be accurately addressed:

1. **Mention Misclassification:** Entity mentions are often misclassified by supervised NER systems mainly because of two different reasons: (i) the training set is composed of very few instances, and (ii) the training set is characterized by an imbalanced distribution over the ontology types. Consider, for instance, the entity mention *Black Sea* (a water body, i.e., a location) which has been erroneously recognized as *Band* in the source ontology, and should be mapped to *Location* in the target ontology. Given a deterministic manual mapping - as the one reported in Figure 5.3 - the mention will instead be mapped to the wrong entity type (*Organization*).
2. **Type Uncertainty:** There are also cases in which the type of an identified entity mention may be particularly uncertain, since a mention may have subtly different meanings in the real world. In this case, the decision of determining its type becomes difficult.

While well-structured text provides meaningful insights into the contextual usage of a mention, there can still be cases where it is difficult for an entity recognition system to correctly classify a mention. Consider, for instance, the entity mention *Starbucks* in a well-structured document snippet:

“It now takes 300 stars to reach gold level status at *Starbucks*, which means spending approximately ...”

A NER system could be uncertain about the type to be associated with *Starbucks*, because it could be equally probable to classify the entity mention in the source ontology as a *Geo-Location* (a particular Starbucks shop), or a *Company* (the Starbucks company). This uncertainty needs to be solved for determining the correct type in the target ontology.

3. **Fork Mappings:** There are cases, which have been named as fork mappings, where mentions classified as member of one type in the source ontology could be cast into one among two (or more) different types in the target ontology. Currently, this investigation identifies three cases of fork mappings (as seen in Figure 5.3):
 - ◇ the mapping of the type *Person* in the source ontology to the types *Person* or *Character* in the target ontology,
 - ◇ the mapping of the type *Other* in the source ontology to the types *Thing* or *Event* in the target ontology,
 - ◇ the mapping of the type *Facility* in the source ontology to the types *Thing*, *Event* or *Location* in the target ontology,

In order to tackle the above-mentioned issues arising when it is intended to adapt a NER system trained on a source generic ontology to a given target one, this thesis presents a novel approach called **Learning To Adapt (L2A)**, where named entities are represented by exploiting **Word**

Embeddings for obtaining a richer semantic input space. The use of a distributional representation is motivated by the intuition that, among all the implicit aspects of a word, Word Embeddings will also be able to reflect the *ontology type*, as entities of the same type are expected to appear in similar context. Although the proposed approach for mapping entity types from a source to a target ontology has been tested on microblog posts, it can be applied to a variety of different textual formats.

The subsequent sections are organized as follows. Section 5.1.1.1 introduces the proposed solution, named Learning To Adapt with Word Embeddings representation. An overview of the evaluation datasets and the results of the conducted analysis are presented in Sections 5.1.1.2 and 5.1.1.3. Results have revealed three main findings: (1) L2A is able not only to adapt an existing NER to a new target ontology, but it also enables the correction of misclassified entities as well as entities involved in type uncertainty and fork mappings; (2) Word Embeddings provide a remarkable improvement on the adaptation performance, (3) L2A outperforms not only the state of the art manual mapping approach but also two additional baselines based on probabilistic sampling. Finally, Section 5.1.1.4 presents some related work.

5.1.1.1 Adaptation Model: Learning to Adapt with Word Embeddings

The problem of adapting the types of entity mentions from a source ontology to the types in a target ontology can be viewed as a Machine Learning problem. In particular, given a set of entity mentions identified by a NER model originally trained in a source ontology, the main goal is to learn how to map the source type probability distribution to the target one.

More precisely, let R_S be a NER model trained on a set $\Omega_S = \{s_1, s_2, \dots, s_{n_s}\}$ of entity mentions annotated according to a source ontology O_S . Let $\Omega_T = \{t_1, t_2, \dots, t_{n_t}\}$ be a set of entity mentions that needs to be automatically labeled according to a target ontology O_T , by using a NER model R_S previously trained on Ω_S . Then, the labeling of Ω_T using R_S can be viewed as a transfer learning problem [33]. In particular, the main goal is to learn a target predictive function $f(\cdot)$ in Ω_T using some knowledge both in the source ontology S and the target ontology T .

More formally, let $P(\Omega_T, O_S)$ be the distribution in the source ontology used to label an entity mention $t_i \in \Omega_T$ with the most probable type $y_S^* \in O_S$ according to R_S and let $E : \Omega_T \rightarrow \mathbb{R}^m$ be the function that maps the entity mention $t_i \in \Omega_T$ to a m -dimensional embedding representation. The input space of the investigated adaptation problem is defined as $X_{P \sim E} = P(\Omega_T, O_S) \frown E(\Omega_T)$, where \frown is the concatenation symbol. Thus, the input space is the concatenation of the probability distribution in the source ontology and the embedded representation related to the entity mention $t_i \in \Omega_T$. Let $y_T \in O_T$ be the type in the target ontology that adaptation model should discover. Now, the adaptation of a source type y_S (of a given entity mention) to a target type y_T can be modeled as a learning problem aimed at seeking a function $\phi : X_{P \sim E} \rightarrow y_T$ over the

hypothesis space Φ . In our case, it is convenient to represent ϕ as a function $f : X_{P \sim E} \times y_T \rightarrow \mathbb{R}$ such that:

$$g(P(t_i, y_S) \sim E(t_i)) = \arg \max_{y_T \in O_T} f\left((P(t_i, y_S) \sim E(t_i)), y_T\right) \quad (5.1)$$

In order to solve this learning problem, it is necessary to create an input space representing each entity mention t_i that can be used for learning how to map the predicted source type $y_S \in O_S$ to the target type $y_T \in O_T$. As formally introduced, the input space $X_{P \sim E}$ for each entity mention t_i corresponds to the union of the explicit distribution given by R_S , $P(t_i, y_S)$, and its embedded representation $E(t_i)$. The output space denotes the most probable type $y_T \in O_T$. Using a model that is able to estimate a posterior distribution of y_T , we can therefore estimate the type distribution $P(\Omega_T, O_T)$ in the target ontology. A graphical example of the proposed approach is reported in Figure 5.4.

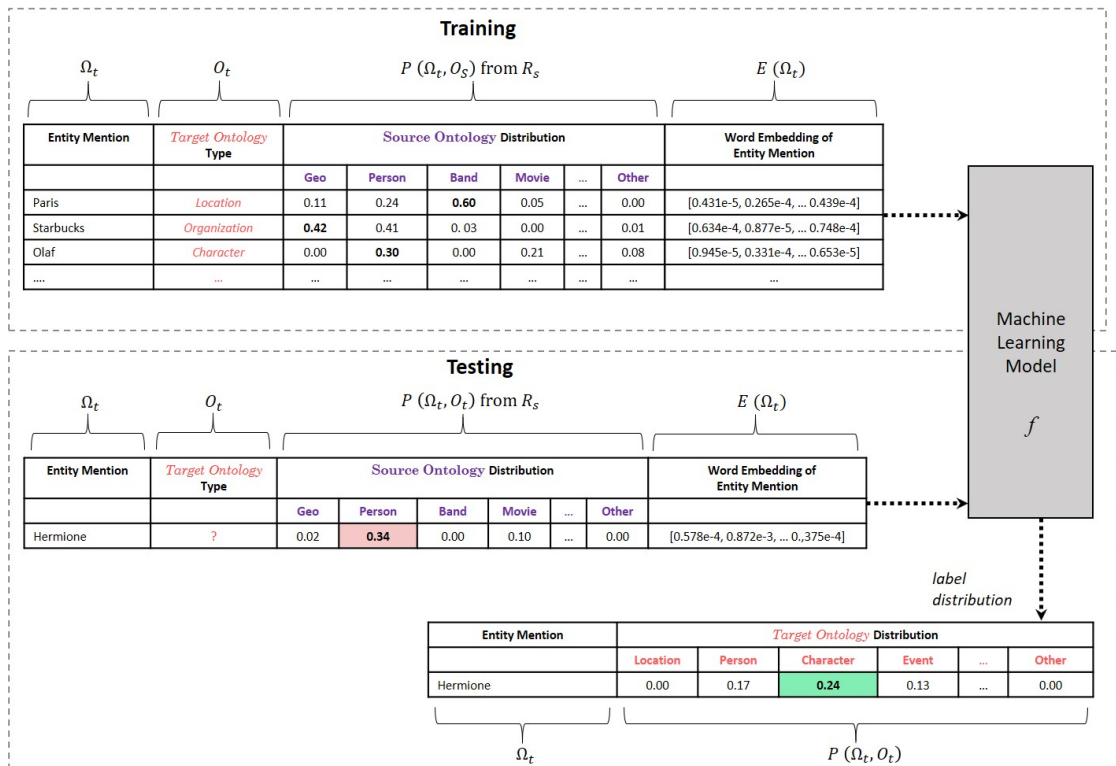


FIGURE 5.4: Graphical example of L2A.

The aim of L2A is then to learn the function f that is able to correctly label an entity mention $t_i \in \Omega_T$ according to the prediction $P(t_i, y_S)$ given by a NER model previously trained on Ω_S and its embedding representation $E(t_i)$. To accomplish this task, any Machine Learning algorithm can be adopted. This thesis investigates the most popular ones, i.e. Naïve Bayes (NB), Support Vector Machines (SVM), Decision Trees (DT), K-Nearest Neighbor (KNN), Bayesian Networks (BN), Multi-Layer Perceptron (MLP) and Multinomial Logistic Regression (MLR).

5.1.1.2 Experimental Settings

This section presents the investigated datasets and the comparative baselines used to evaluate the proposed approach. Then, different evaluation performance measures have been explored to validate the ability of L2A to perform the adaptation problem by assessing several configurations.

Word Embeddings The additional input information provided by the embedding representation of the entity mention can strongly influence the performance of the adaptation model. The expected improvement is strictly related to the increased semantic meaning of the input representation. As presented in Chapter 4, the core idea behind Word Embeddings is that words that appear in similar contexts should have similar vector representations. The proposed approach is motivated by the intuition that, among all the implicit aspects of the word, Word Embeddings will reflect also the *ontology type* property. It is, in fact, intuitive to think that words of type *Person*, for instance, are used in the same context, and the same for all the involved ontology types.

Since the amount of available data is not enough for obtaining a sufficiently trained Word Embeddings model, this investigation considers two different pretrained models for mapping entities to real-valued vectors:

- **Wiki2Vec** model: these Word Embeddings have been obtained by training the Skip-gram model (Sec. 4.1.3) over a Wikipedia dump, a more detailed description of this process is given in Section 5.2.1.2.
- **GoogleNews** model: Google News is the first corpus subjected to the learning of Word2vec models [2]. This corpus is composed of 100 billion words. The model is available online² and it contains 300-dimensional vectors for 3 million words and phrases trained by CBOW model with negative sampling (as presented in Section 4.1.3).

Another important issue to consider is the one related to *multi-word entities*, i.e. entity mentions composed by two or more words, such as *Emma Watson*, *Paris Hilton* or *University of Milano-Bicocca*. Following the definition given in Section 4.1, a Word Embeddings model is defined as a mapping $C : V \rightarrow \mathbb{R}^m$ that associates to each word in the vocabulary $w^* \in V$ a real vector $C(w^*) \in \mathbb{R}^m$. Indeed, given an entity mention t_i composed by several words $t_i = \{w_1^i, \dots, w_n^i\}$, the function E can be written as the aggregation of the mapping C over all the words w_j^i :

$$E(t_i) = \bullet(w_1^i, \dots, w_n^i), \quad (5.2)$$

²<https://code.google.com/archive/p/word2vec/>

where \bullet is the aggregation function. This is a common approach addressed in several state of the art studies [122–124]. Beyond the commonly investigated aggregation functions *max*, *min* and *mean*, the *first* aggregation that corresponds to take the Word Embeddings of the first word only has been also evaluated (eventually considered as the one carrying the type information, e.g. *University of Milano-Bicocca*). Note that, when an entity mention is formed by a single word, the function E will behave exactly as the original mapping C .

Dataset To perform an experimental analysis of the proposed approach, two benchmark datasets of microblog posts have been considered as **ground truth (GT)**, these data have been made available for the Named Entity Recognition and Linking Challenges for #Microposts2015 [117] and #Microposts2016 [125] Workshops. In particular, the datasets used as reference for the evaluation is the training set provided by the challenges (additional results on Test and Dev set are provided in Appendix B). These ground truths are composed of 3,498 and 6,025 posts, respectively, with a total of 4,016 and 8,664 entity mentions in each of them. Beyond Word Embeddings, the input space has been derived using the state of the art Ritter system **T-NER** [109], specifically conceived for dealing with user-generated content. In particular, T-NER makes use of Labeled LDA [116] to derive $P(\Omega_T, O_S)$, one of the components of the L2A input space. T-NER is trained using an underlying source ontology O_S (known as Ritter Ontology) to finally derive a NER model R_S .

Ritter Ontology (O_S): *Band, Company, Facility, Geo-Location, Movie, Other, Person, Product, Sportsteam, TVshow*.

Once the entity types are recognized by R_S and classified according to O_S , they need to be mapped to the entity types available in the target ontology (known as Microposts Ontology) O_T .

Microposts Ontology (O_T): *Character, Event, Location, Person, Product, Organization, Thing*.

T-NER identifies a total of 2,535 and 4,394 entity mentions from the #Microposts2015 and #Microposts2016 datasets respectively. In order to create the input space for the proposed L2A model, it is necessary to create a training set, where, for each entity mention identified by T-NER, the probability distribution $P(\Omega_T, O_S)$, the embedded representation $E(\Omega_T)$, the source type $y_S \in O_S$ and the target type $y_T \in O_T$ should be derived. While the probability distribution and the source type are explicitly provided by the T-NER system, the target type needs to be specified. However, when selecting a target type, it should be taken into account that an entity mention recognized by T-NER could be wrongly segmented, and some tokens of multi-word entity can be classified as non-entity or a single-word entity can be coupled with some adjoining words and therefore wrongly segmented as a multi-word entity. Two examples are reported below.

“The [**Empire State**]_{Geo-Location} [**Building**]_{Other} is amazing!”.

“[**Paris Hilton** will]_{Person} be in Venice next week!”.

To finally induce the L2A model, a *training set*, each for #Microposts2015 and #Microposts2016 has been **automatically** constructed by exploiting a string similarity measure (i.e., edit distance) which captures only the perfect matches between the mentions identified by T-NER and the mentions in the Microposts ground truth datasets. This means that, for each tweet in the Microposts datasets (gold standards), it has been associated each entity mention $t_i(T-NER)$ given by T-NER with the most similar entity mention $t_j(GT)$ in the ground truth. A couple $\langle t_i, y_T \rangle$ is added to the training set if and only if there is a perfect match between the entity mentions $t_i(T-NER)$ and $t_j(GT)$, where y_T is the correct type for that mention in the target ontology (made available from the ground truth). This automatic procedure for generating the training sets used as input by the L2A model is applicable and replicable on any labeled benchmark.

As a result, the training sets for #Microposts2015 and #Microposts2016 are composed of 1,660 and 3,003 training instances, respectively. Tables 5.2 and 5.3 show the distribution of ontology types in the obtained training set, respectively referring to the Ritter Ontology (O_S) and Microposts Ontology (O_T). It is worthy to notice that the distribution is strongly imbalanced, as real-world user-generated content are. While *Person* and *Location* (or *Geo-Location*) are clearly the dominant ontology types, other classes are barely present (e.g. *Character*, *Event*, *TVshow*, *Movie*).

TABLE 5.2: Type Distribution (%) according to Ritter Ontology (O_S).

	#Microposts2015	#Microposts2016
Band	3.19	3.26
Company	8.86	6.99
Facility	1.99	2.53
Geo-Loc	28.86	33.17
Movie	1.87	1.86
Other	11.93	12.32
Person	35.24	33.97
Product	3.67	2.86
Sportsteam	3.07	2.16
TVshow	1.33	0.87

TABLE 5.3: Type Distribution (%) according to Microposts Ontology (O_T).

	#Microposts2015	#Microposts2016
Character	1.27	1.00
Event	1.75	3.83
Location	30.60	37.63
Organization	24.82	19.85
Person	31.69	29.57
Product	7.53	5.83
Thing	2.35	2.30

Baselines In order to compare the proposed approach with a reference, three **Baseline** models have been defined:

- **Baseline-Deterministic (BL-D)**: it considers the manual mapping between O_S and O_T shown in Figure 5.3;
- **Baseline-Probabilistic (BL-P1)**: it extends the previous baseline in order to deal with fork mappings in a non-deterministic way. In particular, for those mentions in O_S that can be classified in more than one type in O_T , the target type has been sampled according to the a priori distribution of mappings in the training set (e.g. 30% of *Person* entity mentions in O_S are classified as *Character* and 70% as *Person* in O_T).
- **Baseline-Probabilistic (BL-P2)**: A major downside of using the deterministic manual mapping (BL-D) is that since it directly depends on the output of the T-NER system, it will never be able to correct the target type of the mentions which have been incorrectly classified by T-NER. For this reason, an additional probabilistic baseline (BL-P2) has been introduced. For each mention, given the associated source type $y_S \in O_S$, the target type y_T has been sampled from the distribution $P(O_T|y_S \in O_S)$ estimated on the training set.

Performance Measures In order to evaluate the different model configurations and to compare them with the afore-mentioned baselines, several state of the art performance measures have been considered.

For evaluating the classification, the terms *true positives* (TP), *true negatives* (TN), *false positives* (FP), and *false negatives* (FN) compare the instances classified by the model with the ground truth. The terms positive and negative refer to the classifier's prediction, while true and false refer to the comparison with the ground truth.

- **Accuracy**: it represents the number of correctly labeled named entities over the total number of instances. This value is defined between 0 and 1.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (5.3)$$

- **Precision**: it represents the number of positive predictions divided by the total number of positively class values predicted. This value is defined between 0 and 1.

$$Precision = \frac{TP}{TP + FP} \quad (5.4)$$

- Recall: it represents the number of positive predictions divided by the number of positive class values in the dataset. This value is defined between 0 and 1.

$$Recall = \frac{TP}{TP + FN} \quad (5.5)$$

- F-measure: it is the harmonic mean of the Precision and the Recall. This thesis considers the F1-metric which weights Recall and Precision equally. It is defined between 0 and 1.

$$F\text{-measure} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (5.6)$$

- Accuracy Contribution: represents the number of correctly labeled named entities, classified as a specific class y_T over the total number of instances of the ontology type y_T .

$$Accuracy(y_T) = \frac{\# \text{ instances correctly classified as } y_T}{\# \text{ instances correctly of type } y_T} \quad (5.7)$$

In the following approach, *Precision*, *Recall* and *F-measure* can be strongly influenced by the imbalanced distribution of the data over the ontology types (Tab. 5.2 and 5.3). For this reason, the overall performance measures of a multi-class classification problem can be computed by two different types of average, *macro-average* and *micro-average* [126]. *Macro-averaged* measures give equal weight to each class, regardless of its frequency. Consequently, it is more influenced by the classifier's performance on rare categories. *Micro-averaged* measures weight each class with respect to its number of instances. It tends to be dominated by the classifier's performance on common categories.

Using the same formalism of [127], M is the number of classes; TP_i (True Positives) is the number of documents assigned correctly to class i ; FP_i (False Positives) is the number of documents that do not belong to class i but are assigned to class i incorrectly by the classifier; and FN_i (False Negatives) is the number of documents that are not assigned to class i by the classifier but which actually belong to class i . Following, the *micro-* and *macro-averaged* version of *Precision*, *Recall* and *F-measure* are reported:

- Precision:

$$Precision_{micro} = \frac{\sum_{i=1}^M TP_i}{\sum_{i=1}^M (TP_i + FP_i)} \quad (5.8) \quad Precision_{macro} = \frac{1}{M} \sum_{i=1}^M \frac{TP_i}{TP_i + FP_i} \quad (5.9)$$

- Recall:

$$Recall_{micro} = \frac{\sum_{i=1}^M TP_i}{\sum_{i=1}^M (TP_i + FN_i)} \quad (5.10) \quad Recall_{macro} = \frac{1}{M} \sum_{i=1}^M \frac{TP_i}{TP_i + FN_i} \quad (5.11)$$

- F-measure:

$$F\text{-measure}_{micro} = 2 \cdot \frac{Precision_{micro} \cdot Recall_{micro}}{Precision_{micro} + Recall_{micro}} \quad (5.12)$$

$$F\text{-measure}_{macro} = 2 \cdot \frac{Precision_{macro} \cdot Recall_{macro}}{Precision_{macro} + Recall_{macro}} \quad (5.13)$$

In the following sections, given the highly imbalanced distribution (Tables 5.2 and 5.3), **Precision**, **Recall** and **F-measure** are shown in their *micro-averaged* version. It is important to note that the state of the art performance measure usually computed for evaluating NER systems, called Strong Typed Mention Match (**STMM**), is defined as the *micro-averaged* F-measure. Moreover, the *macro-averaged* F-measure has been also reported for the sake of completeness.

Experimental Settings Concerning the experimental evaluation, a *10-folds cross validation* has been performed. To compare L2A with the baseline models both on #Microposts2015 and #Microposts2016, $Precision_{micro}$, $Recall_{micro}$, $F\text{-measure}_{micro}$ and $F\text{-measure}_{macro}$ have been used for comparing the types predicted by L2A with the real types available in the ground truth. In order to evaluate the contribution of the different components of the input space, several configurations have been explored: the probability distribution in the source ontology $X_P = P(\Omega_T, O_S)$, the embedded representation $X_E = E(\Omega_T)$ and the joint input space $X_{P \sim E} = P(\Omega_T, O_S) \frown E(\Omega_T)$. Concerning the models used to train L2A, i.e. Naïve Bayes (NB), Support Vector Machines (SVM), Decision Trees (DT), K-Nearest Neighbor (KNN), Bayesian Networks (BN), Multi-Layer Perceptron (MLP) and Multinomial Logistic Regression (MLR), no parameter optimization has been performed³.

5.1.1.3 Experimental Results

In this section, several computational experiments are presented in order to show the relevance of the proposed approach for the afore-mentioned datasets. As first concern, the best configurations of L2A that consider the Word Embeddings feature space (X_E and $X_{P \sim E}$) have been studied. Then, the selected results have been evaluated with respect to the baselines and the Machine Learning models trained over the source ontology probability distribution (X_P).

In order to investigate the advantages of considering the Word Embeddings feature space, Table 5.4 reports the results in terms of Accuracy obtained on both datasets for all the chosen Machine Learning models, aggregation functions and pretrained Word Embeddings models, highlighting the best results for each dataset.

³The experiments have been conducted using default parameters of models implemented in WEKA: www.cs.waikato.ac.nz/ml/weka/

TABLE 5.4: Accuracy performance of L2A model considering Word Embeddings feature space. For each dataset, the best results are reported in bold.

		#Microposts2015				#Microposts2016			
		X_E		$X_{P \sim E}$		X_E		$X_{P \sim E}$	
		Wiki2Vec	GoogleNews	Wiki2Vec	GoogleNews	Wiki2Vec	GoogleNews	Wiki2Vec	GoogleNews
BN	mean	0.61	0.60	0.76	0.76	0.70	0.73	0.71	0.74
	max	0.58	0.58	0.73	0.74	0.71	0.74	0.72	0.75
	min	0.58	0.59	0.73	0.75	0.71	0.75	0.72	0.76
	first	0.60	0.59	0.74	0.73	0.69	0.72	0.70	0.73
DT	mean	0.51	0.54	0.75	0.74	0.70	0.71	0.77	0.78
	max	0.54	0.55	0.72	0.73	0.71	0.71	0.78	0.78
	min	0.52	0.55	0.75	0.74	0.68	0.69	0.78	0.77
	first	0.51	0.55	0.72	0.73	0.68	0.71	0.77	0.78
KNN	mean	0.58	0.60	0.75	0.79	0.80	0.82	0.81	0.83
	max	0.59	0.60	0.75	0.78	0.79	0.81	0.80	0.82
	min	0.59	0.61	0.75	0.79	0.78	0.81	0.80	0.83
	first	0.57	0.59	0.72	0.76	0.73	0.77	0.79	0.81
MLR	mean	0.58	0.54	0.75	0.75	0.83	0.78	0.77	0.78
	max	0.59	0.54	0.74	0.73	0.77	0.78	0.78	0.77
	min	0.59	0.53	0.76	0.74	0.81	0.78	0.77	0.77
	first	0.57	0.55	0.72	0.71	0.80	0.76	0.78	0.76
MLP	mean	0.63	0.64	0.82	0.84	0.85	0.85	0.86	0.85
	max	0.63	0.64	0.82	0.83	0.85	0.85	0.85	0.86
	min	0.63	0.64	0.82	0.83	0.85	0.85	0.86	0.85
	first	0.61	0.62	0.81	0.80	0.81	0.81	0.84	0.83
NB	mean	0.61	0.60	0.76	0.76	0.72	0.72	0.74	0.75
	max	0.58	0.58	0.73	0.74	0.73	0.74	0.75	0.77
	min	0.58	0.60	0.73	0.75	0.73	0.74	0.74	0.77
	first	0.60	0.59	0.74	0.74	0.72	0.73	0.74	0.75
SVM	mean	0.63	0.65	0.84	0.85	0.84	0.84	0.86	0.86
	max	0.63	0.65	0.84	0.84	0.85	0.85	0.86	0.86
	min	0.62	0.63	0.84	0.84	0.85	0.85	0.86	0.86
	first	0.61	0.64	0.82	0.81	0.81	0.81	0.83	0.83

From Table 5.4, it is possible to draw a variety of conclusions as follows:

- Considering the joint input space $X_{P \sim E}$ leads to better results as opposed to considering only the embedded representation (X_E) of the entity words. Although the probability distribution vector covers only the 2% of the complete feature space, when combined with the embedded representation, it brings a great improvement in the classification. This means that considering the Word Embeddings representation only does not provide sufficient information for correctly mapped entity mentions.
- SVM and MLP proved to be the best models for dealing with the real-valued vectors of Word Embeddings, while Decision Tree is observed to exhibit the worst performance. This is likely due to the feature space nature, since the former models are best known for treating large real-valued vectors.
- While it is difficult to decide the best performing aggregation method among *mean*, *max* and *min*, it is clear that taking the representation of the *first* word of an entity mention (when dealing with mentions composed of multiple words) leads to a limited view of the underlying meaning of the entity mention and consequently to lower performance in terms of Accuracy.

- Finally, *GoogleNews* pretrained model performs better than the Wiki2Vec one. This can be due to the different nature and size of training data for these models.

TABLE 5.5: Class-Wise Accuracy Contribution (%) on #Micropost2015 of L2A model considering Word Embeddings feature space (X_E and $X_{P \sim E}$).

	MLP						SVM					
	mean		max		min		mean		max		min	
	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$
Character	0.48	0.54	0.48	0.48	0.42	0.54	0.54	0.48	0.60	0.48	0.48	0.48
Event	0.60	1.14	0.48	0.96	0.42	0.72	0.66	1.14	0.36	0.90	0.36	0.78
Location	21.81	27.23	21.93	27.11	22.17	27.23	22.11	27.59	22.41	27.47	22.05	27.59
Organization	13.86	18.73	14.04	18.92	13.98	19.22	13.61	19.58	14.28	19.34	12.83	19.22
Person	22.83	29.94	22.35	29.58	22.35	29.22	23.86	29.88	22.83	29.4	23.49	29.58
Product	3.19	4.76	3.25	4.52	3.25	4.28	3.43	4.82	3.19	4.82	3.19	4.58
Thing	0.90	1.63	1.08	1.75	1.08	1.63	1.02	1.63	0.96	1.63	1.02	1.57
Overall	63.67	83.98	63.61	83.31	63.67	82.83	65.24	85.12	64.64	84.04	63.43	83.8

TABLE 5.6: Class-Wise Accuracy Contribution (%) on #Micropost2016 of L2A model considering Word Embeddings feature space (X_E and $X_{P \sim E}$).

	MLP						SVM					
	mean		max		min		mean		max		min	
	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$
Character	0.37	0.37	0.30	0.33	0.30	0.37	0.33	0.30	0.27	0.33	0.33	0.30
Event	3.16	3.06	3.10	3.03	3.06	3.03	3.20	3.26	3.13	3.13	3.10	3.16
Location	34.13	34.37	34.53	34.67	34.43	34.63	34.47	34.87	35.06	35.06	35.03	35.00
Organization	14.85	14.65	14.99	15.02	14.62	14.59	13.79	14.82	13.89	14.72	14.19	14.69
Person	27.57	27.74	27.47	27.51	27.54	27.67	27.41	27.91	27.47	27.64	27.27	27.84
Product	2.93	3.20	3.06	3.13	2.86	3.10	2.83	2.96	3.20	3.33	2.80	2.90
Thing	1.70	1.70	1.83	1.83	1.86	1.70	1.73	1.83	1.80	1.76	1.80	1.80
Overall	84.72	85.08	85.28	85.51	84.68	85.08	83.75	85.95	84.82	85.98	84.52	85.68

In order to report a more compact representation of all the experimental results, in Tables 5.5 and 5.6 only the best performing configurations for L2A are given. In particular, SVM and MLP have been selected as Machine Learning models, *mean*, *max* and *min* as aggregation methods and *GoogleNews* pretrained model as Word Embeddings representation. For each type, the highest result has been highlighted in bold.

As it is possible to perceive from both tables, Support Vector Machines drive to better results, by reaching the highest values of the overall Accuracy. While for #Microposts2015 the prevalence of the *mean* aggregation function is evident, the results on #Microposts2016 are less clear. However, by jointly considering the two datasets, the choice of using SVM as Machine Learning model and *mean* as aggregation function results as the best performing adaptation model in terms of Accuracy.

Since Accuracy is a too much punctual measure for evaluating the performance of a Machine Learning classifier, it is important to consider also other measures for a wide and complete overview. As presented in the previous section, $Precision_{micro}$, $Recall_{micro}$, $F-measure_{micro}$ and $F-measure_{macro}$ (Tables 5.7 and 5.8) give a more extensive idea considering also the issues of multi-class classification and imbalanced class distribution. By looking at these measures, the

TABLE 5.7: Precision, Recall, F-Measure and STMM on #Micropost2015 of L2A model considering Word Embeddings feature space (X_E and $X_{P \sim E}$).

	MLP						SVM					
	mean		max		min		mean		max		min	
	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$
Precision _{micro}	0.63	0.84	0.63	0.83	0.64	0.83	0.65	0.85	0.64	0.84	0.63	0.84
Recall _{micro}	0.64	0.84	0.64	0.83	0.64	0.83	0.65	0.85	0.65	0.84	0.63	0.84
F-measure _{micro}	0.63	0.84	0.63	0.83	0.63	0.83	0.65	0.85	0.64	0.84	0.63	0.84
F-measure _{macro}	0.54	0.74	0.54	0.73	0.54	0.70	0.57	0.75	0.54	0.73	0.52	0.71

TABLE 5.8: Precision, Recall, F-Measure and STMM on #Micropost2016 of L2A model considering Word Embeddings feature space (X_E and $X_{P \sim E}$).

	MLP						SVM					
	mean		max		min		mean		max		min	
	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$
Precision _{micro}	0.84	0.85	0.85	0.85	0.84	0.85	0.83	0.85	0.84	0.86	0.84	0.85
Recall _{micro}	0.85	0.85	0.85	0.86	0.85	0.85	0.84	0.86	0.85	0.86	0.85	0.86
F-measure _{micro}	0.84	0.85	0.85	0.85	0.84	0.85	0.83	0.86	0.84	0.86	0.84	0.85
F-measure _{macro}	0.75	0.74	0.75	0.74	0.73	0.74	0.73	0.75	0.74	0.75	0.74	0.74

TABLE 5.9: Class-Wise Accuracy contribution (%) on #Micropost2015 of L2A model and baselines.

Entity Type	Baselines			L2A								
	BL-D	BL-P1	BL-P2	BN (X_P)	NB (X_P)	MLR (X_P)	MLP (X_P)	SVM (X_P)	DT (X_P)	KNN (X_P)	SVM _{mean} (X_E)	SVM _{mean} ($X_{P \sim E}$)
Character	0.00	0.96	0.48	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.54	0.48
Event	0.00	1.14	0.30	1.20	1.20	0.00	0.00	0.00	0.06	0.54	0.66	1.14
Location	24.76	26.69	20.72	26.45	26.45	26.20	27.71	26.27	27.59	27.41	22.11	27.59
Organization	11.63	11.63	11.02	15.24	15.30	17.71	17.59	17.65	17.47	17.11	13.61	19.58
Person	27.29	27.29	22.11	25.30	25.30	27.47	27.05	26.99	26.99	26.75	23.86	29.88
Product	2.35	2.35	1.57	1.02	1.02	2.05	2.71	1.99	2.11	2.35	3.43	4.82
Thing	0.66	0.66	1.57	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.63	1.02
Overall	66.69	70.72	57.77	69.22	69.28	73.43	75.06	72.89	74.22	74.16	65.24	85.12

TABLE 5.10: Class-Wise Accuracy contribution (%) on #Micropost2016 of L2A model and baselines.

Entity Type	Baselines			L2A								
	BL-D	BL-P1	BL-P2	BN (X_P)	NB (X_P)	MLR (X_P)	MLP (X_P)	SVM (X_P)	DT (X_P)	KNN (X_P)	SVM _{mean} (X_E)	SVM _{mean} ($X_{P \sim E}$)
Character	0.00	0.63	0.27	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.33	0.30
Event	0.00	2.70	1.76	2.26	0.20	0.67	1.53	0.63	2.26	2.30	3.20	3.26
Location	29.77	32.77	27.34	32.93	34.43	32.13	33.97	31.90	33.63	33.83	34.47	34.87
Organization	8.72	8.72	8.23	11.92	11.29	13.22	11.92	13.55	13.32	13.39	13.79	14.82
Person	25.31	25.31	20.38	23.34	25.94	25.34	26.04	24.98	25.37	24.94	27.41	27.91
Product	2.00	2.00	1.23	1.47	0.13	1.80	1.67	1.96	1.90	1.96	2.83	2.96
Thing	0.50	0.50	1.76	0.10	0.13	0.00	0.03	0.00	0.13	0.30	1.73	1.83
Overall	66.30	72.63	60.97	72.03	72.13	73.16	75.16	73.03	76.66	76.72	83.75	85.95

supremacy of SVM model is even more unequivocal. Moreover, using the *mean* aggregation function leads to the best results for both datasets, except for the *Precision*_{micro} of #Micropost2016 (by only 1 point percentage). These results have further motivated the choice of SVM-*mean* as the best performing model MLP considering Word Embeddings representation.

Following these considerations, the next evaluation step refers to the comparison of L2A, considering the best model configuration (i.e. SVM as Machine Learning model, *mean* as aggregator method and *GoogleNews* pretrained model as Word Embeddings model), with the baselines and all the considered Machine Learning models on the probability distribution in the source ontology (Tables 5.9 and 5.10).

TABLE 5.11: Precision, Recall, F-Measure and STMM on #Micropost2015 of L2A model and baselines.

Entity Type	Baselines			L2A								
	BL-D	BL-P1	BL-P2	BN (X_P)	NB (X_P)	MLR (X_P)	MLP (X_P)	SVM (X_P)	DT (X_P)	KNN (X_P)	SVM _{mean} (X_E)	SVM _{mean} (X_{P-E})
Precision _{micro}	0.73	0.77	0.78	0.75	0.75	0.69	0.70	0.69	0.71	0.71	0.65	0.85
Recall _{micro}	0.67	0.71	0.58	0.69	0.69	0.73	0.75	0.73	0.74	0.74	0.65	0.85
F-measure _{micro}	0.68	0.72	0.65	0.70	0.70	0.71	0.73	0.70	0.72	0.72	0.65	0.85
F-measure _{macro}	0.38	0.62	0.46	0.38	0.39	0.38	0.40	0.38	0.42	0.43	0.57	0.75

TABLE 5.12: Precision, Recall, F-Measure and STMM on #Micropost2016 of L2A model and baselines.

Entity Type	Baselines			L2A								
	BL-D	BL-P1	BL-P2	BN (X_P)	NB (X_P)	MLR (X_P)	MLP (X_P)	SVM (X_P)	DT (X_P)	KNN (X_P)	SVM _{mean} (X_E)	SVM _{mean} (X_{P-E})
Precision _{micro}	0.72	0.78	0.79	0.73	0.72	0.71	0.72	0.71	0.75	0.75	0.84	0.86
Recall _{micro}	0.66	0.73	0.61	0.72	0.72	0.73	0.75	0.73	0.77	0.77	0.85	0.86
F-measure _{micro}	0.68	0.74	0.68	0.72	0.72	0.71	0.73	0.71	0.75	0.75	0.84	0.86
F-measure _{macro}	0.37	0.61	0.49	0.44	0.44	0.42	0.45	0.42	0.50	0.51	0.74	0.75

It can be easily noticed that all the L2A configurations are able to achieve good adaptation performance in terms of global Accuracy. Lower Accuracy contributions by L2A can be observed for the entity types *Character* and *Thing*. This can be caused by the low number of training instances available for *Character* (1.27% in #Microposts2015 and 0.99% in #Microposts2016 dataset) and for *Thing* (2.35% in #Microposts2015 and 2.30% in #Microposts2016 dataset) that does not allow any algorithm to provide remarkable contributions to the total Accuracy.

Except for few cases in #Microposts2015, the consideration of a joint input space $X_{P \sim E}$ leads to the best Accuracy results, further demonstrating that taking into account the probability distribution and the Word Embeddings representation is the winning strategy for the investigated adaptation problem. This behavior can be motivated by the fact that, while the embedded representation is capable of extracting underlying factors of the named entities, these are not sufficient on their own, but they can bring a great advantage on enhancing the mere probability distribution vector, resulting in significantly better performance.

Analyzing the adaptation results of L2A from a **qualitative** point of view, it is interesting to highlight that the model is able to correctly re-classify the target types of entity mentions that have been misclassified, i.e. those mentions which would have been cast to incorrect target types due to wrong predictions given by the T-NER system. For example, “iPhone” was classified as a *Company* by T-NER (which would lead to the type *Organization* using manual mappings), while L2A correctly re-classifies this entity mention as a *Product*. As another example, “Ron Weasley” (a character in Harry Potter movies/books) was misclassified as *Band* by T-NER, while L2A correctly re-classifies it as a *Character*. In the latter case, L2A was able to assign the correct type among the two possible types defined according to fork mappings. Although there are very few instances in the training sets for the target types *Character* and *Event* and the performance of L2A is not very high in terms of Accuracy contribution, the proposed approach seems to be promising.

Tables 5.11 and 5.12 compare the performance of the proposed approach with respect to different input space configurations with the baselines in terms of $Precision_{micro}$, $Recall_{micro}$, $F-measure_{micro}$ and $F-measure_{macro}$.

As expected, the deterministic baseline (BL-D) achieves good performance in terms of $Precision_{micro}$, but low results of $Recall_{micro}$. In fact, BL-D is accurate when labeling mentions thanks to the deterministic mapping, at the expenses of $Recall_{micro}$. Also in this case, it can be easily noted that using SVM over the joint input space $X_{P \sim E}$ significantly outperforms the baselines and the other L2A configurations both for the #Microposts2015 and #Microposts2016 datasets. These experiments show that the proposed approach provides significant results with respect to all the considered performance measures and obtains a balanced contribution of $Precision_{micro}$ and $Recall_{micro}$. Moreover, L2A drastically improves the $Recall_{micro}$ measure. This is likely due to its ability to learn how to map the initial hypothesis given by T-NER to a new target type, adapting type mentions that were previously misclassified.

Capabilities Evaluation Beyond the classic performance evaluation measures, several *capabilities* have been measured with respect to the three issues stated in Section 5.1.1, i.e. mention misclassification, type uncertainty and fork mapping. These capability measures are described as follows:

1. **Mention Misclassifications Correctly Mapped (MMCM)**: this measure indicates the percentage of entity mentions that T-NER has wrongly classified and L2A is able to correctly map according to the target ontology. For the considered experimental set, in the training sets for #Micropost2015 and #Micropost2016, T-NER has wrongly classified 524 and 921 entity mentions respectively.
2. **Type Uncertainty Correctly Mapped (TUCM)**: this measure denotes the percentage of uncertain entity mentions that L2A correctly maps in the target ontology. To compute this measure, a mention t_i has been defined as an *uncertain mention* when it has a low gap between probability distribution over different types. More formally, t_i is considered as *uncertain* if:

$$P(t_i, y_{T_j}) - P(t_i, y_{T_k}) \leq \alpha_U \quad \forall j \neq k \quad (5.14)$$

where α_U is a parameter that has been experimentally determined as equal to 0.2. The number of mentions that have been recognized as *uncertain* in the training sets are 59 for #Micropost2015 and 109 for #Micropost2016.

3. **Fork Mappings Correctly Resolved (FMCR)**: this measure represents the percentage of mentions of a type defined as fork mappings (i.e. *Event*, *Location*, and *Character*) that have been correctly classified by L2A. According to the training sets, the number of mentions that fall under this category is 50 for #Micropost2015 and 145 for #Micropost2016.

TABLE 5.13: Capabilities performance measures on #Micropost2015 of L2A model and baselines.

Entity Type	Baselines		L2A				
	BL-P1	BL-P2	MLP (X_P)	SVM (X_P)	DT (X_P)	KNN (X_P)	SVM _{mean} (X_{P-E})
MMCM	2.67	19.00	35.88	25.57	36.07	34.92	64.50
TUCM	15.25	27.29	57.63	42.37	45.76	45.76	76.27
FMCR	25.96	22.10	26.19	25.00	28.57	40.48	63.10

TABLE 5.14: Capabilities performance measures on #Micropost2016 of L2A model and baselines.

Entity Type	Baselines		L2A				
	BL-P1	BL-P2	MLP (X_P)	SVM (X_P)	DT (X_P)	KNN (X_P)	SVM _{mean} (X_{P-E})
MMCM	4.67	18.26	32.03	24.00	40.50	39.52	64.71
TUCM	14.68	24.53	53.21	31.19	56.88	52.29	76.15
FMCR	34.00	32.17	48.67	28.52	57.41	58.94	78.33

The results are shown in Tables 5.13 and 5.14, where the most successful models have been considered. Among the baselines, the deterministic one (BL-D) has been discarded because its capability performance always amount to zero scores since it mimics a fixed manual mapping a priori defined. This means that if an entity mention is incorrectly classified in the source ontology, it will always be mapped to the corresponding (incorrect) class in the target ontology. For instance, if the mention “Paris” is incorrectly classified by T-NER as *Movie* (whereas its correct type is *Location*), BL-D will map Paris to *Product*, providing no improvement for the MMCM capability. The same reasoning is applicable also for TUCM and FMCR capabilities.

The first consideration that can be derived from the capabilities performance results is that, once again, SVM over the joint space is performing considerably better for all the considered measures. Secondly, it can also be observed that, in most cases, the results on #Microposts2015 set are worse than the ones on #Microposts2016. This is due to the fact that the number of entity mentions available for training L2A in the #Microposts2016 are about twice as much than in #Microposts2015 (as stated in Section 5.1.1.2). In other words, the higher the number of mentions that L2A can use to learn the correct mappings, the better the capabilities will be.

Furthermore, in order to better understand the poor results of FMCR, a detailed investigation has been conducted on the predictions of the Machine Learning models. For #Microposts2015, the number of mentions involved in a fork mapping is 50 (21 for the entity type *Character* and 29 for the entity type *Event*). Given the low frequency of these entity types in the dataset (note that the entity types *Location*, *Person* and *Organization* are composed of more than 400 instances each), it is very difficult for a Machine Learning algorithm to learn how to recognize their presence. On the other hand, in #Microposts2016, there are 145 entities involved in a fork mapping: 30 entities are *Character* and 115 *Event*. The results in terms of FMCR are promising but, following the previous intuition, the performance increase is mainly due to correctly classified instances for the entity type *Event*, while only a few instances of the type *Character* have been correctly identified.

From the presented results it is possible to conclude that the use of Word Embeddings can strongly improve the performance, both in terms of traditional measures and capabilities, on the task of adapting trained NER systems to new ontologies. The best adaptation abilities have been obtained by jointly consider as input space the Word Embeddings of the named entities and the probability distribution over the source ontology, and by using Support Vector Machines as Machine Learning classifier. As future works, it would be interesting to specifically train Word Embeddings models on corpora where the ontology type class is provided for the named entities, for example by exploiting the Wikipedia pages structure. Moreover, additional experiments over different and more specialized corpora (e.g. medical) are planned.

5.1.1.4 Related Works

To the best of our knowledge, this is the first work aimed at addressing the problem of automatically adapting entity mention types given by a NER system, trained on a source ontology, to comply to a new ontology. Manual mappings have been used to bridge the gap between NER systems using different ontologies [110]. When many-to-one mappings are used, which means that one source type is mapped to at most one target type, and when the source classification is reasonably accurate, manual mappings may achieve a good performance. However, in contexts such as microblogging platforms, where generic ontologies are used for the classification and pretrained NER systems are affected by the dynamics of new upcoming entities, these mappings have several limitations (as discussed in the previous sections).

The problem of adapting NER models has been recently investigated in the context of formal text [128, 129]. While the proposed model aims to adapt two different generic ontologies that can be used independently from the domain (given their generic nature), several state of the art approaches have been introduced to adapt NER models trained on specific ontologies to adapt to new domains. Arnold et al. [130] proposed an approach for domain adaptation able to learn a domain-independent NER base model, which can be adapted to specific domains. Furthermore, in [131] the authors presented a NER rule-based language which is further used for building domain-specific rule-based NER systems. A recent transfer-learning based method has been proposed in [114] for adapting a NER system from a source (medicine) domain to a target domain by using a linear chain CRF model which learns domain-specific patterns based on the correlations between the source and target entity types.

Another difference from these studies regards the considered environment, as they do not tackle the problem in a microblogging context where the language used by the users can vary significantly and new entities can emerge frequently.

Finally, marginally related to the presented investigation, it is possible to find Machine Learning methods applied to Ontology Matching [132–134] in literature. Textual annotation and

re-classification statistics have been also proposed in [135] in order to semantically interpret class-to-class ontology mappings. However, these approaches have been based on collecting feedback on class-to-class mappings in order to improve ontology alignments.

5.2 Named Entity Linking

A follow-up step to Named Entity Recognition and Classification is **Named-Entity Linking** (NEL), which is the task of determining the identity of entities mentioned in a textual document. Sometimes is also called Named-Entity Disambiguation or Named-Entity Normalization.

This task can be of great importance in many fields: it can be used by search engines for disambiguating multiple-meanings entities in indexed documents or for improving queries precision, as named entities are averagely present in 70% of cases [136]. NEL systems can also be used in combination with other Natural Language Processing systems, such as Sentiment Analysis, for the generation of additional knowledge to describe users preferences towards companies, politicians, and so on. Since NEL systems should cover the widest possible number and variations of named entities, several studies investigate user-generated content as source data, in particular messages originated from users in micro-blogging platforms such as Twitter. Due to its dynamic and informal nature, Twitter provides its users an easy way to express themselves and communicate thoughts and opinions in a highly colloquial way. This, in addition to the limitation of characters, induces the users to use abbreviations, slangs, and made-up words increasing the difficulty in recognizing and disambiguating the involved named entities.

The common NEL process typically requires annotating a potentially ambiguous entity mention with a link to global identifiers with unambiguous denotation, such as Uniform Resource Identifier (URI) in *Knowledge Bases*, describing the entity. Popular choices for the KB is Wikipedia, in which each page is considered as a named entity, or DBpedia, which is used as structured background knowledge in many NEL systems. An example of a sentence processed for the Named Entity Linking task is shown in Figure 5.5. In this example, the mention *@EmmaWatson* is correctly linked to the actress. A more difficult case regards the word *hermione* as it can assume very different meanings, e.g. the name of an autobiographical novel, a common given name or the character of the movie Harry Potter.

Beyond the issues previously introduced for NER, an additional problem is represented by the fact that the same entity could have multiple surface forms. An example could be the named entity *USA* also referred as *America*, *US*, *United State of America*, and *United States*. It is also important to consider that some words recognized by NER systems might not have a corresponding description in the KB, these words are referred as Out of Vocabulary (OOV) words.

These problems need to be addressed by any NEL as they are very common and could decrease the overall performance.

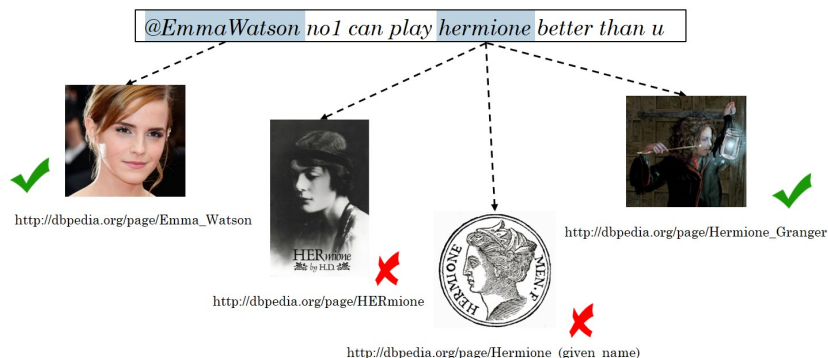


FIGURE 5.5: Example of a sentence processed by Named Entity Linking.

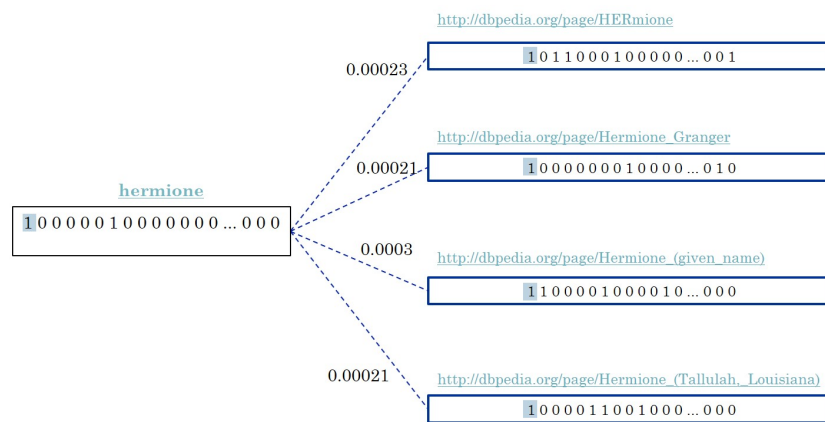
The next section introduces the proposed unsupervised Named Entity Linking system for user-generated content. The core advancement of the model concerns the use of Word Embeddings models to obtain a common representation space for both words and named entities. The majority of the state of the art solutions makes use of similarity measures based on the occurrence or frequencies of words (e.g. hamming distance, character Dice score, etc.) to find, for each named entity, the correspondent resource in the Knowledge Base. Differently, the use of Word Embeddings representation will provide more meaningful high-level similarity relations between named entities and entries in the Knowledge Base, resulting in expected increased results and coverage.

5.2.1 Word Embeddings for Named Entity Linking

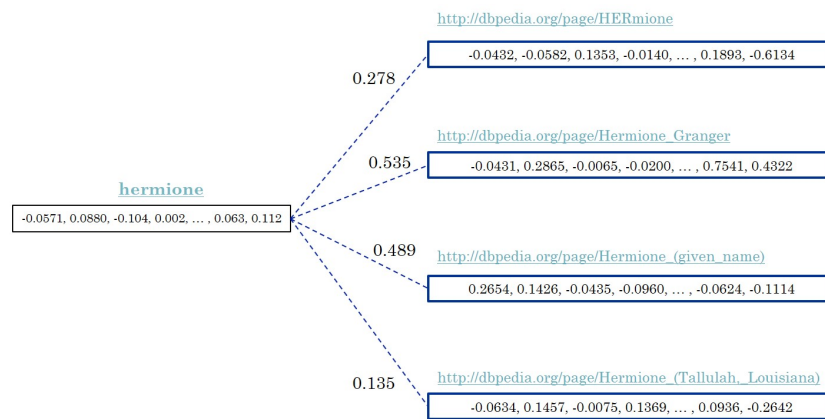
The most challenging factor typically faced when performing the Named Entity Linking task over user-generated content is related to the language of Web 2.0. While good results have been obtained on news articles or other well-written contents [137, 138], the achievement of equally accurate results for social media content is still a long way off [139]. However, the rich information provided by these data sources has received a lot of attention, driving both industrial and scientific community to develop automatic Information Extraction process.

This section introduces the model proposed in [140] for the exploitation of Named Entity Linking task in a micro-blogging environment. The proposed approach makes use of Word Embeddings models to produce a dense real-valued representation of words and KB resources, improving the semantic similarity of the input and desired output of NEL systems. Moreover, the proposed methodology has been integrated in a real-time system for the analysis of user-generated content. This system is described in Appendix A.

Most of the approaches in literature are based on surface form similarity between the words composing named entities and the unambiguous units identified as KB resources. However, these solutions are subjected to errors when dealing with microblogs posts, which are richly characterized by misspelling, abbreviations, and other noisy forms of text. Using the joint representation obtained with Word Embeddings models, the similarity measure will gain on semantic expressiveness resulting in a more accurate discrimination of the entities. Let us consider the



(a) Bag-of-words representation.



(b) Word Embeddings representation.

FIGURE 5.6: Example of Named Entity Linking similarity computation.

tweet “@EmmaWatson no1 can play hermione better than u” and in particular the case of linking the entity mention “hermione”. This ambiguous named entity can be disambiguated and consequently associated with several possible unambiguous entity candidates (e.g. with respect to DBpedia), comprising the correct one related to the character of Hermione Granger. Figure 5.6 reports two possible scheme representations, one using the common text representation bag-of-words and the other using the more meaningful textual distributional representation of Word Embeddings. The numbers in the boxes represent the numerical vector representation associated

with the text, i.e. the tweet text on the left and the textual description of the candidate KB resources on the right. The use of bag-of-words representation has been reported in Figure 5.6(a), highlighting in light blue the presence (1) of the word “hermione” in each box. It is possible to note that the bag-of-words representation is very sparse, resulting in a noisy similarity measure which corresponds to 0.00021 with respect to the correct KB resource. Otherwise, the representation derived from Word Embeddings models (Fig. 5.6(b)) permits to correctly rank as first the correct KB resource (*Hermione Granger*) with a similarity score of 0.535, as it provides a metric that better expresses the semantic properties for words and entities and consequently the similarity between them.

Following, the proposed model for the exploitation of Word Embeddings representation is described in Section 5.2.1.1. Then, Section 5.2.1.2 describes the developed framework, giving an overview of all the module components. The evaluation results on three benchmark datasets are reported in Section 5.2.1.3. Finally, Section 5.2.1.4 provides an overview of the state of the art approaches.

5.2.1.1 Representation and Linking model

The task of Named Entity Linking (NEL) is defined as associating an entity mention $t_i \in \Omega_T$, where $t_i = \{w_1^i, \dots, w_n^i\}$, with an appropriate KB candidate resource k_j from a set of candidate resources $K = \{k_1, k_2, \dots, k_{n_k}\}$.

The main contribution consists in creating a Word Embeddings model that is able to learn a heterogeneous representation space where similarities between KB resources and words can be compared. In particular, given a Word Embeddings training set composed of a large but finite set of words denoting a vocabulary V and the set Ω_E of KB resources, the Word Embeddings model can be expressed as a mapping function $C' : \Gamma \rightarrow \mathbb{R}^m$ with $\Gamma = V \cup \Omega_E$. In this way, the embedding function will be trained on a heterogeneous space of KB resources and words, by ensuring the possibility of inferring the embedded representation from the same model. More details about the training data of Word Embeddings are given in Section 5.2.1.3.

Given an entity t_i and a KB resource k_j , the similarity function s_C can be written as:

$$s_C(t_i, k_j) = \text{sim}(C'(t_i), C'(k_j)), \quad (5.15)$$

where sim , in this study, corresponds to the cosine similarity.

Given an entity t_i , the candidate resource set K is created by taking the top- n_k KB resources k_j ranked by the similarity score $s_C(t_i, k_j)$. The predicted KB resource k^* is then the k_j with the highest similarity score. If K is an empty set, t_i is considered as a NIL entity.

It is worthy to consider the case of multi-word named entities, i.e. entities composed by two

or more words, defined as $t_i = \{w_1^i, \dots, w_n^i\}$. Since words can be considered as point in an m -dimensional feature space, the top- n_k similar KB resources will be the set K that minimizes the distance between k_j and all the entity mention words.

5.2.1.2 Experimental Settings

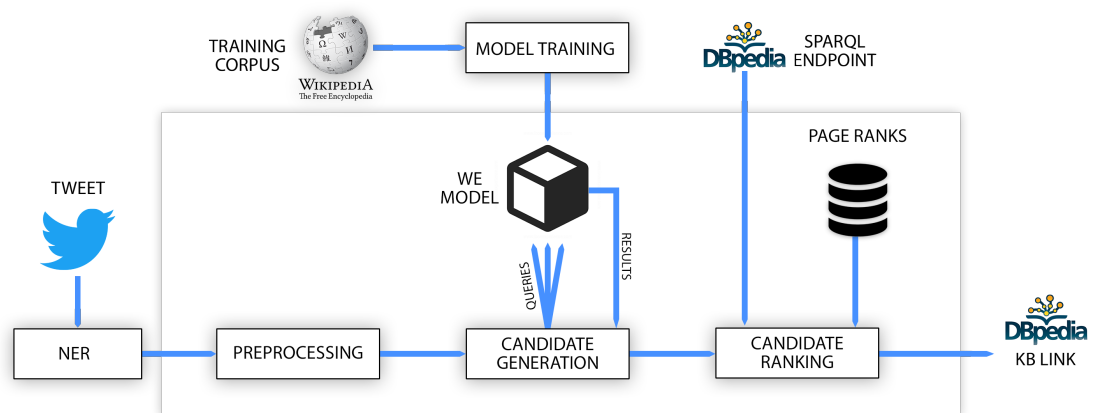


FIGURE 5.7: Pipeline of the proposed Named Entity Linking framework.

For performing the experiments, the system proposed in Figure 5.7 has been implemented, starting from the input named entities (extracted by a NER system from user-generated content) to the output (KB resources). Following, each module of the pipeline is described for a broader understanding.

Model Training In order to obtain a Word Embeddings model able to map both words and KB resources in the same representation space, its training process has been performed over a corpus that comprises both of them. For this reason, the most recent dump of Wikipedia (at the time of writing) has been considered as the training set. The structure of a Wikipedia article fits well the model’s needs since an entity can be directly associated with the corresponding Wikipedia article title. The following snippet reports a sentence from the Wikipedia page of Harry Potter and the Philosopher’s Stone related to the character of Hermione Granger.

“... he quickly befriends fellow first-year Ronald Weasley and Hermione Granger ...”

In this sentence it is possible to identify two named entities (Ronald Weasley and Hermione Granger) that, thanks to the favorable article structure, are represented as a link to their Wikipedia articles which corresponds to https://en.wikipedia.org/wiki/Ron_Weasley and https://en.wikipedia.org/wiki/Hermione_Granger respectively. The training corpus is then

obtained by merging and processing all the Wikipedia articles by specifically identifying each KB resources with the tag “KB_ID/” that corresponds to the article links (e.g. “KB_ID/Hermione_Granger”). After this process, the previous sentence will result as:

“... he quickly befriends fellow first-year *KB_ID/Ron_Weasley* and
KB_ID/Hermione_Granger ...”

Then, the Skip-Gram model detailed in Section 4.1.3 has been used as effective Word Embeddings model for learning the function $C' : \Gamma \rightarrow \mathbb{R}^m$.

Given a sequence s_1, \dots, s_{n_T} containing words and KB resources, the objective function of the Skip-gram model is defined as:

$$\mathcal{L}_{Skip-gram} = \frac{1}{n_T} \sum_{i=1}^T \sum_{-k \leq j \leq k, j \neq 0} \log P(s_{i+j} | s_i) \quad (5.16)$$

where $s_i \in \{s_1, \dots, s_T\}$ and n_T is the sequence length.

Given the large amount of data comprised in the Wikipedia dump, the processing and the learning process of the Skip-Gram model have been performed using the efficient Wiki2Vec tool [141], a software developed using Scala, Hadoop, and Spark that processes a large amount of text and makes it usable for our requirements. Regarding the Knowledge Base choices, it is important to mention that, given the large amount of publicly available information particularly suitable for the proposed model, Wikipedia has been used for training the Word Embeddings model. Analogously to what has been done in numerous studies and organized challenges in the state of the art [117, 125], DBpedia has been used as structured background knowledge for the the Named Entity Linking process. Nevertheless, since DBpedia is a a large-scale Knowledge Base built by extracting structured data from Wikipedia [142], there is a correspondence between the entities included in these two KBs.

Preprocessing Since the input entity mention is originated from a microblog post, it is expected to increase the number of correctly linked named entities by performing textual preprocessing because of its noisy nature. Common preprocessing involves capitalization solving and typographical error correction, such as missing spaces or wrong word separators. Moreover, for improving the retrieval performance, some query expansion techniques have been adopted, i.e. appending the “KB_ID/” token before the named entity.

Candidates Generation, Ranking and NIL Prediction As presented in Section 5.2.1.1, the candidate generation process is performed by computing a similarity measure between the entity

mention t_i and all the words or resources present in the Word Embeddings training set. As for any NNLM, the fundamental property is that semantically similar words have a similar vector representation. Given an entity mention t_i , the model returns the candidate resource set K composed of the top- n_k similar KB resources or words ranked by the similarity measure s_C , from which only the KB resources k_j are extracted.

The candidate resource set can be further reduced by considering the ontology type initially inferred by a given NER model. In particular, this reduction has been performed by considering only the KB resources k_j that have the same ontology type of the named entity t_i . While the ontology type of k_j can be obtained by querying the Knowledge Base, the one of t_i can be inferred with a given NER model⁴.

Finally, the candidate k^* that has the highest similarity score compared to the entity t_i is selected as the predicted KB resource.

In the proposed system, an entity mention t_i has been considered as a NIL entity, if either the similarity between t_i and the predicted resource k^* is lower than a threshold or t_i is not present in the Word Embeddings training set.

5.2.1.3 Experimental Results

This section discusses the datasets and the performance measures involved in the evaluation of the proposed NEL system.

Datasets The datasets adopted for the system evaluation are the same as the ones used in Section 5.1.1.2, i.e. #Microposts2015 and #Microposts2016 datasets divulged by the Making Sense of Microposts challenge [117]. Moreover, a dataset of Twitter posts in the Italian language, as promoted by the NEEL-IT challenge organized by EVALITA [143], has been also considered. In this study, all the datasets provided by the challenges (i.e. Training, Test, Dev) have been used to perform the evaluation.

In Table 5.15, several statistics for both English and Italian micropost challenge datasets are reported. The tables contain the total number of entities, the number of linkable entities, and the number of NIL entities.

Performance Measures The performance measures commonly used to evaluate NER systems have the same terminology as the ones presented for Named Entity Classification task but with

⁴In the experimental investigation, the considered NER model is the one proposed by Ritter et al. [109], which has been specifically designed for dealing with user-generated content.

TABLE 5.15: Datasets statistics.

	#Micropost2015			#Micropost2016		
	# Entities	# Linkable Entities	# NIL Entities	# Entities	# Linkable Entities	# NIL Entities
Training	4016	3565	451	8665	6374	2291
Dev	790	428	362	338	253	85
Test	3860	2382	1478	1022	738	284

EVALITA NEEL-IT 2016			
	Num Entities	Linkable Entities	NIL Entities
Training	787	520	267
Test	357	226	131

a slightly different meaning. NEL systems are evaluated using Strong Link Match (SLM) [117]. Given the ground truth (GT), a pair $\langle t_i, k_i^j \rangle$ can be considered as:

- True Positive (TP): if the system correctly recognizes the link of the entity.
- False Positive (FP): if the link recognized by the system is different by the one in the GS.
- True Negative (TN): if the link is not recognized by the system and in the GS. In this case the link is NIL.
- False Negative (FN): if the system recognizes the entity, but the entity is not recognized by the GS. In other words, the system returns NIL but the GT has a link.

Using these definitions, the following performance for the SLM score, i.e. *Precision*, *Recall* and *F-measure*, can be defined as presented in Section 5.1.1.2 by Equations (5.3) – (5.6). In addition, NEL systems usually measure the *NIL Score*, as the equivalent to the Recall for the NIL labeled entities.

Experimental Evaluation In this section, the results achieved by the proposed approach are introduced, analyzed and presented showing the impact of the different pipeline components on the performance measures. In particular, the system has been investigated by considering three different configurations: without preprocessing, with preprocessing and by including the ontology type into the candidate generation process.

Finally, a comparison with the available state of the art approaches is discussed.

Results In Tables 5.16, 5.17, and 5.18, the results of the proposed approach without preprocessing for both English and Italian challenges are shown. As it is possible to notice, the results are promising, achieving an overall *F-measure* of 40%.

TABLE 5.16: Results for #Micropost2015 without preprocessing.

SLM scores for #Micropost2015				
	Precision	Recall	F-measure	NIL Score
Training	0.4604	0.5186	0.4877	0.7472
Dev	0.2265	0.4182	0.2939	0.8895
Test	0.3370	0.5417	0.4168	0.8748

TABLE 5.17: Results for #Micropost2016 without preprocessing.

SLM scores for #Micropost2016				
	Precision	Recall	F-measure	NIL Score
Training	0.3840	0.5221	0.4425	0.8520
Dev	0.3461	0.4624	0.3959	0.8235
Test	0.2563	0.3550	0.2977	0.8380

TABLE 5.18: Results for NEEL-IT 2016 without preprocessing.

SLM scores for EVALITA NEEL-IT 2016				
	Precision	Recall	F-measure	NIL Score
Training	0.2477	0.3750	0.2983	0.6779
Test	0.2380	0.3761	0.2915	0.5954

In order to deal with the variety of problems related to the language register of Web 2.0, the **preprocessing** step has been performed. The results are reported in Tables 5.19 and 5.20. As expected, all the performance measures have been increased of 10%-15% with respect to previous experimental settings. An example of correctly linked entity mention after preprocessing is “*fl*”: in the baseline experiment it has been labeled with the wrong link “*dbpedia.org/resource/Family_1*”, while, if properly capitalized in “*FI*”, the result is the correct link “*dbpedia.org/resource/Formula_One*”. Another example for the Italian dataset (Table 5.21) regards the entity “*FEDEZ*”, an Italian singer, that has been linked to a NIL entity in the baseline. By performing the capitalization resolution, the model is able to correctly link the entity to “*dbpedia.org/resource/Fedez*”.

In spite of the overall performance improvements, for some entities, the preprocessing module associates erroneous links that were correctly given by the baseline method. An example is the entity “*repubblicait*”, from Italian tweets, which is the account of an Italian newspaper called “La Repubblica”, that after the preprocessing step is determined as NIL.

TABLE 5.19: Results for #Micropost2015 with preprocessing.

SLM scores for #Micropost2015				
	Precision	Recall	F-measure	NIL Score
Training	0.53	0.60	0.56	0.72
Dev	0.28	0.53	0.37	0.87
Test	0.40	0.65	0.50	0.86

TABLE 5.20: Results for #Micropost2016 with preprocessing.

SLM scores for #Micropost2016				
	Precision	Recall	F-measure	NIL Score
Training	0.45	0.61	0.52	0.84
Dev	0.53	0.71	0.61	0.78
Test	0.43	0.59	0.50	0.82

TABLE 5.21: Results for NEEL-IT 2016 with preprocessing.

SLM scores for EVALITA NEEL-IT 2016				
	Precision	Recall	F-measure	NIL Score
Training	0.3100	0.4692	0.3733	0.6479
Test	0.2689	0.4247	0.3293	0.6183

Another investigated experimental setting consists of considering the **ontology type** of the entity mention in the candidate generation process, taking advantage of the preprocessing step. The ontology type can be obtained by performing a type classification with a Named Entity Recognition and Classification method. It is expected that considering the ontology type of entity will help the linking process. For instance, given the entity mention “Paris”, the corresponding inferred type *Person* will contribute to link “Paris” to the celebrity Paris Hilton, instead of the France capital. The achieved results are shown in Tables 5.22, 5.23, and 5.24.

TABLE 5.22: Results for #Micropost2015 with preprocessing and considering entity types.

SLM scores for #Micropost2015				
	Precision	Recall	F-measure	NIL Score
Training	0.5333	0.6008	0.5650	0.7184
Dev	0.2860	0.5280	0.3711	0.8729
Test	0.4015	0.6507	0.4966	0.8626

TABLE 5.23: Results for #Micropost2016 with preprocessing and considering entity types.

SLM scores for #Micropost2016				
	Precision	Recall	F-measure	NIL Score
Training	0.4520	0.6145	0.5209	0.8358
Dev	0.5355	0.7154	0.6125	0.7764
Test	0.4015	0.6507	0.4966	0.8626

TABLE 5.24: Results for NEEL-IT 2016 with preprocessing and considering entity types.

SLM scores for EVALITA NEEL-IT 2016				
	Precision	Recall	F-measure	NIL Score
Training	0.3672	0.5557	0.4422	0.6479
Test	0.3165	0.5000	0.3876	0.6183

Differently from the expectations, with the introduction of the entity types, the performance have barely improved with respect to the configuration with only preprocessing (Tables 5.19, 5.20 and 5.21). From a practical point of view, this behavior can be justified by the fact that the

candidate resources k_j of an entity mention t_i are mostly related to the same ontology class, do not providing any additional discriminative information, e.g. the most similar entities to a birth name will very likely be of type *Person*.

A small improvement in terms of F-measure can be observed when the candidate list is composed of resources with different ontology types. An example is the entity mention “*Interstellar*”: the first match of the system based only on the preprocessing step is “*dbpedia.org/resource/Interstellar_travel*”, while including the entity type *Product* gives the correct resource “*dbpedia.org/resource/Interstellar_(film)*”.

State of the art comparison This section presents a comparison between the proposed NEL system and the current state of the art solutions. Tables 5.25 and 5.26 report a comparison of the proposed approach with the state of the art (only those approaches providing individual results for the specific NEL task have been considered). From the results, it is possible to notice that the proposed system (**UNIMIB-WE**) has comparable performance to the top performant systems proposed at #Micropost challenges. In the #Micropost2015 challenge UNIMIB-WE places in the third position, close to the solution proposed by *Acubelab* [144] in second place. In the 2016 edition, UNIMIB-WE achieves the second place, with 60% of F-measure. The main reason why the proposed system is overcome by *KEA* [145] regards the specific optimization that this model has performed on the challenge dataset, in fact this domain-specific optimization process induced an increase of 40% in terms of F-measure compared to the not optimized version. Similarly, the *Ousia* [146] model beyond the exploitation of an ad-hoc acronym expansion dictionary, is a supervised learning approach.

In spite of the better-achieved results, these models have the main problem of limited generalization abilities and the need of a manually label dataset, which is very expensive in terms of human effort. Differently, the proposed NEL system does not need any supervision or labeled dataset and, given the wider range of named entities that can cover, it provides good generalization abilities to other domains.

Also for the EVALITA NEEL-IT challenge, Table 5.27 reports the results related to the participants that provided the specific NEL performance. Regarding the comparison with the model proposed in [147], UNIMIB-WE obtains similar performance in terms of F-measure, but they differ in terms of Precision and Recall. UNIMIB-WE is less precise, but it has a higher Recall. The same performance gap occurs when comparing with the *sisinflab*'s solution [148], in this case, the higher Precision is due to the combined three different approaches they used in the NEL system. They use DBpedia Spotlight for span and URI detection, DBpedia lookup for URI

generation given a keyword, and a Word Embeddings model trained over tweets with a URI generator. Both of these solutions use an ensemble of state of the art techniques, this gives them the ability to overcome the problems of individual methods and achieve better overall performance.

TABLE 5.25: Comparison for #Micropost2015 sorted by F-measure.

#Micropost2015 Test set		
Team Name	Reference	F-measure
Ousia	[146]	0.7620
Acubelab	[144]	0.5230
UNIMIB-WE	[140]	0.5059
UNIBA	[149]	0.4640

TABLE 5.26: Comparison for #Micropost2016 sorted by F-measure.

#Micropost2016 Dev set				
Team Name	Reference	Precision	Recall	F-measure
KEA	[145]	0.6670	0.8620	0.7520
UNIMIB-WE	[140]	0.5295	0.7075	0.6057
MIT Lincoln Lab	[150]	0.7990	0.4180	0.5490
Kanopy4Tweets	[151]	0.4910	0.3240	0.3900

TABLE 5.27: Comparison for NEEL-IT 2016.

EVALITA NEEL-IT				
Team Name	Reference	Precision	Recall	F-measure
FBK-NLP (train)	[147]	0.5980	0.4540	0.5160
UNIMIB-WE (train)	[140]	0.4231	0.6403	0.5095
UNIMIB-WE (test)	[140]	0.3529	0.5575	0.4322
sisinflab (test)	[148]	0.5770	0.2800	0.3800

As a conclusion, it is possible to state that the results obtained by the proposed model are very promising, given the highly challenging environment of user-generated content over microblogging platforms. This supports the evidence of Word Embeddings as providers of semantically meaningful word representation. The model would certainly gain with the addition of a supervision procedure able to learn which module should be used with respect to the similarity score. For instance, if the similarity score between “*dbpedia.org/resource/La_Repubblica_(quotidiano)*” and “*repubblicait*” is higher than the one to “*RepubblicaIT*”, the capitalization module would not be activated.

5.2.1.4 Related Works

The survey presented in [152] distinguishes the Named Entity Linking task in three different steps:

- Candidate Entity Generation, which is aimed at extracting for each entity mention a set of candidates resources;

- Candidate Entity Ranking, focused on finding the most likely link among the candidate resources for the entity mention.
- Unlinkable Mention Prediction, which has the goal of predicting those mentions that cannot be associated with any resource in the KB. This step corresponds to what has been called so far as NIL prediction.

Candidate Entity Generation The candidate generation step is a critical subprocess for the success of any NEL system. According to experiments conducted by Hachey et al. [153], a more precise candidate generator can also imply improved linking performance.

In the literature, candidate generation techniques can be mainly distinguished in Name Dictionary and Search Engine based methods. The former consists in constructing a dictionary-based structure where one or more KB resources are associated with a given named entity (dictionary key) based on some useful features available in the KB, such as redirect pages, disambiguation pages, bold phrases, etc. [154–156]. Given an entity mention extracted from text, the set of its candidate entities is obtained by using exact matching or partial matching with the corresponding dictionary keys [145]. An alternative solution for Candidate Entity Generation is represented by Search Engine based techniques, which make use of Web search engines for retrieving the list of candidate resources associated with an entity mention [157–159].

Candidate Entity Ranking After the candidates' extraction, the list of candidates should be ranked in order to extract the most probable one. Most of the approaches are mainly based on Machine Learning algorithms for learning how to rank the candidate entities [149, 160–162]. These approaches usually consider several features related to the named entity or the KB entry, such as *entity popularity*, the *ontology type* extracted by NER systems and vector-based representation of the context surrounding the named entity. Beyond Machine Learning models, it has also been proved that the combination of different features can be useful for ranking the mention candidates [163].

Unlinkable Mention Prediction An entity mention does not always have a corresponding entity in the KB, therefore systems have to deal with the problem of predicting NIL entities (unlinkable mentions). Some approaches [160] use a simple heuristic to predict unlinkable entity mentions. If it is not possible to retrieve any candidate for an entity mention, then the entity mention is unlinkable. Many NEL systems are based on a threshold method to predict the unlinkable entity mention [164–169]. In these systems, each ranked candidate is associated to a score and if the score is lower than a given threshold, then the entity mention is considered a NIL. The NIL prediction can be also accomplished using approaches based on supervised Machine Learning, such as binary classification techniques [154, 170].

As stated above, the candidate generation is a crucial part for any NEL task. The process of generating the candidate resource set for a given entity mention is usually obtained by exact or partial matching between the entity mention and the labels of all the resources in the KB. However, these approaches can be error-prone, especially when dealing with microblog posts that are rich in misspellings, abbreviations, nicknames and other noisy forms of text. In order to deal with these issues, the proposed NEL approach has been defined to exploit a similarity measure between the high-level representation of entity mentions and KB resources. These meaningful and dense representation of entity mentions and KB resources has been obtained by taking advantage of one of the most widely used neural network language models, i.e. Word Embeddings [2], described in Section 4.1.

5.3 Sentiment Analysis

Sentiment Analysis (SA) is the research field which focuses on analyzing people's opinions, sentiments, evaluations, expectations, behaviors and emotions related to entities like products, services, organizations, individuals, problems, events and their aspects [20].

As Social Media platforms have become a form of electronic word-of-mouth for sharing the afore-mentioned different facets of user opinions, researchers from academia and companies started to exploit them in order to extract innovative insights and to obtain a competitive advantages for business [171]. As the definition of SA suggests, mining opinions about specific *entities* can be of great importance, broadening its impact beyond its traditional application [103]. For example, a company or a public figure, such as a politician, can be interest in knowing the people's opinion about its/his/her public image. It is then desirable that one would be able to retrieve only specific messages, e.g. the *Ford* company would not be interested in messages about the actor *Harrison Ford*, in the same way *Obama* would like to know the opinion about the 44th President of the United States and not about his wife *Michelle Obama*.

Traditional approaches usually investigate the sentiment analysis problem as a supervised polarity classification task, where polarity refers to the commonly adopted definition of sentiment as *positive*, *negative* or *neutral*. However, adopting supervised approaches for Social Media user-generated content has to deal with several issues related to languages and domains. Since the adopted writing style on Web 2.0 platforms is challenging and dynamic, it would be very difficult to find manually labeled datasets constantly updated for referring to new sentiment terminologies and implications. For example, the occurrence of an event can completely change the language and the polarity of words, e.g. a scandal in politics. Another issue concerns the generalization abilities, as supervised models are highly domain-dependent and usually achieve poor performance over other domains [172]. Each domain can be characterized by specific

polarity-driven words that do not appear in other domains or have a different polarity orientation. For example, the word “*unpredictable*” can have a positive orientation if related to a book, while it has a negative connotation if related to a car, as an “unpredictable car” suggests a vehicle that has unexpected and dangerous behaviors. Therefore, a Machine Learning model trained to classify the sentiment polarity on the textual reviews of *Books* may tend to erroneously classify the sentiment of reviews regarding *Cars*.

This thesis presents an approach, introduced in [173], that addresses the problem of **domain adaptation** for **sentiment classification** by combining **Deep Learning** and **Ensemble Learning** methods. While Deep Learning allows us to acquire a cross-domain high-level feature representation, Ensemble Learning methods permits to reduce the cross-domain generalization error. As presented in Chapter 3, the peculiarity of Deep Learning, and Representation Learning in general, is its ability to identify and disentangle the underlying explanatory factors hidden in the observed data. This property is crucial for increasing the generalization abilities, that in this case corresponds to the ability to interpret words in their domain context. Indeed, in the proposed approach, a Deep Learning architecture is used in order to extract the vector representation of natural language text. Then, since in Natural Language Processing there is no agreement on which Machine Learning model is the best one for addressing the sentiment analysis task, ensemble methods have been adopted for the classification purposes.

5.3.1 Deep Learning Representation and Ensemble Learning methods for Domain Adaptation in Sentiment Classification

The problem of domain adaptation is particularly relevant in Sentiment Analysis where one has to make predictions on examples in a new domain with little or no available labeled data. This is a common situation for user-generated content, where the amount of data is too large to be manually handled. Consider for example a situation in which some product reviews are available in one domain (e.g. *Kitchen appliances*), but the task is to predict user opinions on products from a different domain (e.g. *DVD*). This implies that the feature (word) distribution of the training set can be different from the feature distribution of the testing set. For instance, the word “flat” can be considered differently in the context of *Books* or *Electronics*: a book can be characterized by a “flat storyline”, while a “flat screen” is an electronic device. Domain adaptation techniques are therefore applied to leverage the data available in the *source* domain to improve the Accuracy of the model when testing in the *target* domain.

In order to deal with the problem related to different words distribution among source and target domains, Deep Learning and Ensemble Learning are combined to address the problem of domain adaptation for Sentiment Analysis. In particular, the scenario where the testing target domain is completely unlabeled has been considered.

The following sections are organized as follows. Section 5.3.1.1 presents the proposed model. Section 5.3.1.2 describes the experimental settings, detailing the evaluated dataset, the compared models and the performance measures. Then, Section 5.3.1.3 shows the obtained experimental results and comparison. Finally, Section 5.3.1.4 discusses some related works.

5.3.1.1 Deep Learning Representation and Ensemble Learning model

The domain adaptation problem arises when training instances are provided only for a source domain S and not for a target domain T . The learning problem consists then in finding a function f that is able to transfer the knowledge from S to T .

In the proposed framework, the data have been treated as unlabeled (for both the source and target dataset) in order to learn a common feature representation (meaningful across both domains). Once the high-level representation has been obtained, Ensemble Learning models are used for accomplishing the sentiment classification task.

Deep Learning In the recent literature, there has been an increasing number of investigations on Deep Learning architectures for domain adaptation purposes. In this context, a particular class of Artificial Neural Network, called Auto-encoder, plays a fundamental role for creating a good representation across domains (Sec. 4.2).

The proposed framework makes use of the **marginalized Stacked Denoising Auto-encoder** (mSDA) [36], presented in Section 5.3.1.1. This model has been chosen due to its abilities to preserve the strong feature learning capabilities of SDA [37], while providing several gains in terms of efficiency.

Ensemble Learning In Machine Learning, most of the domain adaptation approaches are based on the classical statistical inference paradigm, where a single model (over a set of hypotheses) is selected to learn from the training data in the source domain and to make predictions on the target domain data. This may lead to over-confident inferences and decisions that do not take into account the inherent uncertainty of the natural language. Instead, the idea behind an ensemble mechanism is to exploit the characteristics of several independent models by combining them in order to achieve higher performance. When combining multiple independent and diverse decisions, where each one is at least as accurate as random guessing, the effect of random errors is reduced, resulting in a reinforcement of correct decisions [174]. To achieve a good ensemble, two necessary conditions should be satisfied: Accuracy and prediction diversity. The strength of this technique in NLP tasks has been proved by several studies in the literature [175–177].

In order to investigate the role of Ensemble Learning with respect to domain adaptation problems, several ensemble methods have been analyzed.

Concerning the paradigm “*same learner type, different samples of source training data*”, the following ensemble models have been considered:

- *Bagging* [178]. Bagging is aimed at bootstrapping replicas of the training set to discriminate the learner. Different training data subsets are randomly drawn (with replacement) from the training dataset in the source domain, to finally induce several learners of the same type.
- *Boosting* [179]. Boosting incrementally builds an ensemble by training each new model to emphasize those instances of the source domain that the previous models have misclassified.
- *Random SubSpace* [180]. The learner is trained on randomly chosen subspaces of the original input space, then the relative outputs are combined.

Regarding the paradigm “*different learner types, same sample of source training data*”, the most popular ensemble method has been considered:

- *Simple Voting* [181]. Simple Voting is characterized by a set of different learners induced on the same training data.

All these models classify the target instances by considering the vote of each learner as equally important and determine the final label by selecting the most popular prediction.

5.3.1.2 Experimental Settings

Dataset The proposed work evaluates the contribution of Deep Learning and Ensemble Learning methods for predicting the sentiment while investigating domain adaptation issues. The benchmark dataset, called *Amazon reviews*¹ [182], contains more than 340,000 reviews from 25 different types of products from Amazon.com. Following the convention of Glorot et al. [85], the binary classification problem has been investigated to distinguish positive reviews (i.e. reviews with a rate greater than 3) from negative ones (i.e. reviews with a rate lower than 3). As in the work of Blitzer et al. [182], the experiments have been conducted by starting from a bag-of-words representation characterized by the 5,000 most frequent terms of unigrams and bigrams extracted from the text reviews.

To counter the effect of class- and size-imbalance, and to ensure a fair comparison with the state of the art approaches, a more controlled and smaller dataset has been used. The final dataset

used for the experimental comparison contains reviews of four domains: books (B), DVDs (D), electronics (E), and kitchen appliances (K). Each domain contains 2,000 labeled instances, perfectly balanced between positive and negative reviews, and a large amount of unlabeled data.

Compared Models The *baseline* is a reference model trained on the bag-of-words space of the source data and tested on the target data (without adaptation). In the following, the baseline model is denoted with *Baseline(learner)*, e.g. *Baseline(SVM)*, which means that the Machine Learning model between the brackets is the reference model. This baseline is useful for evaluating the improvement of applying domain adaptation on models that are trained and tested on different domains. Moreover, in order to evaluate if acquiring knowledge from other domains can provide advantages, a linear Support Vector Machine (SVM) trained and tested on the bag-of-words features of the same domain have been considered for comparison. This model has been denoted as *gold standard*.

Table 5.28 reports the performance in terms of Accuracy of the gold standard model and the *Baseline(SVM)* model. The first column lists the source domains (reporting the initials) and the first row the target domains. Each cell (i, j) corresponds to the Accuracy of the model trained on the source domain data (row i) and tested on the target domain data (column j). Since the gold standard model is trained and tested on the same domain, the grey cells on the diagonal report its performance. Hence, the remaining cells contain the results of the *Baseline(SVM)*.

TABLE 5.28: Accuracy of the baseline and gold standard. Results achieved by the gold standard are reported in the diagonal, while the performance of the *Baseline(SVM)* are given in the remaining cells.

	B	D	E	K
B →	0.784	0.753	0.736	0.751
D →	0.737	0.784	0.701	0.732
E →	0.671	0.682	0.836	0.822
K →	0.701	0.721	0.807	0.860

Regarding the *Ensemble Learning methods*, several learners have been considered: Naïve Bayes (NB) [183], Support Vector Machine (SVM) [184], Voted Perceptron (VP) [185], Decision Tree (DT) [186], Logistic Regression (LR) [187], K-Nearest Neighbor (KNN) [188] and Random Forest (RF) [189]. All the considered ensemble methods, i.e. Bagging, Boosting, Random Subspace Method and Simple Voting, have been derived by taking advantage of the most promising Deep Learning model (i.e. mSDA).

In order to compare the proposed framework, several state of the art models are considered, i.e. Blitzer et al. [190], Glorot et al. [85] and Chen et al. [36].

¹The dataset is available at <https://www.cs.jhu.edu/~mdredze/datasets/sentiment/>

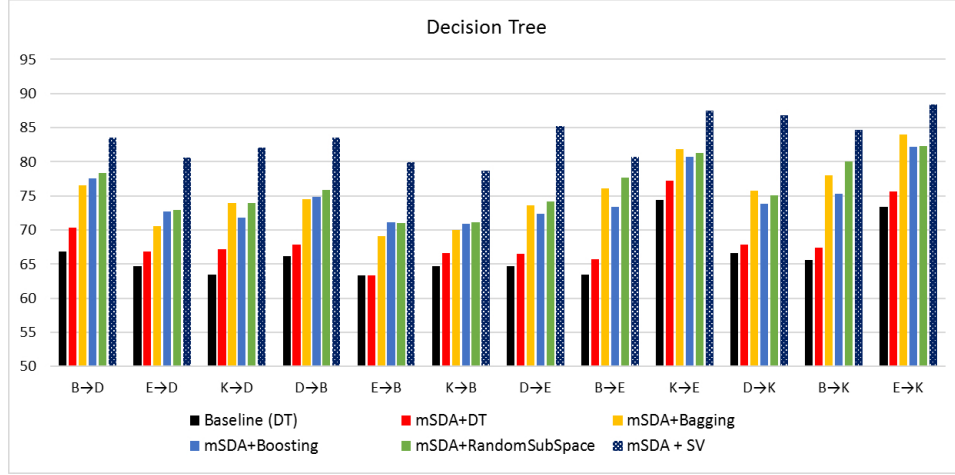


FIGURE 5.8: Accuracy of ensemble methods based on Decision Tree (DT).

Performance Measures The performance evaluation measures used in this Section are based on the *classification error* defined as:

$$\text{classification error} = \frac{FP + FN}{TP + FP + TN + FN} \quad (5.17)$$

It is now possible to estimate the following state of the art error metrics originally introduced in [85] as:

- *transfer error* $e(S, T)$, i.e. the classification error of a model trained on the source domain S and tested on the target domain T ;
- *in-domain error* $e(T, T)$, i.e. the classification error of a model trained and tested on the target domain T ;
- *gold standard error* $e_g(T, T)$, i.e. the error obtained by the gold standard (linear SVM trained and tested on the raw features of the target domain T);

Given the above-mentioned metrics, the following performance measures have been used for the domain adaptation issue:

- **Transfer loss:** it denotes the difference between the transfer error and the gold standard error, i.e.:

$$L_{\text{transfer}} = e(S, T) - e_g(T, T)$$

The lower the transfer loss is, the better the domain adaptation is.

- **Transfer ratio:** it represents the average proportion between the transfer error and the gold standard error, i.e.:

$$Q = \frac{1}{\#(S, T)} \sum_{\substack{(S, T) \\ S \neq T}} \frac{e(S, T)}{e_g(T, T)}$$

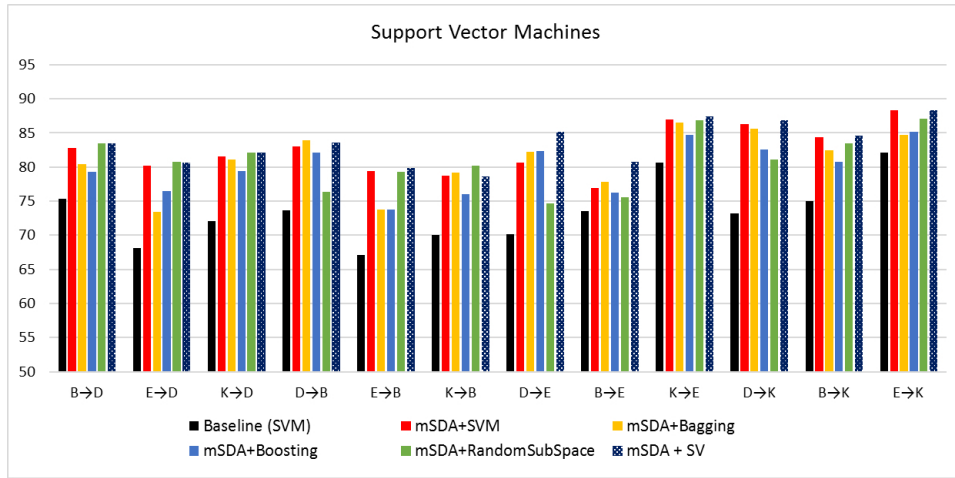


FIGURE 5.9: Accuracy of ensemble methods based on Support Vector Machines (SVM).

where $\#(S, T)$ denotes the number of source-target couples, such that $S \neq T$. Similarly to transfer loss, a lower transfer ratio implies better domain adaptation abilities.

- **In-domain ratio:** the domain adaptation method adopted in this work creates a new feature representation for all the domains, including the source. Therefore, in-domain errors of such methods are different from those of the gold standard. To this purpose, in-domain ratio measures the average value of the rates between the in-domain error of a model and the in-domain error of the gold standard, i.e.

$$I = \frac{1}{|\mathbf{T}|} \sum_{T \in \mathbf{T}} \frac{e(T, T)}{e_g(T, T)}$$

where \mathbf{T} denote the set of target domains.

Additionally to the domain adaptation evaluation measures, **Accuracy** (Eq. 5.3) has been computed for understanding the prediction abilities of the ensemble models. Concerning the testing policy, a 5-folds cross-validation has been performed.

5.3.1.3 Experimental Results

The first analysis relates to the contribution and comparison of ensemble methods. The performance of the investigated ensemble paradigms, i.e. “*same learner type, different samples of source training data*” and “*different learner types, same sample of source training data*”, has been reported in Figures 5.8 and 5.9. In particular, the results of two meaningful learners have been reported: DT is representative of all the classifiers (similar results have been obtained by NB, VP, LR, KNN and RF), while SVM has shown outperforming results compared to the others. The reasons may be related to the Machine Learning model interpretation of the input

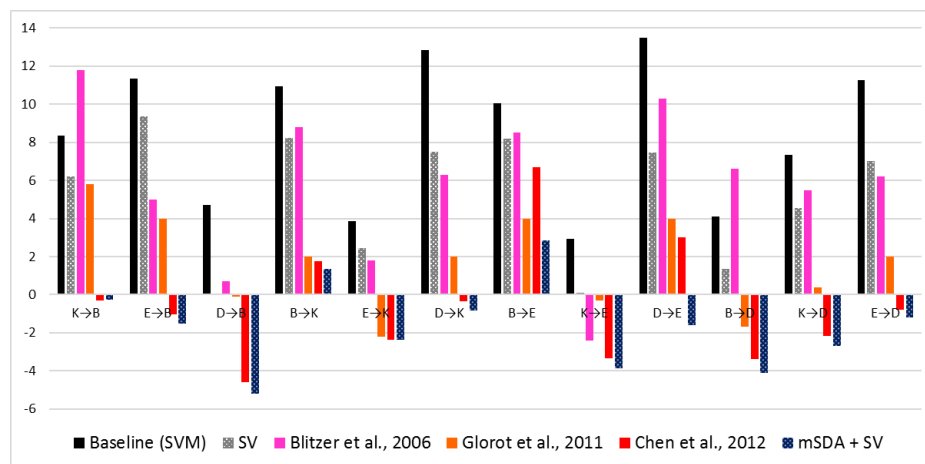


FIGURE 5.10: Transfer losses on the Amazon benchmark of 4 domains: Kitchen (K), Electronics (E), DVDs (D) and Books (B).

space, since SVM has proved to remarkably deal with real-valued vectors, that is the representation commonly obtained with Deep Learning. To support this statement, SVM is the best performing model also for the evaluation of the proposed Word Embeddings based *Learning To Adapt* method (Sec. 5.1.1.3).

The performance is presented in terms of Accuracy for Bagging, Boosting and Random SubSpace grounded on Decision Tree and Support Vector Machines, trained on the mSDA representation (mSDA+*ensemble*). Similarly, the performance of Simple Voting trained on mSDA feature space has been reported (mSDA+SV). For a further comparison, the baseline and the single learner trained on the mSDA feature space have been also reported (i.e. mSDA+DT in Figure 5.8 and mSDA+SVM in Figure 5.9).

As a general remark, it is possible to highlight that all the ensemble methods guarantee significant improvements compared to the single learner trained on the raw features (baseline) and on the mSDA representation (mSDA+*learner*). An additional consideration relates to the comparison among the different ensemble paradigms. For all the source-target couples, mSDA+SV outperforms all the other ensemble methods showing that learning heterogeneous models on the same representation is better than learning one single model on different samples of the same training data.

Since Simple Voting emerged as the most promising ensemble method compared to Bagging, Boosting and RandomSubSpace, it has been used as the reference in the subsequent experiments.

In Figure 5.10, the transfer loss of SV and mSDA+SV has been compared to Baseline(SVM) and the state of the art models, i.e. Blitzer et al. [190], Glorot et al. [85] and Chen et al. [36]. A first consideration relates to the contribution that SV is able to provide with respect to Baseline (SVM), that is the best baseline in our experimental investigation and the most used in the literature. The results clearly highlight that an ensemble of different learners has a reduced

transfer loss than a single classifier (baseline). Indeed, SV outperforms the baseline for all the 12 transfer configurations.

A significant remark is concerned with mSDA+SV. From Figure 5.10 it is possible to note that this model achieved the best adaptation in 11 out of 12 cases (the only exception is the adaptation from $K \rightarrow B$ where the original mSDA approach in Chen et al. [36] performs slightly better). More specifically, the ensemble enclosed in mSDA+SV contributes more on those domains where the original mSDA approach provides less information, i.e. $D \rightarrow E$ with a transfer loss decrease of 4.6 and $B \rightarrow E$ with a transfer loss decrease of 3.5. In 9 out of 12 cases mSDA+SV performed a negative transfer loss and, interestingly, our framework is the only one able to generate a negative transfer loss in $D \rightarrow E$. This suggests that the Ensemble Learning trained on a different domain can outperform the gold standard (the most accurate model based on SVM trained on the same domain).

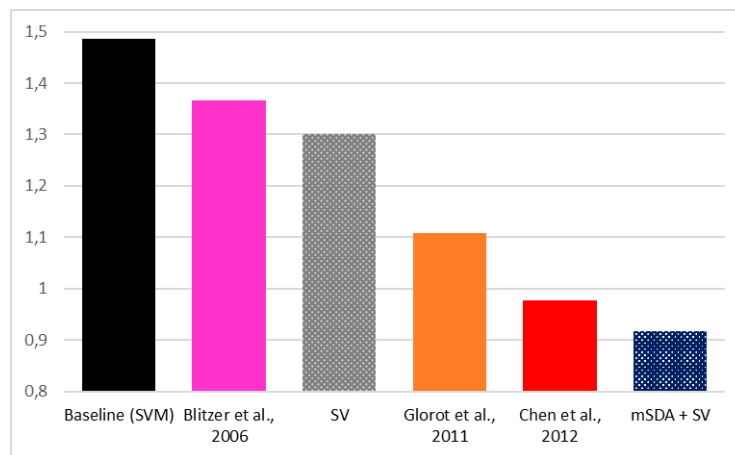


FIGURE 5.11: *Transfer ratio* on the Amazon benchmark.

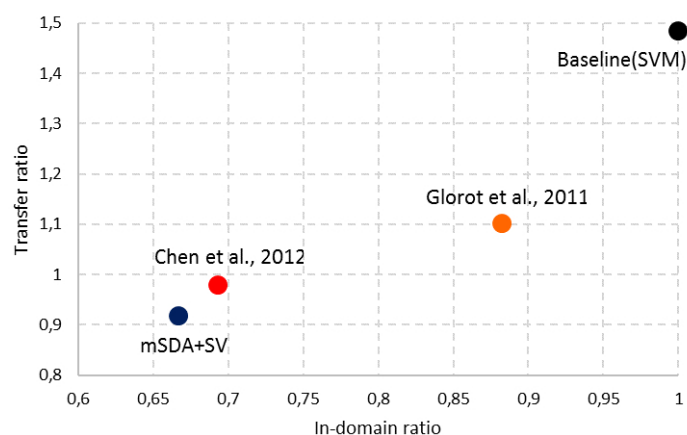


FIGURE 5.12: The *in-domain ratio* versus *transfer ratio*.

The *transfer ratio* (Figure 5.11) provides a global measure to evaluate the average performance over all source-target couples. The reported results further confirm the ability of the proposed

mSDA+SV to perform better adaptation than the state of the art methods. Moreover, the Simple Voting method evaluated on the raw feature proved to be able not only to outperform the baseline but also the Structural Correspondence Learning approach proposed by Blitzer et al. [190].

Figure 5.12 depicts the performance in terms of *transfer ratio* (where the training and testing set are related to different domains) and *in-domain ratio* (where training and testing are performed on the same domain). The results of the most promising domain adaptation techniques have been reported. It can be easily noted that mSDA+SV is not only a promising solution for domain adaptation, but it is also able to improve the in-domain recognition abilities.

As final remarks, the results obtained by the proposed method, in terms of several domain adaptation measures, show a good improvement compared to the most accurate state of the art approaches. This demonstrates the importance of jointly consider a high-level feature representation input space and ensemble models for obtaining good generalization abilities over different domains. As future work, it would be interesting to further extend the concept of Ensemble Learning to the different denoising layers of mSDA, since it is expected that they incorporate marginally different latent factors of the input data.

5.3.1.4 Related Works

Different approaches have been proposed in the literature for addressing the problem of domain adaptation for sentiment classification. To reduce the distribution gap between training and testing data several approaches investigated instance weighting techniques [191–193], which can be regarded as a generalization of semi-supervised learning methods. Basically, instance weighting techniques assign instance-dependent weights to the loss function when minimizing the expected loss over the distribution of data.

Another solution to domain adaptation is to create a novel representation of the source and target domains so that they represent the same joint distribution of observations and labels. Blitzer et al. [190] presented Structural Correspondence Learning (SCL), a method to automatically induce correspondences among features from different domains, while in [194] Daumé III proposed a kernel-mapping function for mapping the data into a high-dimensional feature space, where standard discriminative learning methods are used to train the classifiers.

More recent and effective approaches for Representation Learning are related to Deep Learning architectures, as introduced in Chapter 3. Deep Learning methods have demonstrated very promising abilities for discovering meaningful intermediate representations of the data. The work of Glorot et al. [85] proposed the exploitation of a Stacked Denoising Auto-encoder (SDA) to learn robust feature representation. Later, Chen et al. [36] derived a variation of SDA called marginalized Stacked Denoising Auto-encoder (mSDA) which adopts a greedy layer-by-layer

training of SDA. Differently from Glorot et al. [85], they used linear denoisers as the basic building blocks, leading to a reduction of the computational costs in the parameter estimation phase.

In most of the approaches investigated in the literature, both instance-based and feature-based transfer learning, a single base learner is assumed. However, within the natural language processing research field, there is no agreement on which model is the best: one learner could perform better than others in a given feature distribution, while a further method could outperform the others when dealing with a given language or linguistic register. To this purpose, Ensemble Learning could provide a leverage towards accurate predictions and robust models when dealing with domain adaptation issues. Most of the Ensemble Learning techniques, proposed for transfer learning tasks, take advantage of a small amount of target domain data used for weighting the influence of each learner model on the final ensemble composition [34, 195, 196].

5.4 Irony Detection

As introduced in the previous section, mining opinions from user-generated content is a highly difficult task. It requires a deep understanding of explicit and implicit information conveyed by language structures, whether in a single word or an entire document [197]. In particular, Web 2.0 users are inclined to adopt a creative language making use of original devices such as sarcasm and irony [198].

These figures of speech are very challenging when the sentiment polarity of the text should be inferred (Sentiment Analysis task) because they are commonly used to intentionally convey an implicit meaning that may be the opposite of the literal one. An ironic message typically conveys a negative opinion using only positive words, causing a misleading behavior of Machine Learning models [199]. From the sentiment analysis perspective, such figures of speech represent an interfering factor that can revert the message polarity (usually from positive to negative). Detecting ironic expressions can be crucial in different application domains, such as marketing and politics, where the users tend to subtly communicate dissatisfaction usually referring to a product or to a public figure such as a politician.

Traditional supervised or semi-supervised approaches to Irony Detection can incur into the same issues presented before for Sentiment Analysis prediction. The ironic orientation of words can consistently change with respect to the context as well as it works for sentiment-driven words. For example, defining someone a *clown* can be positive if referred to an actor in a comic movie, or very negative if ironically referred to politicians.

In the following sections, an **unsupervised probabilistic model** exploiting **Word Embeddings** representation for Irony Detection is presented.

5.4.1 A Probabilistic Model with Word Embeddings for Unsupervised Irony Detection

Although sarcasm and irony are a well-studied phenomenon in linguistics, psychology and cognitive science, their automatic detection is still a great challenge because of its complexity. Standard dictionary-based methods for Sentiment Analysis, based on a predefined sentiment-driven lexicon, have often shown to be inadequate in the face of indirect figurative meanings [198]. Several methods have been proposed to evaluate the abilities of semi-supervised and supervised Machine Learning approaches to tackle the irony detection problem. However, they assume as prerequisite human annotation of text as training data, which is a time and effort consuming task that becomes impossible with the exponential increase of available data. Moreover, in a real Social Media context, recognizing irony is a difficult task even for humans, making prohibitive the labeling of huge amount of data. An additional weakness of supervised approaches, as already stated in this chapter, is that it is commonly known that supervised Machine Learning classifiers trained on one domain often fail to produce satisfactory results when shifted to another domain since natural language expressions can be quite different [182].

The proposed model, presented in [200], is a fully unsupervised framework for domain-independent irony detection. To perform unsupervised topic-irony detection, an existing **probabilistic topic model**, initially introduced for sentiment analysis purposes, has been extended. The aim of this model is to discover the hidden thematic structure in large archives of text. Probabilistic topic models are particularly suitable for two main reasons: (i) they are able to discover topics embedded in text messages in an unsupervised way, and (ii) they result in a language model that estimates how much a word is related to each topic and to the irony figure of speech.

Word Embeddings have been used in order to improve the generalization abilities, in particular for obtaining domain-aware ironic orientation for words. At the time of writing, this is the first work that addresses the problem of irony detection in a fully unsupervised settings. Furthermore, this study contributes as the first investigation on irony-topic models.

The following sections are organized as follows. In Section 5.4.1.1, the proposed model grounded on an unsupervised Topic-Irony model and Word Embeddings is presented. Sections 5.4.1.2 and 5.4.1.3 show the experimental investigation, discuss the results computed on a benchmark dataset and finally report a comparison between the proposed model and state of the art approaches. Section 5.4.1.5 outlines the most recent related works.

5.4.1.1 Unsupervised Topic-Irony Model

Topic-Irony Model (TIM) In order to perform unsupervised irony detection, taking into account also the topic-dependency of the words, the investigation has been focused on the suite

of generative models called *probabilistic topic models*, originally defined for sentiment analysis purposes. Three main generative models have been considered, which are extensions of the well-known Latent Dirichlet Allocation model [201].

The first one is Topic Sentiment Mixture (TSM) [202], that jointly models the mixture of topics and sentiment predictions for the entire document. In TSM, the sentiment language model is considered as distinct from the topics ones. This assumption can lead to a language model that is not able to explain the hidden correlation between a topic and a sentiment. The second one is Joint Sentiment-Topic (JST) model [203], which assumes that topics are dependent on sentiment distributions and words are conditioned on sentiment-topic pairs. The last one is Aspect and Sentiment Unification Model (ASUM) [204], that slightly differs from JST with respect to the language distribution constraints. While in JST each word may come from a different language model, ASUM constraints the words in a single sentence to come from the same language model.

Among them, ASUM has been chosen as the core model for the proposed approach. This choice is motivated by the fact that (i) the topic-irony model should generate a topic and an ironic/not-ironic orientation for each word, (ii) ASUM is particularly suitable for user-generated content text, where messages have a limited number of characters and a sentence would be either ironic or not ironic with respect to a specific topic, (iii) ASUM makes use of a set of seed words explicitly integrated into the generative process, making the model more stable from a statistical point of view.

Indeed, the proposed **Topic-Irony model (TIM)** is able to model irony toward different topics in a fully unsupervised paradigm, enabling each word in a sentence to be generated from the same *irony-topic* distribution. More formally, let D be the set of documents, D_s the set of sentences, V the vocabulary corresponding to the set of words, Z the set of topics, and I the set of irony classes {ironic, not ironic}.

The generative process is defined as follows:

1. For every pair of (i, z) such that $i \in I$ and $z \in Z$, draw a word distribution $\phi_{iz} \sim \text{Dirichlet}(\beta_i)$.
2. For each document d ,
 - (a) Draw the document's irony distribution $\pi_d \sim \text{Dirichlet}(\gamma)$
 - (b) For each $i \in I$, draw a topic distribution $\theta_{di} \sim \text{Dirichlet}(\alpha)$
 - (c) For each sentence
 - i. Choose an irony class $\hat{i} \sim \text{Multinomial}(\pi_d)$
 - ii. Given \hat{i} , choose a topic $\hat{z} \sim \text{Multinomial}(\theta_{d\hat{i}})$
 - iii. Generate words $w \sim \text{Multinomial}(\phi_{\hat{i}\hat{z}})$

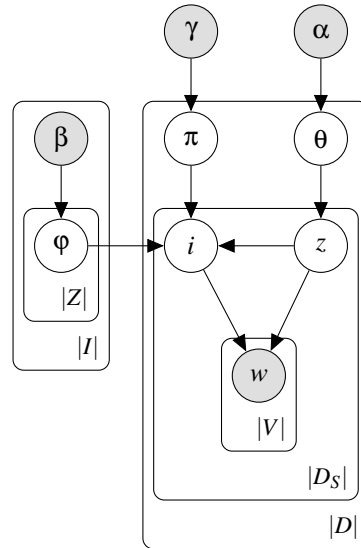


FIGURE 5.13: Graphical representation of the proposed Topic-Irony Model (TIM). Nodes are random variables, edges are dependencies, and plates are replications. Shaded nodes are observable.

Following the study in [204], β_i is the parameter that controls the integration of seed words in the models, in particular its asymmetric form has been adopted. Indeed, the parameter β_i can encode the expectation that words as “*news, bbc, science*” are not usually used in ironic expressions, while “*lol, oh, duh*” are likely to appear in ironic messages. The latent variables θ, π , and ϕ are inferred by Gibbs sampling. The graphical representation of TIM is shown in Figure 5.13.

Word Embeddings (WE) The original ASUM topic model makes use of known general sentiment seed words to derive domain-specific sentiment words [205]. For sentiment seed words, existing sentiment word lexicons can be used (e.g., SentiWordNet [206]) or a new set of words may be obtained by using sentiment propagation techniques [207–210].

For irony detection, a lexicon cannot be a priori defined, but it can be automatically derived in an unsupervised way using a huge quantity of text, as a large amount of unlabeled available user-generated content is easily accessible. To this purpose, **Word Embeddings** can be adopted to derive latent relationships among words (e.g. *irony* is strictly related to *#epicfail*) and therefore to automatically create lexicons based on the language model used in Social Media contexts. This representation is obtained by various training methods inspired by neural network language models (Sec 4.1).

In this study, the ironic-lexicon is derived by leveraging the two model architectures provided by Word2vec [2] presented in Section 4.1.3, i.e. Continuous Bag of Words (CBOW) and Skip-gram. Among the available neural network language models [1, 27], Word2vec has been chosen because of its training efficiency and limited loss of information.

As a practical implication, Skip-gram gives better word representations when the monolingual

data is small. However, CBOW is faster and more suitable for larger datasets. When trained on a large dataset, these models capture a substantial amount of semantic information. Closely related words will have similar vector representations, e.g. *Italy, France, Portugal* will be similar to *Spain*. That is exactly the property that is needed for creating the ironic-driven lexicon.

5.4.1.2 Experimental Settings

Dataset and Evaluation Settings The proposed model has been evaluated on a benchmark dataset for irony detection of user-generated content [107]. The dataset contains 10,000 ironic tweets and 30,000 non-ironic tweets (10,000 for each topic: Education, Humor and Politics). As in [107], the evaluation has been performed as a set of binary classifications, between Irony vs Education, Irony vs Humor and Irony vs Politics in a **balanced** settings (50% ironic text and 50% not ironic text). As a matter of completeness, the task with **imbalanced** classes, i.e. to learn ironic vs others, have been additionally considered. In order to deal with a more realistic and complex scenario, where the term “*irony*” is not explicitly available in the data, the evaluation of the proposed model has been performed according to two experimental conditions:

- Original scenario (O): the dataset has been maintained as it is (where the hashtag symbols have been removed), in order to allow a direct comparison with the state of the art models;
- Simulated scenario (S): the hashtags and the term “*irony*” have been removed from the data in order to simulate a more realistic and complex scenario where the presence of irony is not explicitly pointed out.

Concerning the proposed model, the two hyper-parameters γ and β have been tuned. γ is a prior distribution for the irony distribution in text. Since it is not possible to make assumptions on this distribution, several configurations have been evaluated. The second hyper-parameter β , is the key element for taking advantage of the seed words originated by Word Embeddings. β is the prior of the word-irony-topic distribution defined for ironic seed words, not ironic seed words and all the other words.

The construction of the irony lexicon (to be enclosed as seed words) has been performed by training the *Word Embeddings* model on all the user-generated content in the benchmark dataset. The seed words have been obtained by extracting the most similar words to the term “*irony*”. After a preliminary experimental investigation, the following results report the performance related to the best distributed representation (CBOW).

In the following experimental results, TIM will denote the Topic-Irony Model, while TIM+WE will represent the Topic-Irony Model based on the lexicons induced by Word Embeddings.

The experimental investigation is conducted by comparing TIM, TIM+WE and supervised approaches available in the literature [107, 211–215]). The performance of Precision (P), Recall (R) and F-Measure (F) have been evaluated both in terms of macro-averaged measures (Eqs. 5.9, 5.11, 5.13) and by distinguishing the individual classes of ironic (+) and not ironic (-) (Eqs. 5.4, 5.5, 5.6). Moreover, the results have been also reported in terms of Accuracy (Eq. 5.3).

5.4.1.3 Experimental Results

Balanced Dataset

Original scenario (O) The results of the proposed model are compared to the method introduced in [107] in Table 5.29. TIM clearly outperforms the supervised method of Reyes et al. with significant improvements, i.e. (on average) 11% for Precision, 14% for Recall and 13% for F-Measure.

TABLE 5.29: Results compared to a supervised state of the art method for each binary problem (O).

		P	R	F
irony vs education	Reyes et al. [107]	0.7600	0.6600	0.7000
	TIM	0.8225	0.8746	0.8477
	TIM + WE	0.8228	0.8629	0.8423
irony vs politics	Reyes et al. [107]	0.7500	0.7100	0.7300
	TIM	0.9127	0.8560	0.8834
	TIM + WE	0.9131	0.8373	0.8735
irony vs humor	Reyes et al. [107]	0.7800	0.7400	0.7600
	TIM	0.8414	0.8174	0.8292
	TIM + WE	0.8142	0.7832	0.7983

A further remark relates to the Precision and Recall measures obtained by TIM and TIM+WE. It can be easily noted, from Table 5.29, that the two proposed models achieve homogeneous performance in all the binary classification problems, obtaining Precision and Recall performance of similar magnitude.

In order to grasp more peculiar behaviors, the performance measures both for ironic (+) and not ironic (-) classes have been reported in Table 5.30. It is again possible to observe that Precision and Recall measures are well balanced for both classes, ensuring good performance also on the most difficult target (ironic). Concerning Accuracy, TIM and TIM+WE are able not only to outperform a trivial classifier that would ensure 50% of Accuracy, but they also perform differently according to the binary problem that they address. As expected, tackling Irony vs Humor is more difficult than Irony vs Politics and Irony vs Education. In fact, as stated by the authors that made the dataset available [107], the similarity estimated between pairs of classes is significantly higher in Irony vs Humor in the other binary problems.

TABLE 5.30: Results of the proposed models (TIM and TIM+WE) for each binary problem (O) distinguishing between ironic (+) and not ironic (-).

		P (+)	R (+)	F (+)	P (-)	R (-)	F (-)	Accuracy
irony vs education	TIM	0.8225	0.8746	0.8477	0.8664	0.8116	0.8380	0.8430
	TIM + WE	0.8228	0.8629	0.8423	0.8566	0.8146	0.8350	0.8388
irony vs politics	TIM	0.9127	0.8560	0.8834	0.8644	0.9183	0.8905	0.8871
	TIM + WE	0.9131	0.8373	0.8735	0.8498	0.9204	0.8836	0.8788
irony vs humor	TIM	0.8414	0.8174	0.8292	0.8227	0.8461	0.8342	0.8318
	TIM + WE	0.8142	0.7832	0.7983	0.7911	0.8214	0.8059	0.8022

Regarding the ironic-lexicon derived through Word Embeddings, the presented results show that it does not generally improve the performance of TIM. This is probably due to the impact that the word “*irony*” has into the dataset and into the model: the lexicon of TIM only composed of the “*irony*” term is sufficient to discriminate between the ironic and non-ironic orientations. Although the additional seed words enclosed in TIM+WE allow the model to obtain remarkable results with respect to the supervised settings and similar performance compared to TIM, the only presence of the term “*irony*” guarantees better performance than richer lexicons. As expected, TIM better fits the original scenario where the ironic statements available into the dataset are strongly characterized by the “*irony*” term. In order to evaluate the generalization abilities of the proposed models in a real and more complex scenario, where the term “*irony*” is not explicitly available into the ironic statements, the model performance has been also evaluated in a simulated scenario presented in the next section.

Additional results are reported in order to compare the proposed approaches with respect to other *supervised* related works on the same dataset. In particular, the benchmark corpus exploited for the training and inference process of TIM and TIM+WE has been previously adopted also in [212–215] (only in balanced settings for the original scenario). The results obtained by the proposed model together with the ones by supervised state of the art models are detailed in Table 5.31.

TABLE 5.31: Results in terms of F-Measure of the proposed models (TIM and TIM+WE) against state of the art approaches.

	Irony vs.		
	Education	Humor	Politics
Reyes et al. [107]	0.70	0.76	0.73
Barbieri and Saggion [212]	0.73	0.75	0.75
Hernández-Farías et al., Farias and Irazu [213, 214] ¹	0.78	0.75	0.79
Hernández-Farías et al., Farias and Irazu [213, 214] ²	0.78	0.79	0.79
TIM	0.85	0.83	0.88
TIM+WE	0.84	0.80	0.87
Farías et al. [215]	0.90	0.90	0.92

TABLE 5.32: Results of the proposed models (TIM and TIM+WE) for each binary problem (S) distinguishing between ironic (+) and not ironic (-).

		P (+)	R (+)	F (+)	P (-)	R (-)	F (-)	Accuracy
irony vs education	TIM	0.7996	0.7934	0.7964	0.7958	0.8022	0.7989	0.7977
	TIM + WE	0.8050	0.8103	0.8075	0.8098	0.8046	0.8070	0.8073
irony vs politics	TIM	0.8719	0.8358	0.8534	0.8426	0.8775	0.8596	0.8567
	TIM + WE	0.8780	0.8420	0.8596	0.8485	0.8833	0.8655	0.8627
irony vs humor	TIM	0.7356	0.7675	0.7510	0.7574	0.7247	0.7405	0.7460
	TIM + WE	0.7205	0.8392	0.7752	0.8079	0.6752	0.7354	0.7570

This final comparison clearly highlights the improvements that the proposed models are able to provide. Not only TIM and TIM+WE perform significantly better than most of state of the art models, but it is even more remarkable that they perform better even though their completely unsupervised nature. The peculiarity of the supervised approach that obtain the best performance [215] is the exploitation of the emotional information, which has been considered by computing the frequency of words in the text belonging to an emotional category according different resources (e.g. EMOLEX [216]). Although the good performance of the model proposed in [215], it still remains a supervised approach.

The following paragraph reports the computational results on the simulated scenario, where the ironic figurative language is not explicitly marked in the dataset, but embedded into the sentences. In Table 5.32 the results in terms of Precision, Recall and F-measure are reported distinguishing between ironic (+) and not-ironic (-) classes, together with the global Accuracy measure.

As expected, the recognition performance of TIM and TIM+WE decreases, compared to the original scenario (see Table 5.30), since the term “*irony*” is removed from the corpus. Moreover, in this simulated case, where the irony is not explicitly pointed out, a lexicon able to boost TIM, and therefore the recognition performance of ironic messages, becomes beneficial. By analyzing all the performance measures, it is evident that the introduction of WE derived-lexicon allows the probabilistic model TIM+WE to achieve better results than simple TIM. Also in these experimental settings, it is possible to remark that the two proposed models are able to obtain similar Precision and Recall performance, highlighting robust performance in this complex scenario.

Imbalanced Dataset

Original scenario (O) In order to compare the proposed model to the state of the art approaches on irony detection, Figure 5.14 shows the results obtained by TIM, TIM+WE and two

¹In these experiments, the authors used the Lesk similarity measure.

²In these experiments, the authors used the Wu-Palmer similarity measure.

supervised approaches, i.e. the feature engineering based approach presented by Reyes et al. [107] and the ensemble model introduced by Fersini et al. [211].

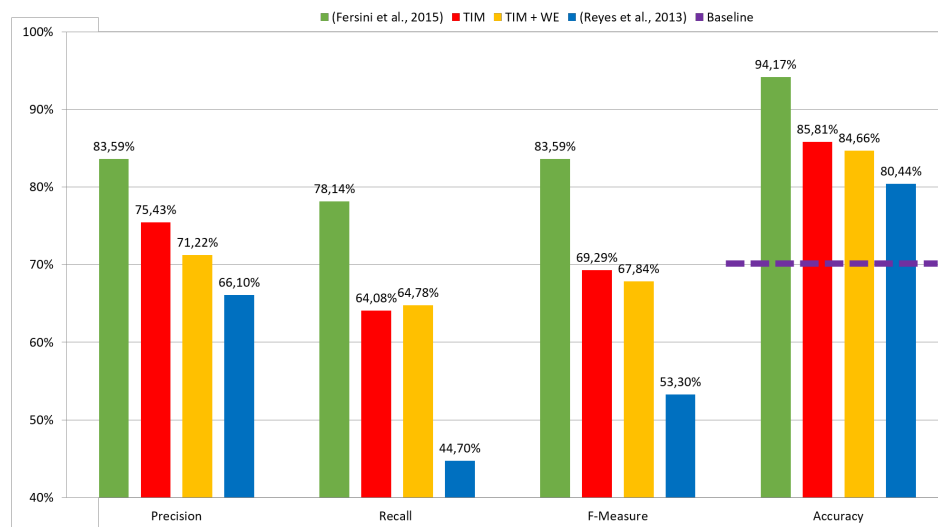


FIGURE 5.14: Comparison of TIM and TIM+WE with supervised state of the art methods on the imbalanced dataset.

First of all, it is important to mention that TIM and TIM+WE are able to perform better than a trivial classifier that would ensure 70% of Accuracy. Furthermore, it can be pointed out that the proposed unsupervised models achieve remarkable results compared to the supervised ones. In particular, both TIM and TIM+WE are able to obtain higher recognition performance than the supervised irony model introduced in [107]. Regarding the comparison with the supervised ensemble method presented in [211], TIM and TIM+WE achieve similar performance.

In summary, considering that the proposed method is fully unsupervised, it can be considered very promising. The evaluation of the framework has been extensively studied in Table 5.33. Similar to the balanced experimental settings on the original scenario, the contribution of WE does not generally improve the performance of TIM, still remaining comparable. Again, the better performance obtained by TIM with respect to TIM+WE is related to the dataset composition, where more than 36% of the ironic textual messages contain the word “*irony*”.

Simulated scenario (S) This section reports the computational results on the simulated scenario, where the irony figure of speech is embedded in the sentences with no reference to the term “*irony*”. Table 5.34 shows the behavior of both the proposed models. Similar to the previous balanced case study, the recognition performance of TIM and TIM+WE decreases, compared to the original scenario (see Table 5.33), once the term “*irony*” is removed from the corpus. The contribution of WE becomes evident when dealing with the simulated scenario. In a more complex and real environment, where the ratio between ironic and not ironic messages is low and the ironic orientation in a sentence can be derived only by the surrounding context, TIM+WE is

TABLE 5.33: Results of the proposed models (TIM and TIM+WE) on the imbalanced dataset (O) distinguishing between ironic (+) and not ironic (-).

	P (+)	R (+)	F (+)	P (-)	R (-)	F (-)	Accuracy
TIM	0.7543	0.6408	0.6929	0.8861	0.9305	0.9078	0.8581
TIM + WE	0.7122	0.6478	0.6784	0.8862	0.9128	0.8993	0.8466

TABLE 5.34: Results of the proposed models (TIM and TIM+WE) on the imbalanced dataset (S) distinguishing between ironic (+) and not ironic (-).

	P (+)	R (+)	F (+)	P (-)	R (-)	F (-)	Accuracy
TIM	0.4406	0.5902	0.5044	0.8464	0.7507	0.7957	0.7107
TIM + WE	0.5320	0.4958	0.5132	0.8361	0.8550	0.8455	0.7654

able to provide a valuable contribution to bridge the semantic gap. By investigating in details Precision and Recall measures of both models, it is possible to derive two main observations:

- TIM and TIM+WE, even if induced in the worst scenario where the dataset is imbalanced and lacks of explicit references to irony, are able to perform better than a trivial classifier that would ensure 70% of Accuracy. This makes the proposed models particularly suitable for real-world applications, in particular on noisy user-generated data.
- TIM+WE obtains Precision and Recall of the same magnitude both for the ironic class (0.5320 for P(+) and 0.4958 for R(+)) and not ironic class (0.8361 for P(-) and 0.8550 for R (-)), compared to TIM which obtains a poor trade-off between the two performance measures (0.4406 for P(+) and 0.5902 for R(+), and 0.8464 for P(-) and 0.7507 for R (-)). This suggests that TIM+WE has good predictive performance characterized by well-proportioned abilities both in terms of Precision and Recall on both ironic and not-ironic orientations.

5.4.1.4 Topic Detection results

In order to perform a qualitative analysis of the obtained results, this section reports some examples of discovered ironic and not ironic topics. Tables 5.35 and 5.36 show the words that both topic-irony models, TIM and TIM+WE, have associated to each topic distinguishing between ironic (+) and not ironic (-). In particular, each column corresponds to the words which have been inferred to be most probable one with respect to a given topic and ironic orientation. For a more effective visualization, the words that have been empirically evaluated as topic-related are highlighted in bold, while the ones that have been empirically considered as irony-related have been underlined.

As general remark, the experimental evaluation suggests that the proposed Topic-Irony models may not only help the irony classification task, but they can also improve the ability to identify

TABLE 5.35: Topic-related words are reported in **bold**, while the irony-related ones are underlined. These results are related to TIM in the original scenario (O) and the balanced settings distinguishing between ironic (+) and not ironic (-) for each topic.

humor(-)	humor(+)	politics(-)	politics(+)	education(-)	education(+)
funny	unions	tcot	<u>irony</u>	technology	<u>irony</u>
posemoticon	workers	politics	<u>oh</u>	education	<u>linux</u>
shoy	benefit	obama	<u>get</u>	new	org
award	<u>always</u>	news	<u>lol</u>	apple	<u>microsoft</u>
nominate	cd	p	<u>like</u>	google	open
lol	movies	gop	<u>u</u>	school	<u>tsunami</u>
humor	labor	tlot	<u>people</u>	news	attack
jokes	<u>porn</u>	teapay	day	ipad	<u>creates</u>
joke	fox	us	one	posemoticon	sponsors
q	tv	palin	<u>love</u>	twitter	<u>openmainframe</u>
comedy	<u>news</u>	iran	common	via	<u>gnu</u>
quote	playboy	pay	got	ac	<u>religion</u>
like	weed	sgp	time	iphone	ban
get	marijuana	iranelection	posemoticon	edtech	thought
one	cannabis	hcr	see	web	<u>dilemma</u>

the underlying topics of user-generated content. Indeed, the specific topics of the benchmark dataset can be well distinguished by looking at the most relevant keywords identified by the proposed approaches, still maintaining a good characterization of ironic and not ironic orientations. For instance, the sentence “@user Deeper irony would be Sarah Palin campaigning for literacy” is correctly classified as ironic and properly related to the topic Politics.

A further remark concerns TIM+WE and, in particular, to its ability to deal with short and noisy natural language text. The fact that user-generated content is composed of a limited number of words poses considerable problems when applying traditional probabilistic topic models. These models typically suffer from data sparsity to estimate robust word co-occurrence statistics when dealing with short and ill-formed text. The proposed model is able to reduce the negative impact of short and noisy text in real and complex scenarios thanks its ability to take advantage of distributed representation derived through Word Embeddings. In conclusion, TIM+WE shows promising results on detecting irony in user-generated content, where Word Embeddings models help on overcoming the issues related to the short and noisy nature of the input text. Moreover, the proposed model resulted as particularly suitable for dealing with those topic-related ironic sentences where the ironic orientation is not explicitly available. An instance of its ability can be grasped by the following sentence “catching up on news... see that Pres. Obama’s aunt is in the news again, and that she said she loves Pres. Bush.”, where the model correctly classifies the statement as Politics and recognizes as ironic (even if the ironic orientation is not explicitly marked in the text).

TABLE 5.36: Topic-related words are reported in **bold**, while the irony-related ones are underlined. These results are related to TIM+WE in the simulated scenario (S) and the balanced settings distinguishing between ironic (+) and not ironic (-) for each topic.

humor(-)	humor(+)	politics (-)	politics(+)	education (-)	education (+)
funny	quote	tcot	<u>oh</u>	technology	common
posemoticon	popular	obama	<u>u</u>	education	postrank
shoy	<u>love</u>	politics	<u>lol</u>	new	education
award	<u>palin</u>	news	<u>get</u>	apple	health
nominate	blind	p	<u>like</u>	google	nowplaying
lol	<u>lingerie</u>	gop	<u>day</u>	news	make
humor	vote	tlot	posemoticon	school	<u>lol</u>
jokes	quickpolls	us	one	twitter	flaker
joke	anonymous	teapay	<u>people</u>	ipad	cholesterol
q	com	pay	got	via	video
comedy	voteglobal	iran	common	posemoticon	man
quote	gotpolitics	sgp	<u>love</u>	ac	difference
like	politics	hcr	yet	iphone	causes
one	friends	iranelection	politics	one	sense
get	<u>barbie</u>	health	time	edtech	fiction

5.4.1.5 Related Works

As defined in [217], a figure of speech is any artful deviation from the ordinary mode of speaking or writing. Among the tasks dealing with the problematic figures of speech in Natural Language Processing, this work studies irony detection on user-generated content. Sulis et al. [218] distinguish two different kinds of irony: *situational* and *verbal* [219–221]. Situational irony refers to the state of events which is the reverse of what has been expected, while the term verbal irony refers to a figure of speech characterized by the possibility of distinguishing between a literal and an intended/implied meaning. Verbal irony is often viewed as a synonym of *sarcasm* and investigated as the same problem [222–224]. Nevertheless, few preliminary studies addressed the task to further deepen the differences between irony and sarcasm [218, 225, 226]. Additionally, a rarely investigated form of irony used in Social Media is *self-mockery* [227, 228]. Differently from other forms of irony, self-mockery does not involve contempt for others, but the speaker wishes to dissociate from the content of the utterance.

This thesis focuses on *verbal* irony, which is commonly used to convey implicit criticism with a particular victim as its target, saying or writing the opposite of what the author means [229]. As mentioned in [230], the language should not be taken literally, especially when addressing a Sentiment Analysis task. The presence of strongly positive (or negative) words that are used ironically, which means that the opposite polarity was intended, can easily mislead Sentiment Analysis classification models [231].

In the last year, several approaches for irony detection based on different set of features have been investigated. In [232], the authors proposed a semi-supervised technique to detect sarcasm in Amazon product reviews and tweets. They used pattern-based (high-frequency words and

content words) and punctuation-based features to build a sarcasm detection model. A supervised approach has been proposed in [233], where the irony detection problem is studied for sentiment analysis in Twitter data. The authors used unigrams, word categories, interjections (e.g., “ah”, “yeah”), and punctuation as features. Emoticons and ToUser (which marks if a tweet is a reply to another tweet) were also used. In [234], the authors considered a specific type of sarcasm where sarcastic tweets include a positive sentiment (such as “love” or “enjoy”) followed by an expression that describes an undesirable activity or state (e.g., “taking exams” or “being ignored”). In [107] the authors focused on developing classifiers to detect verbal irony based on a set of high-level features: ambiguity, polarity unexpectedness and emotional cues. In [235] a supervised model has been exploited for document-level irony detection in Czech and English by using n-grams, patterns, POS tags, emoticons, punctuation and word case. Additionally, in [213, 214] the authors studied structural and sentiment analysis features, while in [215] affective features have been also considered. In [212], the relevance of linguistic features have been evaluated to derive the most representative ones for irony detection. In [211] the authors proposed an ensemble approach, based on a Bayesian Model Averaging paradigm, which makes use of models trained using several linguistic features, such as pragmatic particles and Part-Of-Speech tags. In [213], the irony detection problem has been addressed by investigating statistical-based and lexicon-based features paired with two semantic similarity measures, i.e. Lesk and Wu-Palmer [236].

Other recent works [237, 238] aim to address the sarcasm detection in microblogs by including extra-linguistic information from the context such as properties of the author, the audience, the immediate communicative environment and the user’s past messages. Word Embeddings have been used as features in a supervised approach in [239], where the authors expressed the sarcasm detection task as a Word Sense Disambiguation problem.

Although the above-mentioned studies represent a fundamental step towards the definition of effective irony detection systems, they suffer from three main limitations:

- they assume a labeled corpus for training supervised and semi-supervised models;
- they are tailored for domain-dependent irony detection, restraining their applicability to other domains of interest;
- they disregard the topic subjected to the irony.

In order to overcome these limitations, the proposed work has investigated an unsupervised topic-irony model enriched with domain-independent Word Embeddings.

Chapter 6

Enhancing Textual Feature Representation including Relational Information

The previous chapters have highlighted the importance of user-generated content as a valuable textual source of information for many domains and applications.

Although a portion of user-generated content is created by individuals (*producers*), the great majority is developed through the collaborative efforts of multiple users (*participants*) [240]. An excellent example is Wikipedia, where contributors work together on articles; another case regards scientific publications where scientific collaborations and comparisons are the success keys for innovating; common people collaborate every day by posting on Social Media their opinions and sentiment about several subjects, such as products, events or political parties.

From these observations, it is possible to conclude that user-generated content encloses relevant insights that can be extracted from their textual content and further enhanced by taking into account the interactions that involve them. Let us consider, for example, the task of Sentiment Analysis in a Social Media environment, where opinions are usually collected by gathering the textual content from a large number of users. Do not taking into account the relational structure underlying the users could lead to misleading conclusions. For instance, collecting an isolated reply post within a discussion that state “*I agree with you!*” does not provide any insight on the user’s opinion about the considered topic, because it is unknown to which post is related to. In fact, “*I agree with you!*” could be equally probable related either to a positive or negative original opinion. However, jointly considering the textual content and the existing relationships, can significantly help on the post interpretation.

The most suitable structure for representing both textual and relational information is an **attributed graph**, where the subjects (*nodes*) involved are related to each other through relationships (*edges*) and a set of attributes (i.e. text) is associated to each node.

Dealing with attributed graph is very complex and computationally expensive because of different characteristics of the data, i.e. *size of the graph*, *dynamic nature*, *noise* and *heterogeneity* [241]. Efficient approaches for handling such structure are based on Representation Learning, and in particular on the effective Deep Learning models. However, most of the state of the art works are aimed at efficiently learning good representations for either the textual or relational information.

In the literature, as already detailed in Chapter 4, the most effective vector representation of natural language text can be obtained by several Deep Learning models, which can be roughly distinguished in neural networks language models [1, 2, 27, 28] and Deep Neural Networks in the form of Auto-encoders [35, 37, 96–98, 100].

The embedded representation of the relational structure associated with potentially large and complex graphs has been investigated by the so-called graph embedding models. These models map each graph node to a low-dimensional dense vector representation, encoding meaningful information conveyed by the graph. Indeed, Deep Learning is again the most dominant approach for obtaining graph embeddings.

This Chapter presents a novel Representation Learning model based on Deep Learning architectures. The proposed model, called Constrained deep Attributed Graph Embeddings model (CAGE), will provide a representation able to convey textual attribute content enhanced by the relational structure of the graph. Considering both textual and relational information for obtaining a joint vector representation of user-generated content is expected to improve the results in different Natural Language Processing tasks.

The proposed model will face the following challenges:

- (1) Textual attribute expression: the obtained feature representation will be able to directly encode the textual attribute content and to provide higher expressiveness with respect to traditional models for representing text.
- (2) Relational structure preservation: the obtained feature representation will preserve the structure of the graph, that is often complex and highly nonlinear, by facing the common problem of real-world graphs, i.e. scalability and sparsity.
- (3) Dimensionality of the representation: the dimension of the embedded representation should be chosen as a trade-off between reconstruction precision performance and time/space complexity.

6.1 Related Works

Representation Learning models for attributed graphs comprising textual attributes are mostly investigated by separately considering relational information and textual attributes.

Regarding textual content, Chapter 4 provides an overview of the Deep Learning architectures for providing meaningful representations of natural language text.

Representation Learning models of relational structures have attracted a surge of research attention over the past decade, focusing in particular on developing new embedding algorithms. In this domain, the research studies can be roughly distinguished in Factorization based methods and Deep Learning approaches [242].

The basic idea underlying both methods is to preserve both the local and global graph structures in the derived embedded vector space. Naturally, the local structures are represented by the observed relations in the graphs, which capture the first-order proximity between pairs of nodes. The global structure is captured by the second-order proximity, which considers the shared neighborhood of pairs of nodes. This means that the more neighbors are shared by two nodes, the higher the similarity between them should be.

Factorization based methods represent the connections between nodes in the form of a matrix and factorize this matrix to obtain the embeddings. Although Laplacian Eigenmaps [243] and Locally Linear Embedding [244] are the fundamental approaches in this area, they are not able to scale for real-world graphs. For this reason, Ahmed et al. [245] introduced Graph Factorization to find a low-dimensional embeddings for large graphs. In order to solve the complexity of the computational process of matrix factorization, the framework has been distributed so that the data are partitioned over a set of machines. However, this factorization approach does not preserve the global graph structure. Hence, their work was successively extended for including both first- and second-order proximities [246, 247].

As anticipated in the previous section, the growing interest in Deep Learning algorithms has affected also the task of graph Representation Learning due to their ability to model nonlinear structures in the data.

Among the most relevant related works, deep representations of random walks have been investigated. These methods are efficient for dealing with large graphs because they can be easily parallelized and they are particularly suitable to accommodate small changes in the graph structure without the need for global recomputation of the models. The most promising approaches known as DeepWalk [248] and node2vec [249] take advantage of neural network language modeling. In particular, these methods are aimed at preserving a higher-order proximity between nodes by learning latent representations of random walks, treating them as the equivalent of sentences.

Beyond random walks methods, most of the investigations are focused on improving Deep Learning architectures. Structural Deep Network Embeddings (SDNE) [250] exploits the ability of deep neural networks to generate an embedded representation that can capture nonlinearity in graphs and simultaneously preserve the first- and second-order graph proximities. This unsupervised model is based on an architecture characterized by two main components: the first one is a deep Auto-encoder that estimates the embeddings vectors by reconstructing the nodes' neighborhood, while the second one is based on Laplacian Eigenmaps [243] and aims to make the embedded representation of connected nodes more similar.

Only few studies consider the set of attributes associated with each node in addition to the graph topological structure (*attributed graph*). In [241], the authors proposed a supervised model for transferring different nodes in heterogeneous graphs to unified vector representations. While this approach is scalable and robust, it only considers the first-order proximity and it has a specific focus on image-text relationships. An approach able to generalize independently from the input data type was proposed by Huang et al. [251], where graph embeddings are computed based on the decomposition of an attribute similarity matrix and the embeddings difference between connected nodes. They also developed a distributed optimization algorithm to decompose the original problem into many sub-problems with lower complexity, which can be solved by sub-workers in parallel.

Although the above-mentioned approaches represent a fundamental step and obtained impressive results in terms of efficiency and efficacy performance, they do not explicitly consider the attributes of the nodes but only a measure of attribute similarity. Hence, the computed vector representations encode the graph structure and reflect the attribute similarities between nodes, but it is not a direct expression of their attributes.

In 2016, Pan et al. [252] proposed a supervised model called TriDNR, which makes use of coupled neural network architectures to exploit network information from three parties: node structure, node attributes, and node labels (if available). In this model, node structure and attributes are leveraged by two state of the art approaches, i.e. DeepWalk [248] and Paragraph Vector [253] respectively. Despite the impressive improvements with respect to the other approaches in the literature, it is possible to identify two main disadvantages of using TriDNR. First, it adopts DeepWalk model for obtaining an embedded representation of the node structure. Although this model has been proved to be empirically effective, DeepWalk does not well articulate a clear objective function to preserve the graph structure and it is also prone to preserve only the second-order proximity. Secondly, TriDNR is a supervised model, which implies that the data must be manually labeled.

In order to overcome the above-mentioned limitations, an unsupervised model for learning a feature representation from large attributed graphs has been presented in this thesis. In particular, the proposed model is aimed at creating high-level representations of textual attributes

enhanced by the relational information conveyed by the graph. In order to accomplish this task, promising Deep Learning architectures, able to efficiently and effectively produce dense vector representation from node attributes and structure, are considered.

6.2 Deep Attributed Graph Embeddings Model

In this section, several needed definitions are given for providing an overall description of the problem.

6.2.1 Problem Definition and Motivation

As first concern, the formal definition of an attributed graph is provided:

Definition 6.2.1. An **attributed graph** is defined as $G = (\mathbf{V}, \mathbf{E}, \mathbf{\Pi})$, where $\mathbf{V} = \{v_1, \dots, v_n\}$ is the set of nodes, $\mathbf{E} = \{e_{ij}\}_{i,j=1}^n$ is the set of edges between the nodes and $\mathbf{\Pi} = \{\pi_{v_1}, \dots, \pi_{v_n}\}$ represents the *node attributes* related to each node v_i . Each edge e_{ij} is associated with a weight $s_{ij} > 0$, which indicates the strength of the relation.

A graphical representation of an attributed graph is reported in Figure 6.1

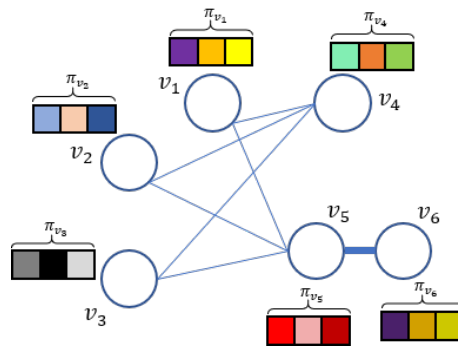


FIGURE 6.1: Graphical example of an attributed graph. Attribute distribution is exemplified by the color representation.

In order to encode the textual attribute information and the graph structure (local and global), the proposed model should preserve several proximity measures between pairs of nodes that are presented in the following section. Hence, the concept of node textual content proximity can be defined as:

Definition 6.2.2. The **textual attribute proximity** between attributes of a pair of nodes (π_{v_i}, π_{v_j}) in a graph is the similarity between their textual attribute contents.

As previously mentioned, considering the node textual attributes permits to model the most essential component in user-generated content for their interpretation and understanding, i.e. the natural language text. The example in Figure 6.2(a) describes the case where there exists a latent relation between two nodes (v_1 and v_6), visible from the similarity between the attributes expressed as similar colors, but an edge between them does not exist. Intuitively, these two nodes should have a similar representation because of their high attribute proximity. As a real-world application example, in a citation network, papers with similar contents would have a similar representation, while papers with different contents will tend to have a distant vectors. Considering only the relational information can lead to imprecise conclusions. For example, a citation between two papers does not always imply that they share the same topics, because they could be related for marginal aspects (e.g. a paper on Irony Detection that cites another paper on Machine Learning). An analogous scenario can be observed in Sentiment Analysis on Social Networks, where it is commonly assumed that the relational information, in the form of friendships or follower relationships, unconditionally represent the sentiment agreement between two connected users. This assumption does not hold in a real-world scenario, where two structurally connected users (e.g. friends) may have divergent opinions on a given topic [254]. This issue can be overcome by considering the textual content provided by the users.

Although the natural language text is of a great importance when dealing with user-generated content, it is not sufficient for representing the whole underlying picture. In fact, especially in Social Media, contents are usually associated to nodes that are related to each others. Taking into consideration this relational information can further help to capture similarities or dissimilarities between nodes enhancing the information obtained by the textual content. For providing a complete representation of the structural component of the graph, both local and global structures are essential to be preserved. Then, the *local* structure can be captured by the *first-order* proximity [246]:

Definition 6.2.3. The **first-order proximity** in a graph is the local pairwise proximity between two nodes. For each pair of nodes linked by an edge $e_{ij} = (v_i, v_j)$, the weight on that edge s_{ij} indicates the first-order proximity between v_i and v_j . If no edge is observed between v_i and v_j , their first-order proximity is 0.

Intuitively, it is necessary for graph embeddings to preserve the first-order proximity because it implies that the representation of two connected nodes should be similar. As shown in Figure 6.2(b), the stronger is the connection between two nodes (v_5 and v_6), the higher will be the first-order proximity between them. The same reasoning is expected to be appropriate also for real-world cases. For instance, two authors that co-authored a paper should share some topics of interest. Consequently, the more papers they have co-authored, the more the authors will have similar topic interests. In Social Networks, two users connected by an approval “like”

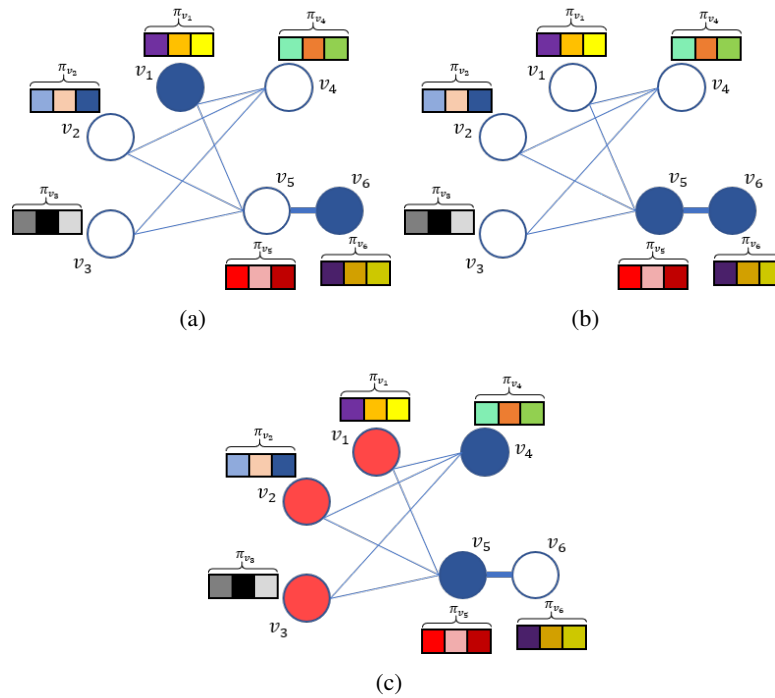


FIGURE 6.2: Selection of nodes used for explaining textual attribute proximity (on the top left), first-order proximity (on the top right) and second-order proximity (on the bottom).

interaction are expected to agree on a given topic, implying that the more approval interactions exist between the users, the more it is likely that they share the same opinions.

In a real-world scenario, graphs are often sparse, meaning that the greater part of the nodes has a degree considerably lower than the maximal possible one. For this reason, considering only the proximity of connected nodes is not enough in order to capture the structure of the graph. The following definition introduces the second-order proximity, which characterizes the *global* graph structure [246].

Definition 6.2.4. The **second-order proximity** between a pair of nodes (v_i, v_j) in a graph is the similarity between their neighborhood graph structures. Formally, let $\mathcal{N}_i = (w_{i1}, \dots, w_{i|V|})$ denotes the first-order proximity of v_i with all the other nodes, then the second-order proximity between v_i and v_j is determined by the similarity between \mathcal{N}_i and \mathcal{N}_j . If no nodes are linked from/to both v_i and v_j , the second-order proximity between v_i and v_j is 0.

The assumption behind the second-order proximity is that nodes are similar if they have common neighbors. Figure 6.2(c) depicts the case when second-order proximity can help to handle edge sparsity. Although nodes v_4 and v_5 are not connected, their embedded representation should be similar. This is due to the fact that they share many common neighbors (v_1 , v_2 and v_3) and therefore they are characterized by a high second-order proximity. Such assumption has been proved reasonable in many fields [255, 256]. Following the previous example, if two papers

cite several common papers, they are expected to discuss the same topics. Similarly, two users are likely to be friends if they have many common friends. Hence, exploiting this proximity measure makes possible to deal with the sparsity problem by coupling nodes that are not directly connected.

The proposed model considers simultaneously all the presented proximities in order to address the following problem:

Definition 6.2.5. Given an attributed graph denoted as $G = (\mathbf{V}, \mathbf{E}, \mathbf{\Pi})$, **attributed graph embeddings** aims at learning a mapping function $C_G : \mathbf{V} \rightarrow \mathbb{R}^m$, where $m \ll |\mathbf{V}|$. The objective of this function is to explicitly preserve first-order, second-order and textual attribute proximities when computing the embedded representation of nodes.

6.2.2 Constrained Deep Attributed Graph Embeddings Model

This section introduces a novel embeddings model named **Constrained deep Attributed Graph Embeddings** (CAGE). The main underlying idea of the proposed model is to learn, in unsupervised settings, the textual feature representation of user-generated content by exploiting Deep Learning methods on large attributed graphs.

In particular, two different Deep Learning models, i.e. SDNE [250] and KATE [100], have been taken into consideration for deriving an embedding representation that explicitly considers both textual attributes and graph structure. The two state of the art models have been preferred among the others because (i) they are the most promising models for inferring node structure and textual embeddings respectively and (ii) they both rely on the same Deep Learning architecture, that is an Auto-encoder architecture. An overview of CAGE is reported in Figure 6.3.

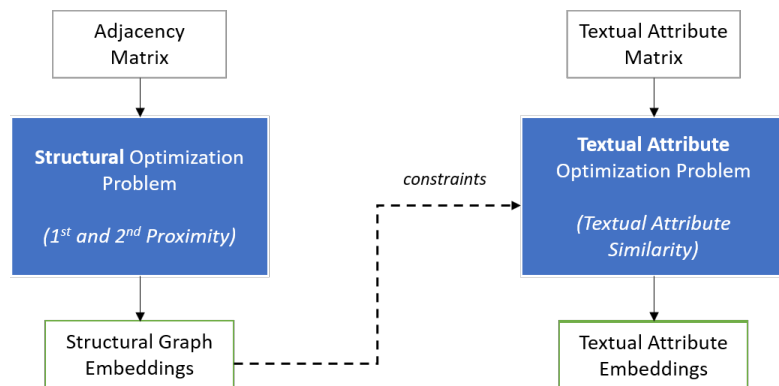


FIGURE 6.3: Graphical representation of the Constrained deep Attributed Graph Embedding model (CAGE).

Once the first deep Auto-encoder SDNE has obtained the **structural graph embeddings**, i.e. the embedded representation of the node structure, the learning process of the second deep

Auto-encoder KATE will be constrained to let the **attribute node embeddings**, i.e. the embedded representation of the textual node attributes, to be coherent with the structural graph embeddings. According to this constrained model, the representation of two nodes, that are structurally different in terms of first- and second-order proximity, but similar from the attribute point of view, will converge. Analogously, the representation of two nodes with similar graph structure, but with dissimilar attributes, will diverge. This model leads to solve two different optimization problems related to the training phase.

Following, first the structural and attribute embeddings model are introduced, then the definition and derivation of the proposed model is provided.

6.2.2.1 Auto-encoder

Auto-encoders are a specific Deep Learning architecture that is trained to attempt to copy its input to its output [32]. As introduced in Section 4.2, these unsupervised models are particularly able to efficiently extract latent factors of variations of the input data.

Formally, following the notations of Chapter 3, the construction of the hidden layers can be formalized as:

$$\begin{aligned} \mathbf{h}_0 &= a_0(\mathbf{W}\mathbf{x} + b_0), \\ \mathbf{h}_l &= a_l(\mathbf{W}\mathbf{h}_{l-1} + b_l), \quad l = 2, \dots, u \end{aligned} \tag{6.1}$$

where the subscript l corresponds to the sequence position of the layer in the neural network.

6.2.2.2 Textual Attribute Embedding Model

The model used for inferring the textual node attribute vector representation is KATE [100], that stands for K-Competitive Auto-encoder for Text. As already presented in Section 4.2.5, this approach has been specifically proposed to deal with natural language text, suggesting a novel technique to address the sparsity and the Zipf's law distribution issues. The main advancement consists in the *competition* hidden layer, where neurons competes with each other. Therefore, it is expected that each neuron becomes specialized in recognizing specific data patterns, learning meaningful representations of textual data. Moreover, KATE is particularly suitable for dealing with short and noisy text that typically characterize user-generated content.

Given the textual attribute matrix \mathbf{X}^T , where $x_i^T = \pi_{v_i}$, the objective of the model is to minimize the reconstruction error:

$$L_x^T(\theta^T) = \sum_{i=1}^n \|x_i^T - \hat{x}_i^T\|_2^2 \tag{6.2}$$

6.2.2.3 Structural Graph Embedding Model

The key element for obtaining the structural graph embeddings is the Structural Deep Network Embedding model (SDNE) proposed in [250]. This model is based on an Auto-encoder architecture able to incorporate both first- and second-order proximity into the embedded representation of each node of the graph.

Wang et al. claim that the second-order proximity can be naturally incorporated into the reconstruction process of the Auto-encoder.

Given the adjacency matrix X^S , where $x_i^S = s_i$, the Auto-encoder will guarantee that nodes with similar neighborhood structures will have similar latent representations. The loss function concerned with the second-order proximity is the following:

$$\begin{aligned} L_x^S &= \sum_{i=1}^n \|(\hat{x}_i^S - x_i^S) \odot \xi_i\|_2^2 \\ &= \|(\hat{\mathbf{X}}^S - \mathbf{X}^S) \odot \boldsymbol{\xi}\|_F^2 \end{aligned} \quad (6.3)$$

where \odot denotes the Hadamard product, i.e. entrywise multiplication, and $\xi_i = \{\xi_{i,j}\}_{j=1}^n$. If $s_{i,j} = 0$ then $\xi_{i,j} = 1$, else $\xi_{i,j} = \alpha_\xi > 1$.

Recalling the architecture of an Auto-encoder presented in Section 4.2, X^S is the input to the encoder function and \hat{X}^S is its reconstruction given by the decoder function. The objective is to minimize the reconstruction error, i.e. the difference between X^S and \hat{X} .

The first-order proximity can be fulfilled by learning a model that makes the similarity between the latent representations of pair of nodes representative of their connection strength. The higher is the weight s_{ij} between a pair of nodes (v_i, v_j) , the higher will be the similarity between their embedded representation h_i, h_j .

The first-order proximity can be fulfilled by constraining the model to distance the latent representations of a pair of nodes respect to their connection strength. The loss function related to the first-order proximity is then defined as:

$$\begin{aligned} L_h^S &= \sum_{i,j=1}^n s_{ij} \|h_{i,u}^S - h_{j,u}^S\|_2^2 \\ &= \sum_{i,j=1}^n s_{ij} \|h_i^S - h_j^S\|_2^2 \end{aligned} \quad (6.4)$$

where

$$\begin{aligned} h_{i,1}^S &= a_1(\mathbf{W}^S x_i^S + \mathbf{b}_1^S) \\ h_{i,l}^S &= a_l(\mathbf{W}^S x_i^S + \mathbf{b}_l^S), \quad l = 2, \dots, u \end{aligned} \quad (6.5)$$

The objective function of Eq. 6.4 borrows the idea of Laplacian Eigenmaps [243], which penalizes when similar nodes are mapped far away in the embedding space.

6.2.2.4 Optimization problem

As previously outlined, CAGE first solves the structural graph embedding optimization problem:

$$\min \mathcal{L}^S(\theta^S) = \underbrace{\sum_{i=1}^n \|x_i^S - \hat{x}_i^S\|_2^2}_{L_x^S} + \underbrace{\sum_{i,j=1}^n s_{ij} \|h_i^S - h_j^S\|_2^2}_{L_h^S} + L_{reg}^S \quad (6.6)$$

where $\theta^S = (\mathbf{W}^S, \mathbf{b}^S)$ are the neural network parameters, involved in the construction of the hidden layers h_i^S and h_j^S (Eq. 6.5).

Then, once the embeddings h_i^S for node v_i is computed, it is used to constrain the attribute embedding optimization problem:

$$\min \mathcal{L}^T(\theta^T) = \underbrace{\sum_{i=1}^n \|x_i^T - \hat{x}_i^T\|_2^2}_{L_x^T} + \alpha \underbrace{\sum_{i,j=1}^n \left(\|h_i^T - h_j^T\|_2^2 - \|h_i^S - h_j^S\|_2^2 \right)}_{L_h^T} + L_{reg}^T \quad (6.7)$$

where $\alpha \in [0, 1]$ is the coefficient that leverages the contribution of the structural component and $\theta^T = (\mathbf{W}^T, \mathbf{b}^T)$ are the neural network parameters used to derive $h_i^T = h_{i,u}^T$ as:

$$\begin{aligned} h_{i,1}^T &= a_1(\mathbf{W}^T x_i^T + \mathbf{b}_1^T) \\ h_{i,l}^T &= a_l(\mathbf{W}^T x_i^T + \mathbf{b}_l^T), \quad l = 2, \dots, u \end{aligned} \quad (6.8)$$

For a more intuitive explanation, the Auto-encoder reconstruction losses have been named as L_x^S and L_x^T , while the losses involving the representation layers have been defined as L_h^S and L_h^T .

The optimization of the proposed model can be performed using *stochastic gradient descent*, or other possible variations as presented in Section 3.5.3. These optimization algorithms rely on the computation of the partial derivatives of the parameters.

As previously explained, the proposed model first solves the structural graph embeddings optimization problem $\mathcal{L}^S(\theta^S)$ (Eq. 6.6), then it constraints the textual attribute embedding optimization problem $\mathcal{L}^T(\theta^T)$ (Eq. 6.7) using the embeddings h_i^S .

Following, the passages performed to compute the partial derivatives of $\mathcal{L}^S(\theta^S)$ are presented. The computation of the derivative of $\mathcal{L}^T(\theta^T)$ is equivalent, computed with respect to the parameter set θ^T . The two loss functions differ only for the term $\|h_i^S - h_j^S\|_2^2$, which is zero if derived with respect to θ^T .

The first required step is the computation of the partial derivative of $\mathcal{L}^S(\theta^S)$ with respect to $\hat{\mathbf{W}}_u^S$ and \mathbf{W}_u^S :

$$\frac{\partial \mathcal{L}^S}{\partial \hat{\mathbf{W}}_l^S} = \frac{\partial L_x^S}{\partial \hat{\mathbf{W}}_l^S} + \frac{\partial L_{reg}^S}{\partial \hat{\mathbf{W}}_l^S} \quad (6.9)$$

$$\frac{\partial \mathcal{L}^S}{\partial \mathbf{W}_l^S} = \frac{\partial L_x^S}{\partial \mathbf{W}_l^S} + \frac{\partial L_h^S}{\partial \mathbf{W}_l^S} + \frac{\partial L_{reg}^S}{\partial \mathbf{W}_l^S} \quad (6.10)$$

The two addends can be expanded starting from the computation of the derivative of the last layer u as follows:

$$\frac{\partial L_x^S}{\partial \hat{\mathbf{W}}_u^S} = \frac{\partial L_x^S}{\partial \hat{\mathbf{X}}} \cdot \frac{\partial \hat{\mathbf{X}}}{\partial \hat{\mathbf{W}}_u^S} \quad (6.11)$$

$$\frac{\partial L_{reg}^S}{\partial \hat{\mathbf{W}}_u^S} = \frac{\partial \left(\frac{1}{2} \sum_{l=1}^u (\|\mathbf{W}_l^S\|_F^2 + \|\hat{\mathbf{W}}_l^S\|_F^2) \right)}{\partial \hat{\mathbf{W}}_u^S} = \hat{\mathbf{W}}_u^S \quad (6.12)$$

In Equation 6.11, $\partial L_x^S / \partial \hat{\mathbf{W}}_u^S$ is multiplied and divided by $\partial \hat{\mathbf{X}}$. This permits to easily compute the two terms obtained, since $\partial L_x^S / \partial \hat{\mathbf{X}}$ is equal to $2(\hat{\mathbf{X}}^S - \mathbf{X}^S) \odot \xi_i$. The second term $\partial \hat{\mathbf{X}} / \partial \hat{\mathbf{W}}_u^S$ is trivial to compute since $\hat{\mathbf{X}} = a(\hat{h}_{l-1}^S \cdot \hat{\mathbf{W}}_u^S + \hat{b}_u^S)$. The derivative of $\partial L_x^S / \partial \hat{\mathbf{W}}_u^S$ can be therefore easily estimated. Based on back-propagation, it is possible to iteratively obtain $\partial L_x^S / \partial \hat{\mathbf{W}}_l^S, l = 1, \dots, u-1$ and $\partial L_x^S / \partial \mathbf{W}_l^S, l = 1, \dots, u$.

In a similar way, Equation 6.10 can be studied by separately analyzing the three terms of $\partial \mathcal{L}^S / \partial \mathbf{W}_u^S$. The computation of the terms $\partial L_x^S / \partial \mathbf{W}_u^S$ and $\partial L_{reg}^S / \partial \mathbf{W}_u^S$ can be performed as in Equations 6.11 and 6.12, respectively. The second term $\partial L_h^S / \partial \mathbf{W}_u^S$ can be rewritten as:

$$\frac{\partial L_h^S}{\partial \mathbf{W}_u^S} = \frac{\partial L_h^S}{\partial H^S} \cdot \frac{\partial H^S}{\partial \mathbf{W}_u^S} \quad (6.13)$$

where $H^S = a(H_{l-1}^S \mathbf{W}_u^S + \hat{b}_u^S)$, resulting in a trivial derivative estimation with respect to \mathbf{W}_u^S .

For an easier computation of the first term, L_h^S can be rewritten as:

$$\sum_{i,j=1}^n s_{ij} \left(\|h_i^S - h_j^S\|_2^2 \right) = 2\text{tr}(H' L H) \quad (6.14)$$

where L is a Laplacian matrix and s_{ij} is equal to 1 for every i and j . Given S the adjacency matrix and a $n \times n$ diagonal matrix D , where $D_{ii} = \sum_j s_{ij}$, the Laplacian matrix is defined as $L = D - S$.

Finally, $\partial L_h^S / \partial H^S$ is equal to $2(L - L') \cdot H^S$.

Once all the partial derivatives have been computed, it is possible to apply the stochastic gradient descent method.

6.2.2.5 Toy Example

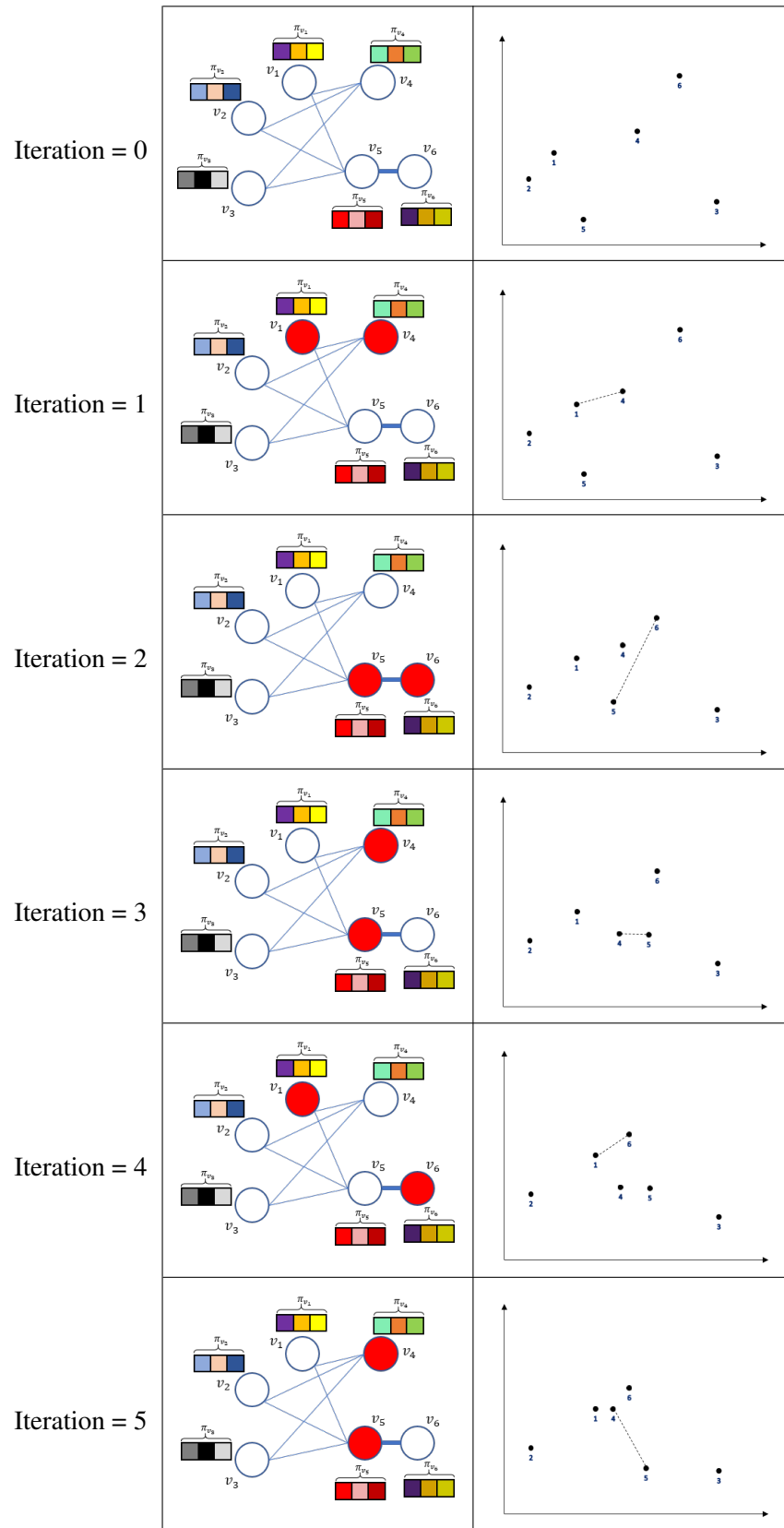
For a better understanding of the model behavior during the learning phase, Table 6.1 reports some exemplified iterations related to the consideration of different proximity measures. For each iteration, the attributed graph is shown on the left, where the red nodes correspond to the nodes actively involved in a given iteration. On the right, a simplified 2-dimensional representation of the attributed graph embeddings is provided, where the dotted line is used to highlight the node movements in the embedded space, caused by the updates of the representations. At iteration = 0, the node representations are initialized as *random* real-valued 2-dimensional vector. Then, at iteration = 1, the model evaluates the *first-order proximity*. Since there exists an edge between nodes v_1 and v_4 , they have a high first-order proximity and consequently their representations move closer. In the same way, as depicted at iteration = 2, the representations of the two connected nodes v_5 and v_6 gets even closer. It is possible to see from the embedded space that v_5 and v_6 move closer to each other with respect to v_1 and v_4 . This is due to the fact that a higher connection weight results in a larger movement, depicted as a more marked edge line in the attributed graph. Iteration = 3 involves the *second-order proximity*, which shifts closer/further the embedded representations of nodes that have many/no common neighbors. Indeed, the embedded representations of nodes v_4 and v_5 are similar, as they share 3 common node neighbors (v_1 , v_2 and v_3). An analogous mechanism is related to the *textual attribute similarity*, i.e. the more the textual contents of two nodes are similar/different, the more the representation of the nodes should be similar/different. Iteration = 4 shows the first case where v_1 and v_6 have similar attributes (depicted with similar colors). In this case, it is important to highlight that these nodes are structurally very different, indeed their representation would have never moved closer without taking into account the textual attributes. On the contrary, the representation of two nodes that are structurally similar (v_4 and v_5) may be not consistent with the attribute proximity, which results in an increase of the distance at iteration = 5.

From this simple case study, it is clear that the consideration of both textual attributes and structural information is needed since each of them is able to capture different explicit or latent relations between nodes.

6.3 Experimental Settings

The evaluation has been performed on two benchmark datasets of *citation networks*, where the *textual attribute* is the paper title and the *interaction* between two nodes corresponds to the cite relation. The aim is to *classify* the research area of the papers comprised in the datasets. This task can be commonly addressed using an ordinary Natural Language Processing approach, i.e. training a Machine Learning model over one of the well-known textual feature representations

TABLE 6.1: Toy example showing the impact of different proximity measures.



presented in Section 2.2. The experimental evaluation on the proposed feature representation model is expected to prove that the use of the relational information enhances the textual feature representation and consequently improves the classification performance, especially with respect to those methods that consider these two types of information independently.

The following sections detail the investigated datasets and the state of the art models considered for the comparison. Then, a brief overview of the evaluation framework is given.

6.3.1 Dataset

For the experimental evaluation, two state of the art *citation network* datasets have been used. A citation network is a graph where the nodes are composed of a set of research articles, the node attribute is the title and the edges are the citation relationships between articles [257].

The DBLP dataset¹ is a collection of papers extracted from DBLP [258].

The dataset has been constructed using the same procedure reported in [252], where a list of venues from 4 research areas has been selected: *database* (SIGMOD, ICDE, VLDB, EDBT, PODS, ICDT, DASFAA, SSDBM, CIKM), *data mining* (KDD, ICDM, SDM, PKDD, PAKDD), *artificial intelligence* (IJCAI, AAI, NIPS, ICML, ECML, ACML, IJCNN, UAI, ECAI, COLT, ACL, KR), *computer vision* (CVPR, ICCV, ECCV, ACCV, MM, ICPR, ICIP, ICME).

The second dataset, called *CiteSeer-M10* [252], is a subset of CiteSeer^X data² [259]. This citation network comprises 10 distinct research areas: *agriculture, archaeology, biology, computer science, financial economics, industrial engineering, material science, petroleum chemistry, physics, and social science*. Since the resultant graphs are very sparse, with a huge number of isolated nodes or isolated connected couple of nodes, the first connected component has been considered as evaluation data for each graph. In particular, the extracted DBLP graph consists of 16,191 nodes and 51,913 edges, while the CiteSeer-M10 graph comprises 2,035 and 3,356 edges.

6.3.2 Compared Models

For a comparison, the evaluation framework considered several algorithms that leverage only the graph structure (S), only the node textual attributes (T) and both of them (S+T). The considered algorithms are reported in the following:

- **LAP** (S) [243] is a nonlinear dimensionality reduction algorithm that has locality preserving properties.

¹<http://arnetminer.org/citation>

²<http://citeseerx.ist.psu.edu>

- **SDNE** (S) [250] is a deep Auto-encoder that jointly preserves the first- and second-order proximities.
- **node2vec** (S) [249] preserves a higher-order proximity between nodes by maximizing the probability of occurrence of subsequent nodes in fixed length random walks.
- **DeepWalk** (S) [248] jointly adopts random walk and skip-gram model to obtain graph embeddings.
- **KATE** (T) [100] is based on an Auto-encoder architecture specialized on dealing with natural language text.
- **Doc2Vec** (T) [253] is the Paragraph Vectors algorithm which embeds any piece of text in a distributed vector using a neural network language model.
- **DW+D2V** (S+T) [252] is a concatenation of the vectors learned by DeepWalk and Doc2Vec, for obtaining a combined representation of node structure and textual attributes.
- **TriDNR*** (S+T) is the unsupervised variant of TriDNR [252], which is a supervised neural network based approach that jointly learns graph structure, textual attributes and labels. For a fair comparison with CAGE, TriDNR has been evaluated without considering the label information.

6.3.3 Evaluation Framework

The experimental evaluation has been conducted on the task of *node classification*, aimed at associating with each research article (node) the correspondent research area (label). The textual attributes and the graph structure have been used to derive the feature representation with the proposed model. Once the embedding representation is obtained, a linear Support Vector Machine is firstly induced using the training data and then used to predict the labels of the test data. Analogously to what has been done in [252], a linear SVM has been chosen in order to reduce the impact of complex learning approaches on the classification performance, like non-linear models or sophisticated relational classifiers [260]. The datasets have been divided in training and testing sets by randomly selecting a percentage of labeled nodes. The parameters of the model have been estimated by using a sequential model-based optimization [261, 262].

The adopted performance measures are the same as presented in Section 5.1.1.2: Precision (Eq. 5.4), Recall (Eq. 5.5), F-Measure (Eq. 5.6) and Accuracy (Eq. 5.3). Since these measures can be strongly influenced by the imbalanced distribution of the data classes, both *micro* and *macro* performance (Eqs. 5.8-5.13) have been considered. Each experiment has been performed 10 times and the corresponding results have been reported by showing the average results and the corresponding standard deviation.

6.4 Experimental Results

The experimental results are presented by separately comparing the proposed model against the Representation Learning models that exploit only the graph structure (S), the ones that consider only the textual node attributes (T) and finally those approaches that combine both representations (S+T).

TABLE 6.2: Comparison of CAGE vs S models on DBLP using 50% of labeled data.

	CAGE	LAP	SDNE	node2vec	DeepWalk
Accuracy	0.7479 ± 0.004	0.4792 ± 0.004	0.4684 ± 0.003	0.6933 ± 0.004	0.4952 ± 0.000
Precision _{micro}	0.7399 ± 0.004	0.3445 ± 0.145	0.3642 ± 0.037	0.7160 ± 0.003	0.4549 ± 0.000
Precision _{macro}	0.7180 ± 0.003	0.2198 ± 0.123	0.2830 ± 0.091	0.6792 ± 0.008	0.4180 ± 0.001
Recall _{micro}	0.7479 ± 0.004	0.4792 ± 0.004	0.4684 ± 0.003	0.6933 ± 0.004	0.4952 ± 0.000
Recall _{macro}	0.6640 ± 0.005	0.2501 ± 0.000	0.2560 ± 0.002	0.5655 ± 0.005	0.3046 ± 0.000
F-measure _{micro}	0.7393 ± 0.004	0.3106 ± 0.005	0.3524 ± 0.005	0.6814 ± 0.004	0.4238 ± 0.000
F-measure _{macro}	0.6831 ± 0.005	0.1621 ± 0.001	0.2010 ± 0.002	0.5824 ± 0.005	0.2841 ± 0.000

TABLE 6.3: Comparison of CAGE vs S models on CiteSeer-M10 using 50% of labeled data.

	CAGE	LAP	SDNE	node2vec	DeepWalk
Accuracy	0.8487 ± 0.006	0.5175 ± 0.040	0.3548 ± 0.011	0.7949 ± 0.012	0.6375 ± 0.003
Precision _{micro}	0.8496 ± 0.007	0.2089 ± 0.015	0.2530 ± 0.032	0.7886 ± 0.010	0.6251 ± 0.003
Precision _{macro}	0.7928 ± 0.076	0.4452 ± 0.086	0.1162 ± 0.027	0.6203 ± 0.069	0.4523 ± 0.010
Recall _{micro}	0.8487 ± 0.006	0.3576 ± 0.040	0.3548 ± 0.011	0.7949 ± 0.012	0.6375 ± 0.003
Recall _{macro}	0.6817 ± 0.048	0.1426 ± 0.001	0.1450 ± 0.001	0.5043 ± 0.031	0.4036 ± 0.008
F-measure _{micro}	0.8456 ± 0.006	0.2124 ± 0.039	0.2113 ± 0.012	0.7833 ± 0.012	0.6281 ± 0.003
F-measure _{macro}	0.7168 ± 0.055	0.0858 ± 0.010	0.0875 ± 0.005	0.5252 ± 0.034	0.4204 ± 0.009

Tables 6.2 and 6.3 show the performance considering 50% of the training set. It is possible to grasp from the results on both datasets that CAGE achieves the best performance comparing to all the considered structural graph embedding models. As expected, SDNE is the approach that achieves the highest performance, after CAGE. From both tables, it clearly emerges that considering both the textual attributes and the interactions of the nodes provides significant improvements compared to the results obtained by modeling only the relational information.

Similarly, the proposed model is able to achieve better results compared to recent Natural Language Processing models, i.e. KATE and Doc2Vec (Tabs. 6.4 and 6.5). When compared to KATE, CAGE obtains satisfactory improvements, additionally demonstrating that the textual feature representation can be enhanced by considering the relational information. The results of Doc2Vec, DeepWalk and DW+D2V give further evidence of this statement, as the consideration of the concatenated representation space of DeepWalk and Doc2Vec is able to achieve significant improvements with respect to the single methods: DeepWalk for the graph structure and

Doc2Vec for the text. The unsupervised version of TriDNR*, which naturally includes textual and relational information, achieves good performance that however are lower than the ones obtained by CAGE (for all the performance measures).

TABLE 6.4: Comparison of CAGE vs T and S+T models on DBLP using 50% of labeled data.

	CAGE	KATE	Doc2Vec	DW+D2V	TriDNR*
Accuracy	0.7479 ± 0.004	0.745 ± 0.004	0.7095 ± 0.000	0.7106 ± 0.000	0.7019 ± 0.003
Precision _{micro}	0.7399 ± 0.004	0.7377 ± 0.003	0.6924 ± 0.000	0.6980 ± 0.000	0.6886 ± 0.003
Precision _{macro}	0.7180 ± 0.003	0.7278 ± 0.004	0.6543 ± 0.000	0.6654 ± 0.000	0.6558 ± 0.003
Recall _{micro}	0.7479 ± 0.004	0.7450 ± 0.004	0.5001 ± 0.000	0.7106 ± 0.000	0.7019 ± 0.003
Recall _{macro}	0.6640 ± 0.005	0.6433 ± 0.005	0.6080 ± 0.000	0.6133 ± 0.000	0.5952 ± 0.002
F-measure _{micro}	0.7393 ± 0.004	0.7325 ± 0.004	0.4005 ± 0.000	0.6992 ± 0.000	0.6885 ± 0.003
F-measure _{macro}	0.6831 ± 0.005	0.6702 ± 0.004	0.6198 ± 0.000	0.6309 ± 0.000	0.6144 ± 0.002

TABLE 6.5: Comparison of CAGE vs T and S+T models on CiteSeer-M10 using 50% of labeled data.

	CAGE	KATE	Doc2Vec	DW+D2V	TriDNR*
Accuracy	0.8487 ± 0.006	0.7482 ± 0.002	0.7750 ± 0.000	0.8092 ± 0.002	0.8128 ± 0.004
Precision _{micro}	0.8496 ± 0.007	0.8382 ± 0.003	0.7785 ± 0.000	0.8019 ± 0.002	0.8041 ± 0.003
Precision _{macro}	0.7928 ± 0.076	0.7244 ± 0.022	0.6314 ± 0.000	0.6435 ± 0.011	0.6515 ± 0.013
Recall _{micro}	0.8487 ± 0.006	0.8360 ± 0.002	0.7750 ± 0.000	0.8092 ± 0.002	0.8128 ± 0.004
Recall _{macro}	0.6817 ± 0.048	0.6612 ± 0.003	0.4708 ± 0.000	0.5682 ± 0.003	0.5452 ± 0.003
F-measure _{micro}	0.8456 ± 0.006	0.8282 ± 0.010	0.7642 ± 0.000	0.8031 ± 0.002	0.8039 ± 0.004
F-measure _{macro}	0.7168 ± 0.055	0.5871 ± 0.033	0.4775 ± 0.000	0.5912 ± 0.005	0.5711 ± 0.003

Further results have been reported (Tabs. 6.6-6.9) in terms of F-measure_{micro} by varying the percentage of the training set from 10% to 90% with a step of 10%.

From Tables 6.6 and 6.7, it is possible to draw similar conclusions to the ones referred to Tables 6.2 and 6.3. CAGE achieves the best results, immediately followed by SDNE. This confirms that considering relationships to enhance the textual representation plays a fundamental role for predicting the research area (label) of each research article (node).

TABLE 6.6: Comparison of CAGE vs S models in terms of F-measure_{micro} on DBLP using different % of labeled data.

	CAGE	LAP	SDNE	node2vec	DeepWalk
10 %	0.7118 ± 0.005	0.3111 ± 0.001	0.3697 ± 0.004	0.6641 ± 0.003	0.4373 ± 0.000
20 %	0.7310 ± 0.002	0.3109 ± 0.001	0.3633 ± 0.005	0.6690 ± 0.005	0.4201 ± 0.000
30 %	0.7324 ± 0.001	0.3130 ± 0.002	0.3616 ± 0.005	0.6765 ± 0.003	0.4235 ± 0.000
40 %	0.7372 ± 0.004	0.3095 ± 0.004	0.3564 ± 0.005	0.6794 ± 0.002	0.4208 ± 0.000
50 %	0.7402 ± 0.005	0.3106 ± 0.005	0.3524 ± 0.005	0.6814 ± 0.004	0.4238 ± 0.000
60 %	0.7381 ± 0.006	0.3143 ± 0.005	0.3481 ± 0.005	0.6805 ± 0.003	0.4234 ± 0.000
70 %	0.7439 ± 0.005	0.3124 ± 0.007	0.3491 ± 0.007	0.6822 ± 0.004	0.4268 ± 0.000
80 %	0.7355 ± 0.002	0.3137 ± 0.006	0.3419 ± 0.008	0.6806 ± 0.010	0.4294 ± 0.000
90 %	0.7421 ± 0.010	0.3246 ± 0.016	0.3512 ± 0.014	0.6944 ± 0.012	0.4365 ± 0.000

TABLE 6.7: Comparison of CAGE vs S models in terms of F-measure_{micro} on CiteSeer-M10 using different % of labeled data.

	CAGE	LAP	SDNE	node2vec	DeepWalk
10 %	0.7611 ± 0.012	0.1789 ± 0.062	0.2238 ± 0.051	0.6882 ± 0.013	0.5163 ± 0.002
20 %	0.7964 ± 0.009	0.1947 ± 0.143	0.2390 ± 0.013	0.7180 ± 0.012	0.5867 ± 0.003
30 %	0.8152 ± 0.013	0.1885 ± 0.135	0.2369 ± 0.014	0.7529 ± 0.007	0.6032 ± 0.007
40 %	0.8177 ± 0.014	0.2036 ± 0.083	0.2384 ± 0.009	0.7758 ± 0.009	0.6143 ± 0.002
50 %	0.8260 ± 0.008	0.2124 ± 0.039	0.2113 ± 0.012	0.7817 ± 0.009	0.6281 ± 0.003
60 %	0.8306 ± 0.009	0.2110 ± 0.028	0.2349 ± 0.011	0.7874 ± 0.018	0.6345 ± 0.004
70 %	0.8282 ± 0.016	0.2058 ± 0.018	0.2146 ± 0.018	0.7878 ± 0.011	0.6470 ± 0.007
80 %	0.8388 ± 0.020	0.2098 ± 0.036	0.2322 ± 0.039	0.7868 ± 0.041	0.6438 ± 0.008
90 %	0.8577 ± 0.044	0.1868 ± 0.037	0.2410 ± 0.025	0.8085 ± 0.014	0.6277 ± 0.016

Analogous results can also be observed in Tables 6.8 and 6.9. CAGE outperforms both models that consider only the textual attributes and models that combine textual and structure information.

Another important conclusion that it is possible to derive from the results is that the proposed model shows the best improvement with respect to the state of the art models in the case of little available training data (from 10% to 50% of the training data). The robustness with respect to the size of the training data is due to the joint contribution of textual and structural information conveyed by the attributed graph.

TABLE 6.8: Comparison of CAGE vs T and S+T models in terms of F-measure_{micro} on DBLP using different % of labeled data.

	CAGE	KATE	Doc2Vec	DW+D2V	TriDNR*
10 %	0.7118 ± 0.005	0.6927 ± 0.007	0.3668 ± 0.000	0.6541 ± 0.000	0.6620 ± 0.003
20 %	0.7310 ± 0.002	0.7122 ± 0.003	0.3898 ± 0.000	0.6799 ± 0.000	0.6743 ± 0.003
30 %	0.7324 ± 0.001	0.7275 ± 0.001	0.3943 ± 0.000	0.6971 ± 0.000	0.6868 ± 0.003
40 %	0.7372 ± 0.004	0.7319 ± 0.002	0.3968 ± 0.000	0.6975 ± 0.000	0.6872 ± 0.003
50 %	0.7402 ± 0.005	0.7325 ± 0.004	0.4005 ± 0.000	0.6992 ± 0.000	0.6885 ± 0.003
60 %	0.7381 ± 0.006	0.7374 ± 0.007	0.4091 ± 0.000	0.7040 ± 0.000	0.6903 ± 0.002
70 %	0.7439 ± 0.005	0.7368 ± 0.003	0.4097 ± 0.000	0.7072 ± 0.000	0.6916 ± 0.004
80 %	0.7355 ± 0.002	0.7362 ± 0.003	0.4187 ± 0.000	0.7088 ± 0.000	0.6889 ± 0.003
90 %	0.7421 ± 0.010	0.7378 ± 0.005	0.4116 ± 0.000	0.7261 ± 0.000	0.7059 ± 0.004

TABLE 6.9: Comparison of CAGE vs T and S+T models in terms of F-measure_{micro} on CiteSeer-M10 using different % of labeled data.

	CAGE	KATE	Doc2Vec	DW+D2V	TriDNR*
10 %	0.7611 ± 0.012	0.7183 ± 0.009	0.6917 ± 0.000	0.7104 ± 0.003	0.7324 ± 0.003
20 %	0.7964 ± 0.009	0.7482 ± 0.005	0.7423 ± 0.000	0.7759 ± 0.001	0.7739 ± 0.006
30 %	0.8152 ± 0.013	0.7667 ± 0.006	0.7369 ± 0.000	0.7799 ± 0.002	0.7784 ± 0.004
40 %	0.8177 ± 0.014	0.7806 ± 0.010	0.7541 ± 0.000	0.7876 ± 0.003	0.7889 ± 0.005
50 %	0.8260 ± 0.008	0.7825 ± 0.013	0.7642 ± 0.000	0.8031 ± 0.002	0.8039 ± 0.004
60 %	0.8306 ± 0.009	0.8069 ± 0.009	0.7698 ± 0.000	0.8144 ± 0.000	0.8062 ± 0.006
70 %	0.8282 ± 0.016	0.8007 ± 0.014	0.7661 ± 0.000	0.8187 ± 0.005	0.8114 ± 0.005
80 %	0.8388 ± 0.020	0.8055 ± 0.016	0.7744 ± 0.000	0.8178 ± 0.002	0.8211 ± 0.005
90 %	0.8577 ± 0.044	0.8102 ± 0.023	0.7751 ± 0.000	0.8183 ± 0.004	0.8033 ± 0.004

The experimental evaluation further demonstrates that the enhancement of textual representation models with the relational information can provide significant improvements in terms of different performance measures.

As future work, the definition of a joint model able to simultaneously learn both attribute and graph embeddings is expected to result in better evaluation performance.

In particular, formalizing a unique optimization problem, where the contribution of attributes and graph structure will be determined according to Lagrangian methods, would lead to even more valuable feature representations.

Chapter 7

Conclusion and Future Works

Since the advent of Web 2.0, the amount of available user-generated content has exponentially increased to unprecedented levels. This immense source of information is often not exploited to its full potential mainly due to two challenging problems: (i) natural language text is expressed in the form of discrete symbols, making difficult to obtain a mathematical representation that machines can elaborate and (ii) keeping track of the ever-increasing number of generated data creates the need of computationally efficient models able to extract valuable knowledge from data that have not been manually annotated.

This thesis addresses these issues by proposing different novel Natural Language Processing models enhanced by the representation obtained with unsupervised Deep Learning. These models are then able to learn a vector representation from text that encodes semantic and syntactic meanings of the language units and to efficiently take advantage of a large amount of user-generated content for identifying and disentangling their underlying explanatory factors. In particular, this thesis provides two contributions for addressing the problem of making sense of user-generated content, exploiting high-level representations of text and relational structure.

First, it has been demonstrated that the joint exploitation of Natural Language Processing and Deep Learning models can significantly improve the understanding and interpreting of user-generated content.

In particular, this thesis has initially shown the contribution of learning Deep Learning feature representation (Auto-Encoders and Word Embeddings) for several Natural Language Processing tasks. First, a novel model (L2A) has been provided for dealing with the problem of adapting the classification of named entity types of a NER system to different ontologies by including a representation of named entities derived by Word Embeddings. Second, this thesis has introduced an unsupervised model for Named Entity Linking based on a novel heterogeneous representation space characterized by common embeddings of both words and named entities. Third, the

problem of Sentiment Analysis has been addressed by proposing a model, based on an Auto-Encoder architecture and Ensemble Learning, that is able to adapt the prediction of a sentiment orientation from a source domain to a target domain, where little or no training data is available. Finally, this thesis contributed to provide an unsupervised model for Irony Detection that is able to take into account domain-aware ironic orientation of words derived by Word Embeddings. The proposed models for making sense of user-generated content has shown significant performance and enhanced generalization abilities. Moreover, it has been proved that leveraging large amount of unlabelled data with Deep Learning models can strongly help to acquire meaningful data representation even in noisy environments such as Social Media.

The second main contribution of this thesis is concerned with the enhancement of textual feature representations by taking into account the relational structure underlying user-generated content. The proposed Representation Learning model (CAGE), based on Auto-encoder architectures, is able to encode both the textual content (attributes) and the relational structure of user-generated data (graph) for obtaining a common vector representation. The experimental evaluation, also in this case, has demonstrated that the enhancement of textual representation models with the relational information can provide significant improvements in terms of classification performance.

There are several possible research directions that can be investigated to further improve the results presented in this thesis.

The first issue is concerned with the cross-task generalization abilities. The representations obtained by Deep Learning models have demonstrated excellent generalization abilities across several domains. However, learning a representation that is also independent on the task would permit to extensively reduce the amount of computational training time and the human effort on developing different models for each task. In the simplest case, a representation model able to express this property would then be able to obtain satisfactory results over all the presented Natural Language Processing tasks by only using off-the-shelf Machine Learning algorithms.

An analogous research study that can be conducted regards the ability to perform multilingual Natural Language Processing tasks for making sense of user-generated content. Although the majority of natural language textual content on the Web is written in English, the amount of text related to other languages is more than enough for creating an unlabelled training set. In this context, the abilities of Deep Learning models to disentangle the factors of variations in the data can strongly and positively impact on the addressed multilingual tasks [263, 264].

Regarding user-generated content provided in a networked environment, it is important to take into account that relationships can be uncertain. The collection of large amount of noisy data on the Web comes with imprecision and uncertainty: we cannot be completely confident that data about individuals, or the connections between them, is accurate and truthful [265]. In this

context, uncertain, or probabilistic, graphs have been increasingly deployed to represent noisy linked data in many emerging application scenarios [266–268]. In many cases, uncertainty or imprecise information become a critical issue to understand and effectively take advantage of the information contained in such relational environments.

The more comprehensive work that can be performed starting from this thesis regards the exploitation of attributed graph structure as input for learning meaningful representation for making sense of user-generated content. For example, a Social Network can be seen as a heterogeneous attributed graph, where the nodes are the users and the posts, while edges are the different relations that exist between users (e.g. friendship), users and posts (e.g. like, share), posts and posts (e.g. comment). For instance, obtaining expressive representation could help the classification of the sentiment of the posts or of the users. In this context, several works have already suggested that the exploitation of users' social relations could help to detect their opinions [269–271]. Mapping users and posts to an embedded space, where similarities are the expression of several explicit and implicit properties, would also permit to easily and more accurately suggest new friends or new posts related to common topics of interest and not based only on the structural connections.

Another example task, which heterogeneous attributed graph embedding are expected to improve, is Named Entity Linking. In this case, a graph would be composed of words and entities, where the relationships would be related to the number of times a word has been used to denote an entity. Mapping these entities to a common embedded space would result in more meaningful representations, where finding the corresponding entity associated with a word would be easier.

Considering the increasing volume of interactions in Social Media that are characterized by temporal information, there are considerable motivations to develop latent representations for relational data over time. These interactions can be explicit observations, e.g. a follower relationships, or more complex social interactions among users, e.g. two users may share similar tags or read the same feeds. Efficiently dealing with these data poses different challenges on treating and modeling networked data consisting of multiple co-evolving dimensions, e.g. users, tags, feeds, comments, etc. A common solution is to represent a dynamic graph through a set on independent snapshots and to adopt existing embedding algorithms. However, this usually leads to unsatisfactory performance in terms of stability, flexibility and efficiency [272]. An alternative solution that could be investigated as future work is to consider the dynamics directly into the embedding model, in order to give more importance to more recent and actual data.

Appendix A

TWINE: A real-time system for TWEet analysis via INformation Extraction

A.1 Introduction

The Word Embeddings model for Named Entity Linking presented in Section 5.2 has been developed and integrated in a real-time system that collects, analyzes and shows entities spread through Social Media. In particular, the investigated environment is Twitter, which is a popular microblogging service that is particularly focused on the speed and ease of publication. This choice is motivated by the fact that nearly 300 million of active users share over 500 million of posts¹ everyday. The proposed system, named TWINE, has the characteristic of combining a big data architecture and user interface in order to perform and explore real-time analysis of social media content streams.

In particular, TWINE has been defined in order to:

- perform real-time monitoring of tweets related to a set of topics of interest, with unrestricted keywords;
- explore the information extracted by semantic-based analysis of large amount of tweets, i.e. (i) recognition of named entities and the information of the correspondent KB resources, (ii) multi-dimensional spatial geo-tagging for each tweet, including the geo-localization of the named entities identified as locations and (iii) two semantic-driven interactive visualization interfaces.

The following section will present the details of the architecture for supporting real-time tweets analysis and the description of the conceived graphical user interface.

¹<http://www.internetlivestats.com/>

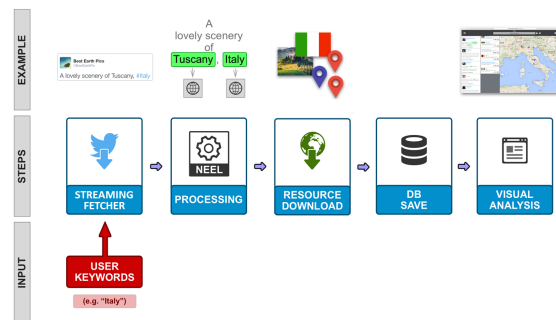


FIGURE A.1: TWINE system overview.

A.2 TWINE system

TWINE, acronym for TWEet analysis via INFORMATION Extraction, is a real-time system for the analysis and exploration of information extracted from Twitter data. Figure A.1 outlines its macro steps coupled with corresponding examples.

Given a set of keywords provided by the user (e.g. “Italy”) as input query, the system fetches the stream of all the tweets (text and tweet’s author information) matching the keywords using Twitter APIs. Next, each tweet text is processed by the Named Entity Recognition and Linking (NEEL) pipeline.

Once each tweet text has been processed by Conditional Random Fields for identifying entities and these entities are linked to the KB by using the proposed model in Section 5.2, several additional information are retrieved: image, text description, type and coordinates (if the entity is a location) are taken from the KB, while the account profile of the author of the tweet is collected from Twitter, where the location information is resolved with a georeferencing system.

This information is subsequently stored in a database that incrementally enriches information generated by the precedent phases. Then, the TWINE web interface fetches the stored data from the database for populating two different interactive visualization interfaces.

A.2.1 System Architecture

TWINE is implemented using a centralized system architecture, as shown in Figure A.2. The main requirement was to develop a system able to process real-time large incoming data streams.

In TWINE, all the afore-mentioned processes are triggered by the user from the client and elaborated on the server-side, i.e. the streaming fetching phase, the NEEL processing, the KB resources retrieval, the geo-codification of the locations and the database storing.

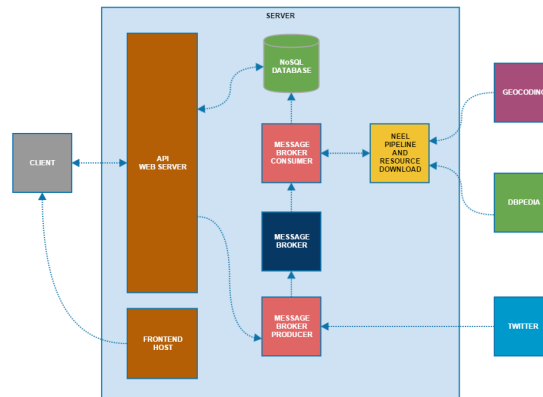


FIGURE A.2: TWINE system architecture.

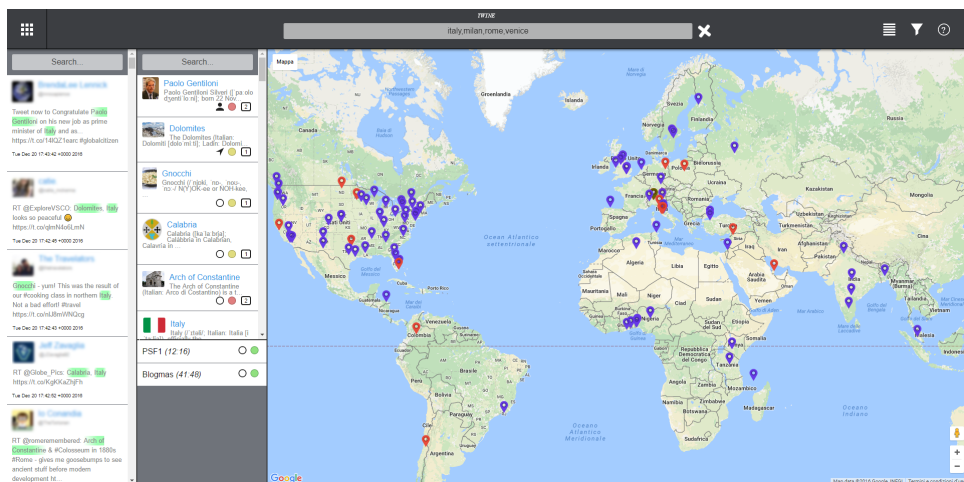


FIGURE A.3: TWINE Map View snapshot.

According to this design implementation all the computations are performed on the server. This improves the independence on the client technical specifications, preventing different problems such as slow loading, high processor usage and even freezing.

The system architecture, presented in Figure A.2, is composed of several independent modules:

External Services. The system makes use of Twitter APIs for fetching the streaming of tweets given an input query, a SPARQL endpoint over the DBpedia data set for the retrieval of the KB resource information and a georeferencing system, OpenStreetMap², to obtain the geographic coordinates from the tweet author account's profile location.

²<https://www.openstreetmap.org/>

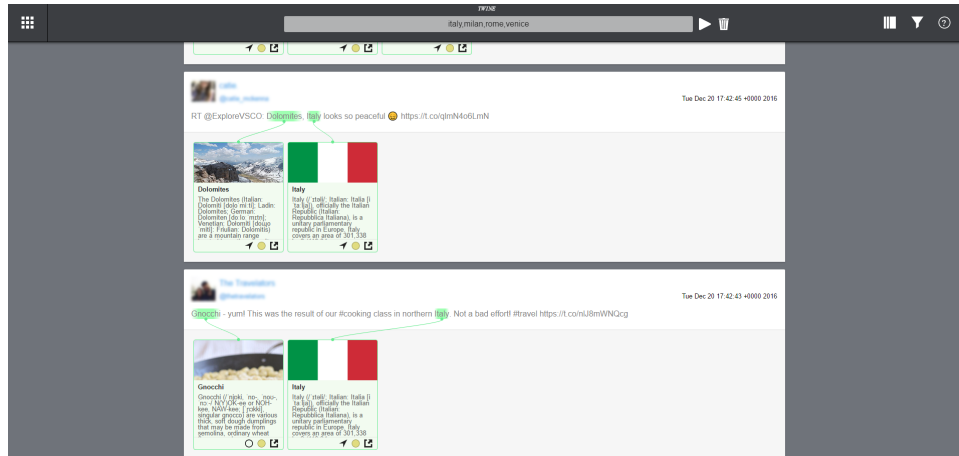


FIGURE A.4: TWINE List View snapshot.

NEEL pipeline. In order to extract and link entities contained in a tweet, the NEEL pipeline uses the two following subcomponents:

- The NER system of Ritter et al. [109].
- The NEL model proposed in Section 5.2.

Message Broker system. This module is necessary to build pipelines for processing streaming data in real time, in such a way that components can exchange data reliably. The Apache Kafka platform³ permits us to store and process the data in a fault-tolerant way and to ignore the latency due to the Information Extraction processing.

Database. All the source and processed data are stored in a NoSQL database. In particular, it has been chosen a MongoDB⁴ database because of its flexibility, horizontal scalability and its representation format that is particularly suitable for storing Twitter contents.

Frontend host and API web server. The presence of these two server-side modules is motivated by the need of making the TWINE user-interface independent on its functionalities. In this way, the modularity and flexibility of the entire system are improved.

³<https://kafka.apache.org/>

⁴<http://www.mongodb.org/>

A.2.2 User Interface

TWINE provides two different visualizations of the extracted information: the Map View, which shows the different geo-tags associated with tweets in addition to the NEEL output, and the List View, that better emphasizes the relation between the text and its named entities.

The Map View (Figure A.3) provides in the top panel a textual search bar where users can insert keywords related to their topic of interest (e.g. *italy, milan, rome, venice*). The user can also, from left to right, start and stop the stream fetching process, clear the current results, change View and apply semantic filters related to the geo-localization and KB resource characteristics, i.e. type and classification confidence score.

Then, in the left-hand panel the user can read the content of each fetched tweet (text, user information and recognized named entities) and directly open it in the Twitter platform.

The center panel can be further divided into two sub-panels: the top one shows the information about the Knowledge Base resources related to the linked named entities present in the tweets (image, textual description, type as symbol and the classification confidence score), and the bottom one provides the list of the recognized named entities for which it does not exist a correspondence in the KB, i.e. NIL entities.

These two panels, the one that reports the tweets and the one with the recognized and linked KB resources, are responsive. For example, by clicking on the entity *Italy* in the middle panel, only tweets containing the mention of the entity *Italy* will be shown in the left panel. Respectively, by clicking on a tweet, the center panel will show only the related entities.

In the right-hand panel, the user can visualize the geo-tag extracted from the tweets, (i) the original geo-location where the post is emitted (*green marker*), (ii) the user-defined location for the user account's profile (*blue marker*) and (iii) the geo-location of the named entities extracted from the tweets, if the corresponding KB resource has the latitude-longitude coordinates (*red marker*).

Finally, a text field is present at the top of the first two panels to filter the tweets and KB resources that match specific keywords.

The List View is reported in Figure A.4. Differently from the Map View, the focus is on the link between the words, i.e. recognized named entities, and the corresponding KB resources. In the reported example, this visualization is more intuitive for catching the meaning of *Dolomites* and *Gnocchi* thanks to a direct connection between the named entities and the snippet and the image of associated KB resources.

A.3 Conclusion

This appendix has briefly overviewed TWINE, a system that provides an efficient real-time data analytics platform for social media streaming content. The system is supported by a scalable and modular architecture and by an intuitive and interactive user interface.

As future work, it is intended to implement a distributed solution in order to more efficiently manage huge quantity of data. Additionally, current integrated modules will be improved: the NEEL pipeline will be replaced by a multilingual method, the web interface will include more insights such as the user network information, a heatmap visualization and a time control filter.

Appendix B

Additional Results for LearningToAdapt model

TABLE B.1: Accuracy performance of L2A model considering Word Embeddings feature space. For each dataset, the best results are reported in bold.

		2015				2016			
		X_E		$X_{P \cup E}$		X_E		$X_{P \cup E}$	
		Wiki2Vec	GoogleNews	Wiki2Vec	GoogleNews	Wiki2Vec	GoogleNews	Wiki2Vec	GoogleNews
BN	mean	0.64	0.59	0.64	0.59	0.47	0.48	0.48	0.48
	max	0.56	0.54	0.62	0.57	0.49	0.51	0.48	0.50
	min	0.56	0.54	0.57	0.58	0.49	0.50	0.49	0.50
	first	0.58	0.57	0.59	0.59	0.44	0.50	0.45	0.50
DT	mean	0.49	0.50	0.65	0.65	0.49	0.45	0.48	0.42
	max	0.42	0.48	0.64	0.66	0.50	0.35	0.44	0.40
	min	0.46	0.51	0.64	0.64	0.34	0.41	0.44	0.47
	first	0.44	0.45	0.57	0.66	0.46	0.31	0.36	0.47
KNN	mean	0.55	0.55	0.64	0.68	0.56	0.53	0.56	0.54
	max	0.53	0.54	0.61	0.65	0.51	0.51	0.51	0.54
	min	0.52	0.55	0.60	0.66	0.48	0.51	0.52	0.54
	first	0.49	0.52	0.63	0.60	0.43	0.46	0.46	0.54
MLR	mean	0.53	0.48	0.61	0.60	0.57	0.53	0.43	0.47
	max	0.54	0.49	0.62	0.55	0.46	0.49	0.53	0.47
	min	0.57	0.47	0.61	0.59	0.55	0.53	0.43	0.44
	first	0.55	0.49	0.57	0.58	0.55	0.55	0.51	0.51
MLP	mean	0.59	0.58	0.73	0.74	0.57	0.67	0.60	0.55
	max	0.62	0.61	0.73	0.71	0.54	0.62	0.55	0.55
	min	0.59	0.60	0.72	0.72	0.55	0.61	0.58	0.55
	first	0.57	0.58	0.73	0.69	0.53	0.57	0.54	0.53
NB	mean	0.64	0.59	0.65	0.60	0.52	0.57	0.50	0.54
	max	0.56	0.54	0.61	0.57	0.56	0.58	0.57	0.55
	min	0.55	0.54	0.57	0.58	0.55	0.56	0.54	0.56
	first	0.58	0.57	0.59	0.59	0.48	0.55	0.50	0.50
SVM	mean	0.59	0.62	0.74	0.69	0.59	0.62	0.62	0.59
	max	0.59	0.63	0.74	0.74	0.58	0.64	0.60	0.60
	min	0.59	0.61	0.74	0.68	0.59	0.60	0.61	0.57
	first	0.56	0.57	0.72	0.67	0.56	0.55	0.58	0.55

TABLE B.2: Class-Wise Accuracy Contribution (%) on #Micropost2015 Test set considering Word Embeddings feature space (X_E and X_{P-E}).

	MLP						SVM					
	mean		max		min		mean		max		min	
	X_E	X_{PUE}	X_E	X_{PUE}	X_E	X_{PUE}	X_E	X_{PUE}	X_E	X_{PUE}	X_E	X_{PUE}
Character	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Event	0.09	3.00	0.09	2.82	0.09	3.18	0.09	2.82	0.09	2.82	0.18	2.74
Location	31.95	38.92	33.89	39.01	32.22	38.13	32.48	39.81	31.51	39.81	31.77	39.63
Organization	5.83	7.33	6.44	7.15	5.91	7.33	5.03	7.06	5.21	7.15	6.00	7.15
Person	19.59	22.07	20.30	21.98	19.86	21.71	20.30	22.68	19.95	21.98	19.95	22.68
Product	0.79	0.79	0.97	1.06	0.62	0.88	0.79	1.15	1.24	1.24	0.79	1.24
Thing	0.62	0.88	0.35	0.88	0.53	0.88	0.88	1.06	0.97	1.06	0.88	1.06
Overall	58.87	72.99	62.05	72.90	59.22	72.11	59.58	74.58	58.96	74.05	59.58	74.49

TABLE B.3: Class-Wise Accuracy Contribution (%) on #Micropost2016 Test set considering Word Embeddings feature space (X_E and X_{P-E}).

	MLP						SVM					
	mean		max		min		mean		max		min	
	X_E	X_{PUE}	X_E	X_{PUE}	X_E	X_{PUE}	X_E	X_{PUE}	X_E	X_{PUE}	X_E	X_{PUE}
Character	9.18	2.04	6.12	5.10	3.06	7.14	4.08	3.06	5.10	3.06	3.06	3.06
Event	1.02	1.02	1.02	1.02	1.02	1.02	1.02	1.02	1.02	1.02	1.02	1.02
Location	15.31	14.29	14.29	13.27	13.27	15.31	15.31	13.27	14.29	13.27	15.31	14.29
Organization	5.10	3.06	5.10	5.10	4.08	4.08	5.10	5.10	5.10	5.10	5.10	4.08
Person	21.43	21.43	20.41	20.41	19.39	19.39	20.41	21.43	22.45	23.47	20.41	20.41
Product	14.29	12.24	14.29	10.20	13.27	13.27	14.29	13.27	14.29	12.24	14.29	13.27
Thing	1.02	1.02	1.02	0.00	1.02	1.02	2.04	2.04	2.04	2.04	1.02	1.02
Overall	67.35	55.10	62.24	55.10	55.10	61.22	62.24	59.18	64.29	60.20	60.20	57.14

TABLE B.4: Precision, Recall, F-Measure and STMM on #Micropost2015 Test set considering Word Embeddings feature space (X_E and X_{P-E}).

	MLP						SVM					
	mean		max		min		mean		max		min	
	X_E	X_{PUE}	X_E	X_{PUE}	X_E	X_{PUE}	X_E	X_{PUE}	X_E	X_{PUE}	X_E	X_{PUE}
Precision	0.61	0.74	0.60	0.74	0.61	0.74	0.60	0.75	0.58	0.75	0.61	0.75
Recall	0.59	0.73	0.62	0.73	0.59	0.72	0.60	0.75	0.59	0.74	0.60	0.74
F-Measure	0.32	0.49	0.33	0.49	0.32	0.49	0.33	0.51	0.35	0.51	0.34	0.51
STMM	0.59	0.73	0.60	0.73	0.58	0.72	0.59	0.74	0.57	0.74	0.59	0.74

TABLE B.5: Precision, Recall, F-Measure and STMM on #Micropost2016 Test set considering Word Embeddings feature space (X_E and X_{P-E}).

	MLP						SVM					
	mean		max		min		mean		max		min	
	X_E	X_{PUE}	X_E	X_{PUE}	X_E	X_{PUE}	X_E	X_{PUE}	X_E	X_{PUE}	X_E	X_{PUE}
Precision	0.73	0.70	0.72	0.68	0.72	0.71	0.67	0.69	0.65	0.70	0.65	0.67
Recall	0.57	0.60	0.54	0.55	0.55	0.58	0.59	0.62	0.58	0.60	0.59	0.61
F-Measure	0.50	0.53	0.49	0.50	0.50	0.53	0.58	0.60	0.51	0.54	0.51	0.59
STMM	0.52	0.55	0.47	0.50	0.49	0.53	0.55	0.56	0.52	0.52	0.53	0.55

TABLE B.6: Class-Wise Accuracy Contribution (%) on #Micropost2015 Test set of L2A model and baselines.

Entity Type	Baselines			L2A								
	BL-D	BL-P1	BL-P2	BN (X_p)	NB (X_p)	MLR (X_p)	MLP (X_p)	SVM (X_p)	DT (X_p)	KNN (X_p)	SVM _{mean} (X_E)	SVM _{mean} ($X_{P \cup E}$)
Character	0.00	0.26	0.00	0.00	0.09	0.00	0.00	0.00	0.00	0.00	0.0	0.0
Event	0.00	3.80	2.91	1.32	0.09	0.00	0.00	0.00	1.44	0.35	0.09	2.82
Location	37.51	42.01	37.51	41.39	43.78	40.69	43.42	40.78	28.37	43.69	32.48	39.81
Organization	4.41	4.41	4.41	6.00	7.41	8.47	7.24	8.74	8.65	7.77	5.03	7.06
Person	20.83	20.83	16.95	18.80	20.39	20.92	22.33	20.30	30.29	20.92	20.30	22.68
Product	1.50	1.50	0.88	0.44	0.26	1.41	1.68	1.41	0.48	1.59	0.79	1.15
Thing	0.18	0.18	2.12	0.00	0.00	0.00	0.00	0.00	0.00	0.18	0.88	1.06
Overall	64.43	72.99	64.78	67.96	72.02	71.49	74.67	71.23	69.23	74.49	59.58	74.58

TABLE B.7: Class-Wise Accuracy Contribution (%) on #Micropost2016 Test set of L2A model and baselines.

Entity Type	Baselines			L2A								
	BL-D	BL-P1	BL-P2	BN (X_p)	NB (X_p)	MLR (X_p)	MLP (X_p)	SVM (X_p)	DT (X_p)	KNN (X_p)	SVM _{mean} (X_E)	SVM _{mean} ($X_{P \cup E}$)
Character	0.00	16.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.04	2.04	1.02
Event	0.00	1.02	1.02	1.02	1.02	0.00	1.02	0.00	1.02	1.02	1.02	1.02
Location	7.14	10.20	8.16	11.22	11.22	11.22	13.27	12.24	14.29	12.24	15.31	13.27
Organization	3.06	3.06	0.00	2.04	2.04	4.08	2.04	4.08	4.08	4.08	5.10	5.10
Person	21.43	21.43	15.31	18.37	18.37	22.45	23.47	21.43	22.45	21.43	20.41	21.43
Product	13.27	13.27	1.02	10.20	10.20	13.27	13.27	13.27	12.24	7.14	14.29	13.27
Thing	1.02	1.02	5.10	1.02	1.02	0.00	0.00	0.00	0.00	0.00	2.04	2.04
Overall	45.92	66.33	30.61	43.88	43.88	51.02	53.06	51.02	54.08	47.96	62.24	59.18

TABLE B.8: Precision, Recall, F-Measure and STMM on #Micropost2015 Test set of L2A model and baselines.

Entity Type	Baselines			L2A								
	BL-D	BL-P1	BL-P2	BN (X_p)	NB (X_p)	MLR (X_p)	MLP (X_p)	SVM (X_p)	DT (X_p)	KNN (X_p)	SVM _{mean} (X_E)	SVM _{mean} ($X_{P \cup E}$)
Precision	0.67	0.67	0.75	0.67	0.67	0.63	0.62	0.63	0.72	0.67	0.60	0.75
Recall	0.66	0.66	0.63	0.68	0.68	0.70	0.72	0.70	0.69	0.69	0.60	0.75
F-Measure	0.37	0.37	0.43	0.37	0.37	0.36	0.37	0.36	0.36	0.36	0.33	0.51
STMM	0.66	0.66	0.68	0.67	0.67	0.66	0.66	0.66	0.66	0.66	0.59	0.74

TABLE B.9: Precision, Recall, F-Measure and STMM on #Micropost2016 Test set of L2A model and baselines.

Entity Type	Baselines			L2A								
	BL-D	BL-P1	BL-P2	BN (X_p)	NB (X_p)	MLR (X_p)	MLP (X_p)	SVM (X_p)	DT (X_p)	KNN (X_p)	SVM _{mean} (X_E)	SVM _{mean} ($X_{P \cup E}$)
Precision	0.78	0.78	0.80	0.71	0.70	0.73	0.76	0.72	0.76	0.76	0.67	0.69
Recall	0.77	0.77	0.54	0.65	0.73	0.77	0.81	0.69	0.81	0.81	0.59	0.62
F-Measure	0.48	0.48	0.43	0.37	0.45	0.59	0.64	0.52	0.64	0.69	0.58	0.60
STMM	0.76	0.76	0.62	0.64	0.70	0.73	0.77	0.68	0.77	0.77	0.55	0.56

TABLE B.10: Capabilities performance measures on #Micropost2015 Test set of L2A model and baselines.

Entity Type	Baselines		L2A				
	BL-P1	BL-P2	MLP (X_p)	SVM (X_p)	DT (X_p)	KNN (X_p)	SVM _{mean} ($X_{P \cup E}$)
MMCM	2.67	19.00	35.88	25.57	36.07	34.92	49.86
TUCM	15.25	27.29	57.63	42.37	45.76	45.76	46.67
FMCRC	25.96	22.10	26.19	25.00	28.57	40.48	64.84

TABLE B.11: Capabilities performance measures on #Micropost2016 Test set of L2A model and baselines.

Entity Type	Baselines		L2A				
	BL-P1	BL-P2	MLP (X_p)	SVM (X_p)	DT (X_p)	KNN (X_p)	SVM _{mean} ($X_{P \cup E}$)
MMCM	4.67	18.26	32.03	24.00	40.50	39.52	43.24
TUCM	14.68	24.53	53.21	31.19	56.88	52.29	42.86
FMCRC	34.00	32.17	48.67	28.52	57.41	30.56	33.33

TABLE B.12: Accuracy performance of L2A model considering Word Embeddings feature space. For each dataset, the best results are reported in bold.

		2015				2016			
		X_E		$X_{P \cup E}$		X_E		$X_{P \cup E}$	
		Wiki2Vec	GoogleNews	Wiki2Vec	GoogleNews	Wiki2Vec	GoogleNews	Wiki2Vec	GoogleNews
BN	mean	0.64	0.53	0.63	0.55	0.85	0.73	0.85	0.77
	max	0.50	0.51	0.59	0.50	0.77	0.81	0.81	0.85
	min	0.48	0.46	0.49	0.50	0.81	0.81	0.81	0.77
	first	0.51	0.49	0.50	0.52	0.77	0.69	0.77	0.69
DT	mean	0.39	0.41	0.67	0.68	0.62	0.73	0.69	0.62
	max	0.48	0.46	0.63	0.66	0.62	0.77	0.73	0.73
	min	0.45	0.39	0.66	0.67	0.69	0.46	0.85	0.69
	first	0.38	0.41	0.65	0.68	0.65	0.62	0.73	0.69
KNN	mean	0.51	0.53	0.67	0.75	0.85	0.92	0.81	0.88
	max	0.45	0.54	0.56	0.67	0.69	0.88	0.85	0.85
	min	0.45	0.54	0.56	0.66	0.73	0.81	0.85	0.85
	first	0.46	0.49	0.65	0.57	0.58	0.81	0.73	0.81
MLR	mean	0.46	0.42	0.52	0.62	0.92	0.85	0.69	0.77
	max	0.45	0.43	0.53	0.63	0.65	0.77	0.85	0.85
	min	0.55	0.41	0.52	0.59	0.69	0.73	0.77	0.65
	first	0.56	0.43	0.56	0.54	0.88	0.73	0.73	0.69
MLP	mean	0.54	0.57	0.77	0.76	0.92	0.88	0.88	0.85
	max	0.55	0.60	0.75	0.72	0.92	0.85	0.85	0.81
	min	0.52	0.59	0.77	0.75	0.88	0.88	0.88	0.88
	first	0.59	0.55	0.77	0.68	0.85	0.85	0.85	0.85
NB	mean	0.64	0.53	0.63	0.68	0.81	0.77	0.77	0.88
	max	0.50	0.51	0.57	0.53	0.73	0.81	0.85	0.81
	min	0.48	0.46	0.58	0.55	0.77	0.77	0.81	0.85
	first	0.50	0.50	0.49	0.63	0.73	0.69	0.73	0.65
SVM	mean	0.50	0.62	0.75	0.68	0.92	0.85	0.88	0.77
	max	0.49	0.56	0.73	0.77	0.92	0.85	0.88	0.85
	min	0.48	0.54	0.75	0.68	0.92	0.81	0.88	0.85
	first	0.61	0.50	0.75	0.64	0.88	0.88	0.88	0.88

TABLE B.13: Class Wise Accuracy Contribution (%) on #Micropost2015 Dev set considering Word Embeddings feature space (X_E and X_{P-E}).

	MLP						SVM					
	mean		max		min		mean		max		min	
	X_E	$X_{P \cup E}$	X_E	$X_{P \cup E}$	X_E	$X_{P \cup E}$	X_E	$X_{P \cup E}$	X_E	$X_{P \cup E}$	X_E	$X_{P \cup E}$
Character	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Event	0.48	7.21	0.48	6.73	0.48	7.21	0.48	6.73	0.48	6.73	0.48	6.73
Location	16.83	25.96	17.79	24.52	16.83	24.52	15.87	24.52	15.38	24.04	14.90	24.52
Organization	6.73	7.69	7.21	8.65	6.73	8.17	5.77	8.17	5.77	8.65	7.69	8.17
Person	27.40	33.65	26.92	33.17	25.48	34.62	25.48	32.69	27.40	31.25	24.52	33.65
Product	1.92	1.92	1.92	1.92	1.92	1.92	1.92	1.92	1.92	1.92	2.40	1.92
Thing	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48
Overall	53.85	76.92	54.81	75.48	51.92	76.92	50.00	74.52	51.44	73.08	50.48	75.48

TABLE B.14: Class Wise Accuracy Contribution (%) on #Micropost2015 Dev set considering Word Embeddings feature space (X_E and X_{P-E}).

	MLP						SVM					
	mean		max		min		mean		max		min	
	X_E	$X_{P \cup E}$	X_E	$X_{P \cup E}$	X_E	$X_{P \cup E}$	X_E	$X_{P \cup E}$	X_E	$X_{P \cup E}$	X_E	$X_{P \cup E}$
Character	3.85	3.85	3.85	3.85	3.85	3.85	3.85	3.85	3.85	3.85	3.85	3.85
Event	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Location	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08
Organization	11.54	11.54	11.54	11.54	11.54	11.54	11.54	11.54	11.54	11.54	11.54	11.54
Person	50.00	46.15	50.00	42.31	46.15	46.15	50.00	46.15	50.00	46.15	50.00	46.15
Product	3.85	3.85	3.85	3.85	3.85	3.85	3.85	3.85	3.85	3.85	3.85	3.85
Thing	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Overall	92.31	88.46	92.31	84.62	88.46	88.46	92.31	88.46	92.31	88.46	92.31	88.46

TABLE B.15: Class Wise Accuracy Contribution (%) on #Micropost2015 Dev set of L2A model and baselines.

Entity Type	Baselines			L2A								
	BL-D	BL-P1	BL-P2	BN (X_p)	NB (X_p)	MLR (X_p)	MLP (X_p)	SVM (X_p)	DT (X_p)	KNN (X_p)	MLP _{mean} (X_E)	MLP _{mean} ($X_{P \sim E}$)
Character	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Event	0.00	0.00	7.21	2.88	0.00	0.00	0.00	0.00	1.44	1.44	0.48	7.21
Location	24.52	24.52	23.56	25.96	28.37	27.40	28.85	27.40	28.37	28.37	16.83	25.96
Organization	6.73	6.73	5.77	8.17	9.62	9.13	9.13	9.62	8.65	8.65	6.73	7.69
Person	33.17	33.17	24.52	30.77	31.25	32.69	33.17	32.21	30.29	29.81	27.40	33.65
Product	0.96	0.96	0.00	0.48	0.00	0.96	0.96	0.96	0.48	0.48	1.92	1.92
Thing	0.48	0.48	2.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.48	0.48
Overall	65.87	65.87	63.46	68.27	69.23	70.19	72.12	70.19	69.23	68.75	53.85	76.92

TABLE B.16: Class Wise Accuracy Contribution (%) on #Micropost2016 Dev set of L2A model and baselines.

Entity Type	Baselines			L2A								
	BL-D	BL-P1	BL-P2	BN (X_p)	NB (X_p)	MLR (X_p)	MLP (X_p)	SVM (X_p)	DT (X_p)	KNN (X_p)	MLP _{mean} (X_E)	MLP _{mean} ($X_{P \sim E}$)
Character	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.85	3.85
Event	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Location	19.23	19.23	15.38	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08	23.08
Organization	7.69	7.69	7.69	11.54	7.69	7.69	11.54	7.69	11.54	11.54	11.54	11.54
Person	46.15	46.15	26.92	30.77	42.31	42.31	42.31	34.62	42.31	42.31	50.00	46.15
Product	3.85	3.85	3.85	0.00	0.00	3.85	3.85	3.85	3.85	3.85	3.85	3.85
Thing	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Overall	76.92	76.92	53.85	65.38	73.08	76.92	80.77	69.23	80.77	80.77	92.31	88.46

TABLE B.17: Precision, Recall, F-Measure and STMM on #Micropost2015 Dev set considering Word Embeddings feature space (X_E and $X_{P \sim E}$).

	MLP						SVM					
	mean		max		min		mean		max		min	
	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$
Precision	0.77	0.55	0.76	0.54	0.77	0.54	0.52	0.75	0.52	0.75	0.50	0.76
Recall	0.77	0.54	0.75	0.55	0.77	0.52	0.50	0.75	0.51	0.73	0.50	0.75
F-Measure	0.54	0.33	0.53	0.37	0.53	0.33	0.31	0.51	0.32	0.50	0.33	0.51
STMM	0.76	0.52	0.75	0.51	0.76	0.50	0.48	0.74	0.49	0.73	0.48	0.75

TABLE B.18: Precision, Recall, F-Measure and STMM on #Micropost2016 Dev set considering Word Embeddings feature space (X_E and $X_{P \sim E}$).

	MLP						SVM					
	mean		max		min		mean		max		min	
	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$	X_E	$X_{P \sim E}$
Precision	0.92	0.93	0.92	0.93	0.92	0.90	0.93	0.90	0.93	0.91	0.93	0.90
Recall	0.88	0.92	0.85	0.92	0.88	0.88	0.92	0.88	0.92	0.88	0.92	0.88
F-Measure	0.83	0.89	0.64	0.89	0.83	0.86	0.89	0.86	0.89	0.80	0.89	0.86
STMM	0.88	0.91	0.85	0.91	0.88	0.87	0.91	0.87	0.91	0.87	0.91	0.87

TABLE B.19: Precision, Recall, F-Measure and STMM on #Micropost2015 Dev set of L2A model and baselines.

Entity Type	Baselines			L2A								
	BL-D	BL-P1	BL-P2	BN (X_p)	NB (X_p)	MLR (X_p)	MLP (X_p)	SVM (X_p)	DT (X_p)	KNN (X_p)	MLP _{mean} (X_E)	MLP _{mean} ($X_{P \sim E}$)
Precision	0.67	0.67	0.75	0.67	0.67	0.63	0.62	0.63	0.72	0.67	0.52	0.77
Recall	0.66	0.66	0.63	0.68	0.68	0.70	0.72	0.70	0.69	0.69	0.54	0.77
F-Measure	0.37	0.37	0.43	0.37	0.37	0.36	0.37	0.36	0.36	0.36	0.33	0.54
STMM	0.66	0.66	0.68	0.67	0.67	0.66	0.66	0.66	0.66	0.66	0.52	0.76

TABLE B.20: Precision, Recall, F-Measure and STMM on #Micropost2016 Dev set of L2A model and baselines.

Entity Type	Baselines			L2A								
	BL-D	BL-P1	BL-P2	BN (X_p)	NB (X_p)	MLR (X_p)	MLP (X_p)	SVM (X_p)	DT (X_p)	KNN (X_p)	MLP _{mean} (X_E)	MLP _{mean} ($X_{P \sim E}$)
Precision	0.78	0.78	0.80	0.71	0.70	0.73	0.76	0.72	0.76	0.76	0.93	0.92
Recall	0.77	0.77	0.54	0.65	0.73	0.77	0.81	0.69	0.81	0.81	0.92	0.88
F-Measure	0.48	0.48	0.43	0.37	0.45	0.59	0.64	0.52	0.64	0.69	0.89	0.83
STMM	0.76	0.76	0.62	0.64	0.70	0.73	0.77	0.68	0.77	0.77	0.91	0.88

TABLE B.21: Capabilities performance measures on #Micropost2015 Dev set of L2A model and baselines..

Entity Type	Baselines		L2A				
	BL-P1	BL-P2	MLP (X_P)	SVM (X_P)	DT (X_P)	KNN (X_P)	MLP _{mean} ($X_{P \cup E}$)
MMCM	2.67	19.00	35.88	25.57	36.07	34.92	41.51
TUCM	15.25	27.29	57.63	42.37	45.76	45.76	37.50
FMCR	25.96	22.10	26.19	25.00	28.57	40.48	61.11

TABLE B.22: Capabilities performance measures on #Micropost2016 Dev set of L2A model and baselines.

Entity Type	Baselines		L2A				
	BL-P1	BL-P2	MLP (X_P)	SVM (X_P)	DT (X_P)	KNN (X_P)	MLP _{mean} ($X_{P \cup E}$)
MMCM	4.67	18.26	32.03	24.00	40.50	39.52	50.00
TUCM	14.68	24.53	53.21	31.19	56.88	52.29	100.00
FMCR	34.00	32.17	48.67	28.52	57.41	58.94	33.33

Bibliography

- [1] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.
- [2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *CoRR*, abs/1301.3781, 2013.
- [3] Florentina T. Hristea. Statistical Natural Language Processing. In *International Encyclopedia of Statistical Science*, pages 1452–1453. Springer, 2011.
- [4] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2000.
- [5] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 2001.
- [6] Yoav Goldberg. *Neural Network Methods for Natural Language Processing*. Morgan & Claypool Publishers, 2017.
- [7] Federico Alberto Pozzi, Elisabetta Fersini, Enza Messina, and Bing Liu. *Sentiment Analysis in Social Networks*. Morgan Kaufmann, 2016.
- [8] Simon Carter, Wouter Weerkamp, and Manos Tsagkias. Microblog language identification: overcoming the limitations of short, unedited and idiomatic text. *Language Resources and Evaluation*, 47(1):195–215, 2013.
- [9] Francesco Barbieri, Miguel Ballesteros, and Horacio Saggion. Are emojis predictable? In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, 2017.
- [10] Ben Eisner, Tim Rocktäschel, Isabelle Augenstein, Matko Bosnjak, and Sebastian Riedel. emoji2vec: Learning Emoji Representations from their Description. In *Proceedings of the 4th International Workshop on Natural Language Processing for Social Media*, pages 48–54, 2016.

- [11] Francesco Barbieri, Jose Camacho-Collados, Francesco Ronzano, Luis Espinosa Anke, Miguel Ballesteros, Valerio Basile, Viviana Patti, and Horacio Saggion. SemEval 2018 Task 2: Multilingual Emoji Prediction. In *Proceedings of the 12th International Workshop on Semantic Evaluation*, pages 24–33, 2018.
- [12] Hannah Miller, Jacob Thebault-Spieker, Shuo Chang, Isaac Johnson, Loren Terveen, and Brent Hecht. “Blissfully Happy” or “Ready to Fight”: Varying Interpretations of Emoji. In *Proceedings of the 10th International Conference on Web and Social Media*, 2016.
- [13] Debora Nozza, Elisabetta Fersini, and Enza Messina. A Multi-View Sentiment Corpus. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, volume 1, pages 273–280, 2017.
- [14] Francesco Barbieri, Miguel Ballesteros, Francesco Ronzano, and Horacio Saggion. Multimodal Emoji Prediction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2018.
- [15] Andrew Kehoe and Matt Gee. Social Tagging: A new perspective on textual ‘aboutness’. *Studies in Variation, Contacts and Change in English*, 6, 2011.
- [16] Lei Yang, Tao Sun, Ming Zhang, and Qiaozhu Mei. We Know What@ You# Tag: Does the Dual Role Affect Hashtag Adoption? In *Proceedings of the 21st International Conference on World Wide Web*, pages 261–270, 2012.
- [17] Yu-Ru Lin, Drew Margolin, Brian Keegan, Andrea Baronchelli, and David Lazer. #Big-birds Never Die: Understanding Social Dynamics of Emergent Hashtags. In *Proceedings of the 7th International Conference on Weblogs and Social Media*, 2013.
- [18] Ruth E. Page. *Stories and Social Media: Identities and Interaction*. Routledge, 2013.
- [19] Michele Zappavigna. Searchable talk: the linguistic functions of hashtags. *Social Semiotics*, 25(3):274–291, 2015.
- [20] Bing Liu. *Sentiment Analysis and Opinion Mining*. Morgan & Claypool Publishers, 2012.
- [21] Elisabetta Fersini, Enza Messina, and Federico Alberto Pozzi. Expressive signals in social media languages to improve polarity detection. *Information Processing & Management*, 52(1):20 – 35, 2016.
- [22] Philipp Koehn. *Statistical machine translation*. Cambridge University Press, 2009.
- [23] John R. Firth. A synopsis of linguistic theory 1930-55. In *Studies in Linguistic Analysis (special volume of the Philological Society)*, volume 1952-59, pages 1–32. The Philological Society, 1957.

- [24] Zellig Harris. Distributional structure. *Word*, 10(23):146–162, 1954.
- [25] Peter F. Brown, Vincent J. Della Pietra, Peter V. de Souza, Jennifer C. Lai, and Robert L. Mercer. Class-Based n-gram Models of Natural Language. *Computational Linguistics*, 18(4):467–479, 1992.
- [26] Scott Miller, Jethran Guinness, and Alex Zamanian. Name Tagging with Word Clusters and Discriminative Training. In *Proceedings of the 2004 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 4, pages 337–342, 2004.
- [27] Ronan Collobert and Jason Weston. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 160–167, 2008.
- [28] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In *Proceedings of the 27th Annual Conference on Neural Information Processing Systems*, pages 3111–3119, 2013.
- [29] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education Inc., 2003.
- [30] Adrian A. Hopgood. The State of Artificial Intelligence. *Advances in Computers*, 65: 3–77, 2005.
- [31] Aaron Courville Yoshua Bengio and Pascal Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.
- [32] Yoshua Bengio Yann LeCun and Geoffrey E. Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.
- [33] Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.
- [34] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Boosting for Transfer Learning. In *Proceedings of the 24th International Conference on Machine Learning*, pages 193–200, 2007.
- [35] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and Composing Robust Features with Denoising Autoencoders. In *Proceedings of the 25th International Conference on Machine learning*, volume 307, pages 1096–1103, 2008.

- [36] Minmin Chen, Zhixiang Xu, Fei Sha, and Kilian Q. Weinberger. Marginalized Denoising Autoencoders for Domain Adaptation. In *Proceedings of the 29th International Conference on Machine Learning*, pages 767–774, 2012.
- [37] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research*, 11: 3371–3408, 2010.
- [38] Simon Osindero, Geoffrey E. Hinton, and Yee-Whye Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, 2006.
- [39] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy Layer-Wise Training of Deep Networks. In *Proceedings of the 19th Annual Conference on Neural Information Processing Systems*, pages 153–160, 2006.
- [40] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why Does Unsupervised Pre-training Help Deep Learning? *Journal of Machine Learning Research*, 11:625–660, 2010.
- [41] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, 2017.
- [42] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. Pedestrian Detection with Unsupervised Multi-stage Feature Learning. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3626–3633, 2013.
- [43] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.
- [44] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Awni Y. Hannun, Billy Jun, Tony Han, Patrick LeGresley, Xiangang Li, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Sheng Qian, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Chong Wang, Yi Wang, Zhiqian Wang, Bo Xiao, Yan Xie, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 173–182, 2016.

- [45] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Learning Visual Feature Spaces for Robotic Manipulation with Deep Spatial Autoencoders. *CoRR*, abs/1509.06113, 2015.
- [46] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112, 2014.
- [47] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR*, abs/1409.0473, 2014.
- [48] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A CPU and GPU Math Compiler in Python. In *Proceedings of the 9th Python in Science Conference*, pages 1–7, 2010.
- [49] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. In *Deep Learning and Unsupervised Feature Learning Neural Information Processing Systems 2012 Workshop*, 2012.
- [50] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A Matlab-like Environment for Machine Learning. In *BigLearn, Neural Information Processing Systems Workshop*, 2011.
- [51] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*, pages 675–678, 2014.
- [52] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. *CoRR*, abs/1603.04467, 2016.
- [53] Patricia S. Churchland and Terrence J. Sejnowski. *The Computational Brain*. MIT Press, 1992. ISBN 978-0-262-03188-2.

- [54] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.
- [55] Richard H.R. Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and H. Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951, 2000.
- [56] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.
- [57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.
- [58] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In *Proceedings of the 4th International Conference on Learning Representations*, 2016.
- [59] Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating Second-Order Functional Knowledge for Better Option Pricing. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, pages 472–478, 2000.
- [60] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [61] Koby Crammer and Yoram Singer. On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. *Journal of Machine Learning Research*, 2:265–292, 2002.
- [62] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., 1995.
- [63] Solomon Kullback. *Information Theory and Statistics*. Courier Corporation, 1968.
- [64] Andrey Tikhonov. Solution of Incorrectly Formulated Problems and the Regularization Method. *Soviet Mathematics Doklady*, 4:1035–1038, 1963.
- [65] Robert Tibshirani. Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.
- [66] Hui Zou and Trevor Hastie. Regularization and Variable Selection via the Elastic Net. *Journal of the Royal Statistical Society: Series B*, 67(2), 2005.

- [67] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.
- [68] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [69] Paul J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.
- [70] Arthur E. Bryson. A gradient method for optimizing multi-stage allocation processes. In *Proceedings of the Harvard University Symposium on Digital Computers and Their Applications*, page 72, 1961.
- [71] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.
- [72] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.
- [73] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient Back-Prop. In *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 9–48. Springer, 2012.
- [74] Léon Bottou. Stochastic Gradient Descent Tricks. In *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 421–436. Springer, 2012.
- [75] Boris T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.
- [76] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the Importance of Initialization and Momentum in Deep Learning. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1139–1147, 2013.
- [77] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.
- [78] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.
- [79] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-RMSProp, COURSERA: Neural networks for machine learning. *University of Toronto, Technical Report*, 2012.

- [80] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, 2014.
- [81] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.
- [82] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, 2006.
- [83] Ruslan Salakhutdinov and Geoffrey E. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.
- [84] Marc’Aurelio Ranzato and Martin Szummer. Semi-supervised Learning of Compact Document Representations with Deep Networks. In *Proceedings of the 25th International Conference on Machine Learning*, pages 792–799, 2008.
- [85] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach. In *Proceedings of the 28th International Conference on Machine Learning*, pages 513–520, 2011.
- [86] Li Deng and Dong Yu. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.
- [87] Stanley F. Chen and Joshua Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. *Computer Speech & Language*, 13(4):359–393, 1999.
- [88] Joshua T. Goodman. A Bit of Progress in Language Modeling. *Computer Speech & Language*, 15(4):403–434, 2001.
- [89] Frederick Jelinek and Robert Mercer. Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of Workshop on Pattern Recognition in Practice, 1980*, 1980.
- [90] Stanley F. Chen and Joshua Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 310–318, 1996.
- [91] Tomáš Mikolov, Anoop Deoras, Daniel Povey, Lukáš Burget, and Jan Černocký. Strategies for Training Large Scale Neural Network Language Models. In *IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 196–201, 2011.
- [92] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by Latent Semantic Analysis. *Journal of the Association for Information Science*, 41(6):391–407, 1990.

- [93] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent Trends in Deep Learning Based Natural Language Processing. *CoRR*, abs/1708.02709, 2017.
- [94] Frederic Morin and Yoshua Bengio. Hierarchical Probabilistic Neural Network Language Model. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*, 2005.
- [95] Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Conference on Machine Learning*, 2012.
- [96] Marc’Aurelio Ranzato, Christopher S. Poultney, Sumit Chopra, and Yann LeCun. Efficient Learning of Sparse Representations with an Energy-Based Model. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, pages 1137–1144, 2006.
- [97] Marc’Aurelio Ranzato, Y-Lan Boureau, and Yann LeCun. Sparse Feature Learning for Deep Belief Networks. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, pages 1185–1192, 2007.
- [98] Alireza Makhzani and Brendan J. Frey. K-sparse autoencoders. *CoRR*, abs/1312.5663, 2013.
- [99] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [100] Yu Chen and Mohammed J. Zaki. KATE: K-Competitive Autoencoder for Text. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 85–94, 2017.
- [101] Guosong Shao. Understanding the appeal of user-generated media: a uses and gratification perspective. *Internet Research*, 19(1):7–25, 2009.
- [102] Kalina Bontcheva and Dominic Paul Rout. Making sense of social media streams through semantics: A survey. *Semantic Web*, 5(5):373–403, 2014.
- [103] Pierpaolo Basile, Valerio Basile, Malvina Nissim, and Nicole Novielli. Deep Tweets: from Entity Linking to Sentiment Analysis. In *Proceedings of the Italian Computational Linguistics Conference*, 2015.
- [104] Andranik Tumasjan, Timm Oliver Sprenger, Philipp G. Sandner, and Isabell M. Welpe. Predicting Elections with Twitter: What 140 Characters Reveal about Political Sentiment. In *Proceedings of the 4th International Conference on Weblogs and Social Media*, 2010.
- [105] Kate Starbird and Leysia Palen. (How) Will the Revolution be Retweeted? Information Diffusion and the 2011 Egyptian Uprising. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, pages 7–16, 2012.

- [106] Michael J. Paul and Mark Dredze. You Are What You Tweet: Analyzing Twitter for Public Health. In *Proceedings at 5th International AAAI Conference on Weblogs and Social Media*, volume 20, pages 265–272, 2011.
- [107] Antonio Reyes, Paolo Rosso, and Tony Veale. A multidimensional approach for detecting irony in twitter. *Language resources and evaluation*, 47(1):239–268, 2013.
- [108] Jihen Karoui, Farah Benamara, Véronique Moriceau, Nathalie Aussenac-Gilles, and Lamia Hadrich Belguith. Towards a Contextual Pragmatic Model to Detect Irony in Tweets. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 644–650, 2015.
- [109] Alan Ritter, Sam Clark, Mausam, and Oren Etzioni. Named Entity Recognition in Tweets: An Experimental Study. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1524–1534, 2011.
- [110] Giuseppe Rizzo and Raphaël Troncy. NERD: A Framework for Unifying Named Entity Recognition and Disambiguation Extraction Tools. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 73–76, 2012.
- [111] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370, 2005.
- [112] Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. DBpedia Spotlight: Shedding Light on the Web of Documents. In *Proceedings of the 7th International Conference on Semantic Systems*, pages 1–8, 2011.
- [113] Dieter Fensel. Ontologies. In *Ontologies*, pages 11–18. Springer, 2001.
- [114] Lizhen Qu, Gabriela Ferraro, Liyuan Zhou, Weiwei Hou, and Timothy Baldwin. Named Entity Recognition for Novel Types by Transfer Learning. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 899–905, 2016.
- [115] John D. Lafferty, Andrew McCallum, and Fernando C.N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289, 2001.
- [116] Daniel Ramage, David Hall, Ramesh Nallapati, and Christopher D. Manning. Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 248–256, 2009.

- [117] Giuseppe Rizzo, Amparo E. Cano, Bianca Pereira, and Andrea Varga. Making Sense of Microposts (# Microposts2015) Named Entity rEcognition and Linking (NEEL) Challenge. In *Proceedings of the 5th Workshop on Making Sense of Microposts*, pages 44–53, 2015.
- [118] Markus Kroetzsch and Gerhard Weikum. Journal of Web Semantics: Special Issue on Knowledge Graphs. <http://www.websemanticsjournal.org/index.php/ps/announcement/view/19/>, 2015. [Online; accessed May-2018].
- [119] Pikakshi Manchanda, Elisabetta Fersini, Matteo Palmonari, Debora Nozza, and Enza Messina. Towards Adaptation of Named Entity Classification. In *Proceedings of the Symposium on Applied Computing*, pages 155–157, 2017.
- [120] Elisabetta Fersini, Pikakshi Manchanda, Enza Messina, Debora Nozza, and Matteo Palmonari. Adapting Named Entity Types to New Ontologies in a Microblogging Environment. In *Proceedings of the 31st International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems*, 2018. To appear.
- [121] Leon Derczynski, Diana Maynard, Giuseppe Rizzo, Marieke van Erp, Genevieve Gorrell, Raphaël Troncy, Johann Petrak, and Kalina Bontcheva. Analysis of named entity recognition and linking for tweets. *Information Processing & Management*, 51(2):32–49, 2015.
- [122] Cícero Nogueira dos Santos and Maira Gatti. Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts. In *Proceedings of the 25th International Conference on Computational Linguistics*, pages 69–78, 2014.
- [123] Jason Weston, Sumit Chopra, and Keith Adams. #TagSpace: Semantic Embeddings from Hashtags. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1822–1827, 2014.
- [124] Cedric De Boom, Steven Van Canneyt, Thomas Demeester, and Bart Dhoedt. Learning Representations for Tweets through Word Embeddings. In *Proceedings of Benelearn*, 2016.
- [125] Amparo Elizabeth Cano, Daniel Preotiuc-Pietro, Danica Radovanovic, Katrin Weller, and Aba-Sah Dadzie. #Microposts2016: 6th Workshop on Making Sense of Microposts: Big things come in small packages. In *Proceedings of the 25th International Conference on World Wide Web*, pages 1041–1042, 2016.
- [126] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 42–49, 1999.

- [127] Arzucan Özgür, Levent Özgür, and Tunga Güngör. Text categorization with class-based and corpus-based keyword selection. In *Proceedings of the International Symposium on Computer and Information Sciences*, pages 606–615, 2005.
- [128] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 168–175, 2002.
- [129] Hal Daumé III. Frustratingly easy domain adaptation. *arXiv preprint arXiv:0907.1815*, 2009.
- [130] Andrew Arnold, Ramesh Nallapati, and William W. Cohen. Exploiting Feature Hierarchy for Transfer Learning in Named Entity Recognition. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*, pages 245–253, 2008.
- [131] Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Frederick Reiss, and Shivakumar Vaithyanathan. Domain Adaptation of Rule-Based Annotators for Named-Entity Recognition Tasks. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1002–1012, 2010.
- [132] Kai Eckert, Christian Meilicke, and Heiner Stuckenschmidt. Improving Ontology Matching Using Meta-level Learning. In *Proceedings of the 6th European Semantic Web Conference*, pages 158–172, 2009.
- [133] Feng Shi, Juanzi Li, Jie Tang, Guotong Xie, and Hanyu Li. Actively Learning Ontology Matching via User Interaction. In *Proceedings of the 8th International Semantic Web Conference*, pages 585–600, 2009.
- [134] Songyun Duan, Achille Fokoue, and Kavitha Srinivas. One Size Does Not Fit All: Customizing Ontology Alignment Using User Feedback. In *Proceedings of the 9th International Semantic Web Conference*, pages 177–192, 2010.
- [135] Manuel Atencia, Alexander Borgida, Jérôme Euzenat, Chiara Ghidini, and Luciano Serafini. A Formal Semantics for Weighted Ontology Mappings. In *Proceedings of the 11th International Semantic Web Conference*, pages 17–33, 2012.
- [136] Jeffrey Pound, Peter Mika, and Hugo Zaragoza. Ad-hoc Object Retrieval in the Web of Data. In *Proceedings of the 19th International Conference on World Wide Web*, pages 771–780, 2010.
- [137] Lev-Arie Ratinov and Dan Roth. Design Challenges and Misconceptions in Named Entity Recognition. In *Proceedings of the 13th Conference on Computational Natural Language Learning*, pages 147–155, 2009.

- [138] Delip Rao, Paul McNamee, and Mark Dredze. Entity Linking: Finding Extracted Entities in a Knowledge Base. In *Multi-source, Multilingual Information Extraction and Summarization*, pages 93–115. Springer, 2013.
- [139] Leon Derczynski, Diana Maynard, Niraj Aswani, and Kalina Bontcheva. Microblog-Genre Noise and Impact on Semantic Annotation Accuracy. In *Proceedings of the 24th ACM Conference on Hypertext and Social Media*, pages 21–30, 2013.
- [140] Flavio Massimiliano Cecchini, Elisabetta Fersini, Pikakshi Manchanda, Enza Messina, Debora Nozza, Matteo Palmonari, and Cezar Sas. UNIMIB@ NEEL-IT: Named Entity Recognition and Linking of Italian Tweets. In *Proceedings of the 3rd Italian Conference on Computational Linguistics*, 2016.
- [141] idio Ltd. Wiki2Vec. <https://github.com/idio/wiki2vec>, 2014.
- [142] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, and Christian Bizer. DBpedia—A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.
- [143] Pierpaolo Basile, Franco Cutugno, Malvina Nissim, Viviana Patti, and Rachele Sprugnoli. EVALITA 2016: Overview of the 5th Evaluation Campaign of Natural Language Processing and Speech Tools for Italian. *Associazione Italiana di Linguistica Computazionale*, 2016.
- [144] Francesco Piccinno and Paolo Ferragina. From TagME to WAT: a new Entity Annotator. In *Proceedings of the 1st ACM International Workshop on Entity Recognition & Disambiguation*, pages 55–62, 2014.
- [145] Jörg Waitelonis and Harald Sack. Named Entity Linking in #Tweets with KEA. In *Proceedings of the 6th Workshop on Making Sense of Microposts*, 2016.
- [146] Ikuya Yamada, Hideaki Takeda, and Yoshiyasu Takefuji. An End-to-End Entity Linking Approach for Tweets. In *Proceedings of the the 5th Workshop on Making Sense of Microposts*, 2015.
- [147] Anne-Lyse Minard, Mohammed R.H. Qwaider, and Bernardo Magnini. FBK-NLP at NEEL-IT: Active Learning for Domain Adaptation. *Proceedings of the 5th Evaluation Campaign of Natural Language Processing and Speech Tools for Italian*, 2016.
- [148] Vittoria Cozza, Wanda La Bruna, and Tommaso Di Noia. sisinflab: an ensemble of supervised and unsupervised strategies for the NEEL-IT challenge at Evalita 2016. *Proceedings of the 5th Evaluation Campaign of Natural Language Processing and Speech Tools for Italian*, 2016.

- [149] Pierpaolo Basile, Annalina Caputo, Giovanni Semeraro, and Fedelucio Narducci. UNIBA: Exploiting a Distributional Semantic Model for Disambiguating and Linking Entities in Tweets. In *Proceedings of the the 5th Workshop on Making Sense of Microposts*, 2015.
- [150] Kara Greenfield, Rajmonda Caceres, Michael Coury, Kelly Geyer, Youngjune Gwon, Jason Matterer, Alyssa Mensch, Cem Sahin, and Olga Simek. A Reverse Approach to Named Entity Extraction and Linking in Microposts. In *Proceedings of the the 6th Workshop on Making Sense of Microposts*, pages 67–69, 2016.
- [151] Pablo Torres-Tramón, Hugo Hromic, Brian Walsh, Bahareh Rahmanzadeh Heravi, and Conor Hayes. Kanopy4Tweets: Entity Extraction and Linking for Twitter. *Proceedings of the 6th Workshop on Making Sense of Microposts*, 2016.
- [152] Wei Shen, Jianyong Wang, and Jiawei Han. Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460, 2015.
- [153] Ben Hachey, Will Radford, Joel Nothman, Matthew Honnibal, and James R. Curran. Evaluating Entity Linking with Wikipedia. *Artificial intelligence*, 194:130–150, 2013.
- [154] Zhicheng Zheng, Fangtao Li, Minlie Huang, and Xiaoyan Zhu. Learning to Link Entities with Knowledge Base. In *Proceedings of the 2010 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–491, 2010.
- [155] Mark Dredze, Paul McNamee, Delip Rao, Adam Gerber, and Tim Finin. Entity Disambiguation for Knowledge Base Population. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 277–285, 2010.
- [156] Wei Zhang, Jian Su, Chew Lim Tan, and Wen Ting Wang. Entity Linking Leveraging: Automatically Generated Annotation. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1290–1298, 2010.
- [157] Xianpei Han and Jun Zhao. NLPR_KBP in TAC 2009 KBP Track: A Two-Stage Method to Entity Linking. In *Proceedings of the 2nd Text Analysis Conference*, 2009.
- [158] John Lehmann, Sean Monahan, Luke Nezda, Arnold Jung, and Ying Shi. LCC Approaches to Knowledge Base Population at TAC 2010. In *Proceedings of the 3rd Text Analysis Conference*, 2010.
- [159] Sean Monahan, John Lehmann, Timothy Nyberg, Jesse Plymale, and Arnold Jung. Cross-Lingual Cross-Document Coreference with Entity Linking. In *Proceedings of the 4th Text Analysis Conference*, 2011.

- [160] Silviu Cucerzan. Large-Scale Named Entity Disambiguation Based on Wikipedia Data. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 708–716, 2007.
- [161] Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. Robust Disambiguation of Named Entities in Text. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 782–792, 2011.
- [162] Stephen Guo, Ming-Wei Chang, and Emre Kiciman. To Link or Not to Link? A Study on End-to-End Tweet Entity Linking. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1020–1030, 2013.
- [163] Davide Caliano, Elisabetta Fersini, Pikakshi Manchanda, Matteo Palmonari, and Enza Messina. UniMiB: Entity Linking in Tweets using Jaro-Winkler Distance, Popularity and Coherence. *Proceedings of the 6th International Workshop on Making Sense of Microposts*, pages 70–72, 2016.
- [164] Razvan Bunescu and Marius Paşca. Using Encyclopedic Knowledge for Named Entity Disambiguation. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, 2006.
- [165] Paolo Ferragina and Ugo Scaiella. TAGME: On-the-fly Annotation of Short Text Fragments (by Wikipedia Entities). In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, pages 1625–1628, 2010.
- [166] Swapna Gottipati and Jing Jiang. Linking Entities to a Knowledge Base with Query Expansion. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 804–813, 2011.
- [167] Anja Pilz and Gerhard Paaß. From Names to Entities using Thematic Context Distance. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, pages 857–866, 2011.
- [168] Wei Shen, Jianyong Wang, Ping Luo, and Min Wang. LINDEN: Linking Named Entities with Knowledge Base via Semantic Knowledge. In *Proceedings of the 21st International Conference on World Wide Web*, pages 449–458, 2012.
- [169] Wei Shen, Jianyong Wang, Ping Luo, and Min Wang. Linking Named Entities in Tweets with Knowledge Base via User Interest Modeling. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 68–76, 2013.

- [170] Lev Ratinov, Dan Roth, Doug Downey, and Mike Anderson. Local and Global Algorithms for Disambiguation to Wikipedia. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1375–1384, 2011.
- [171] Bernard J. Jansen, Mimi Zhang, Kate Sobel, and Abdur Chowdury. Twitter power: Tweets as electronic word of mouth. *Journal of the Association for Information Science and Technology*, 60(11):2169–2188, 2009.
- [172] Michael Gamon, Anthony Aue, Simon Corston-Oliver, and Eric K. Ringger. Pulse: Mining Customer Opinions from Free Text. In *Advances in Intelligent Data Analysis VI, 6th International Symposium on Intelligent Data Analysis*, volume 3646, pages 121–132, 2005.
- [173] Debora Nozza, Elisabetta Fersini, and Enza Messina. Deep Learning and Ensemble Methods for Domain Adaptation. In *Proceedings of the 28th IEEE International Conference on Tools with Artificial Intelligence*, pages 184–189, 2016.
- [174] Thomas G. Dietterich. Ensemble Learning. *The Handbook of Brain Theory and Neural Networks*, 2:110–125, 2002.
- [175] Elisabetta Fersini, Enza Messina, and Federico Alberto Pozzi. Sentiment analysis: Bayesian ensemble learning. *Decision Support Systems*, 68:26–38, 2014.
- [176] Gang Wang, Jianshan Sun, Jian Ma, Kaiquan Xu, and Jibao Gu. Sentiment classification: The contribution of ensemble learning. *Decision Support Systems*, 57:77–93, 2014.
- [177] Federico Alberto Pozzi, Elisabetta Fersini, and Enza Messina. Bayesian Model Averaging and Model Selection for Polarity Classification. In *Proceedings of the International Conference on Application of Natural Language to Information Systems*, pages 189–200, 2013.
- [178] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
- [179] Yoav Freund and Robert E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.
- [180] Tin Kam Ho. The Random Subspace Method for Constructing Decision Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.
- [181] Josef Kittler, Mohamad Hatef, Robert P.W. Duin, and Jiri Matas. On Combining Classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226–239, 1998.

- [182] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 440–447, 2007.
- [183] George H. John and Pat Langley. Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, pages 338–345, 1995.
- [184] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine learning*, 20(3):273–297, 1995.
- [185] Yoav Freund and Robert E. Schapire. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 37(3):277–296, 1999.
- [186] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.
- [187] Niels Landwehr, Mark Hall, and Eibe Frank. Logistic Model Trees. *Machine Learning*, 59(1-2):161–205, 2005.
- [188] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-Based Learning Algorithms. *Machine Learning*, 6(1):37–66, 1991.
- [189] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- [190] John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 120–128, 2006.
- [191] Steffen Bickel and Tobias Scheffer. Dirichlet-Enhanced Spam Filtering based on Biased Samples. *Proceedings of the 20th Annual Conference on Neural Information Processing Systems*, page 161, 2007.
- [192] Jing Jiang and ChengXiang Zhai. Instance Weighting for Domain Adaptation in NLP. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 264–271, 2007.
- [193] Miroslav Dudík, Steven J. Phillips, and Robert E. Schapire. Correcting sample selection bias in maximum entropy density estimation. In *Advances in Neural Information Processing Systems*, pages 323–330, 2005.
- [194] Hal Daumé III and Daniel Marcu. Domain Adaptation for Statistical Classifiers. *Journal of Artificial Intelligence Research*, pages 101–126, 2006.

- [195] Koby Crammer, Michael Kearns, and Jennifer Wortman. Learning from Multiple Sources. In *Proceedings of the 20th Annual Conference on Neural Information Processing Systems*, page 321, 2007.
- [196] Toshihiro Kamishima, Masahiro Hamasaki, and Shotaro Akaho. TrBagg: A Simple Transfer Learning Method and its Application to Personalization in Collaborative Tagging. In *Proceedings of the 9th IEEE International Conference on Data Mining*, pages 219–228, 2009.
- [197] Cristina Bosco, Viviana Patti, and Andrea Bolioli. Developing Corpora for Sentiment Analysis: The Case of Irony and Senti-TUT. *IEEE Intelligent Systems*, 28(2):55–63, 2013.
- [198] Aniruddha Ghosh, Guofu Li, Tony Veale, Paolo Rosso, Ekaterina Shutova, John Barnden, and Antonio Reyes. SemEval-2015 Task 11: Sentiment Analysis of Figurative Language in Twitter. In *Proceedings of the 9th International Workshop on Semantic Evaluation*, pages 470–478, 2015.
- [199] Herbert Colston and Raymond Gibbs. A Brief History of Irony. In *Irony in Language and Thought: A Cognitive Science Reader*, pages 3–21. Lawrence Erlbaum Assoc Incorporated, 2007.
- [200] Debora Nozza, Elisabetta Fersini, and Enza Messina. Unsupervised Irony Detection: A Probabilistic Model with Word Embeddings. In *Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pages 68–76, 2016.
- [201] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.
- [202] Qiaozhu Mei, Xu Ling, Matthew Wondra, Hang Su, and ChengXiang Zhai. Topic Sentiment Mixture: Modeling Facets and Opinions in Weblogs. In *Proceedings of the 16th International Conference on World Wide Web*, pages 171–180, 2007.
- [203] Chenghua Lin and Yulan He. Joint Sentiment/Topic Model for Sentiment Analysis. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 375–384, 2009.
- [204] Yohan Jo and Alice H. Oh. Aspect and Sentiment Unification Model for Online Review Analysis. In *Proceedings of the 4th ACM International Conference on Web Search and Data Mining*, pages 815–824, 2011.
- [205] Valentin Jijkoun, Maarten de Rijke, and Wouter Weerkamp. Generating Focused Topic-Specific Sentiment Lexicons. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 585–594, 2010.

- [206] Andrea Esuli and Fabrizio Sebastiani. SENTIWORDNET: A Publicly Available Lexical Resource for Opinion Mining. In *Proceedings of the 5th Conference on Language Resources and Evaluation*, pages 417–422, 2006.
- [207] Nobuhiro Kaji and Masaru Kitsuregawa. Building Lexicon for Sentiment Analysis from Massive Collection of HTML Documents. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1075–1083, 2007.
- [208] Saif Mohammad, Cody Dunne, and Bonnie Dorr. Generating High-Coverage Semantic Orientation Lexicons From Overtly Marked Words and a Thesaurus. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 599–608, 2009.
- [209] Delip Rao and Deepak Ravichandran. Semi-Supervised Polarity Lexicon Induction. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 675–682, 2009.
- [210] Yue Lu, Malu Castellanos, Umeshwar Dayal, and ChengXiang Zhai. Automatic Construction of a Context-aware Sentiment Lexicon: An Optimization Approach. In *Proceedings of the 20th International Conference on World Wide Web*, pages 347–356, 2011.
- [211] Elisabetta Fersini, Federico Alberto Pozzi, and Enza Messina. Detecting Irony and Sarcasm in Microblogs: The Role of Expressive Signals and Ensemble Classifiers. In *Proceedings of IEEE International Conference on Data Science and Advanced Analytics*, pages 1–8, 2015.
- [212] Francesco Barbieri and Horacio Saggion. Modelling Irony in Twitter. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 56–64, 2014.
- [213] Irazú Hernández-Farías, José-Miguel Benedí, and Paolo Rosso. Applying Basic Features from Sentiment Analysis for Automatic Irony Detection. In *Pattern Recognition and Image Analysis - 7th Iberian Conference*, pages 337–344, 2015.
- [214] Hernández Farias and Delia Irazu. *Irony and Sarcasm Detection in Twitter: The Role of Affective Content*. PhD thesis, Universitat Politècnica de València, Spain, 2017.
- [215] Delia Irazú Hernández Farías, Viviana Patti, and Paolo Rosso. Irony detection in Twitter: The role of affective content. *ACM Transactions on Internet Technology*, 16(3):19, 2016.
- [216] Saif M. Mohammad and Peter D. Turney. Crowdsourcing a Word–Emotion Association Lexicon. *Computational Intelligence*, 29(3):436–465, 2013.

- [217] Corbett Edward P.J. and Robert Connors. *Classical Rhetoric for the Modern Student*. New York: Oxford University Press, 1971.
- [218] Emilio Sulis, Delia Irazú Hernández Farías, Paolo Rosso, Viviana Patti, and Giancarlo Ruffo. Figurative messages and affect in Twitter: Differences between #irony,#sarcasm and #not. *Knowledge-Based Systems*, 108:132–143, 2016.
- [219] Salvatore Attardo. *Irony*. Elsevier, 2006.
- [220] Marta Dynel. Linguistic approaches to (non) humorous irony. *Humor*, 27(4):537–550, 2014.
- [221] David C. Littman and Jacob L. Mey. The nature of irony: Toward a computational model of irony. *Journal of Pragmatics*, 15(2):131–151, 1991.
- [222] Rachel Giora, Elad Livnat, Ofer Fein, Anat Barnea, Rakefet Zeiman, and Iddo Berger. Negation Generates Nonliteral Interpretations by Default. *Metaphor and Symbol*, 28(2): 89–115, 2013.
- [223] Rachel Giora, Shir Givoni, and Ofer Fein. Defaultness Reigns: The Case of Sarcasm. *Metaphor and Symbol*, 30(4):290–313, 2015.
- [224] Rachel Giora, Ari Drucker, Ofer Fein, and Itamar Mendelson. Default Sarcastic Interpretations: On the Priority of Nonsalient Interpretations. *Discourse Processes*, 52(3): 173–200, 2015.
- [225] Po-Ya Angela Wang. # Irony or# Sarcasm—A Quantitative and Qualitative Study Based on Twitter. In *Proceedings of the 27th Pacific Asia Conference on Language, Information, and Computation*, pages 349–356, 2013.
- [226] Francesco Barbieri, Horacio Saggion, and Francesco Ronzano. Modelling Sarcasm in Twitter, a Novel Approach. In *Proceedings of the 5th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 50–58, 2014.
- [227] Raymond W. Gibbs and Herbert L. Colston. *Irony in language and thought: A cognitive science reader*. Psychology Press, 2007.
- [228] Helga Kotthoff. Gender and joking: On the complexities of women’s image politics in humorous narratives. *Journal of Pragmatics*, 32(1):55–80, 2000.
- [229] Skye McDonald. Exploring the Process of Inference Generation in Sarcasm: A Review of Normal and Clinical Studies. *Brain and Language*, 68(3):486–506, 1999.
- [230] Leila Weitzel, Ronaldo Cristiano Prati, and Raul Freire Aguiar. The Comprehension of Figurative Language: What Is the Influence of Irony and Sarcasm on NLP Techniques?

- In *Sentiment Analysis and Ontology Engineering: An Environment of Computational Intelligence*, pages 49–74. Springer, 2016.
- [231] Antonio Reyes and Paolo Rosso. On the difficulty of automatically detecting irony: beyond a simple case of negation. *Knowledge and Information Systems*, 40(3):595–614, 2014.
- [232] Dmitry Davidov, Oren Tsur, and Ari Rappoport. Semi-supervised recognition of sarcastic sentences in twitter and amazon. In *Proceedings of the 14th Conference on Computational Natural Language Learning*, pages 107–116. Association for Computational Linguistics, 2010.
- [233] Roberto González-Ibáñez, Smaranda Muresan, and Nina Wacholder. Identifying Sarcasm in Twitter: A Closer Look. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Short Papers*, pages 581–586, 2011.
- [234] Ellen Riloff, Ashequl Qadir, Prafulla Surve, Lalindra De Silva, Nathan Gilbert, and Ruihong Huang. Sarcasm as Contrast between a Positive Sentiment and Negative Situation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 704–714, 2013.
- [235] Tomáš Ptáček, Ivan Habernal, and Jun Hong. Sarcasm Detection on Czech and English Twitter. In *Proceedings of the 25th International Conference on Computational Linguistics: Technical Papers*, pages 213–223, 2014.
- [236] Ted Pedersen, Siddharth Patwardhan, and Jason Michelizzi. WordNet::Similarity - Measuring the Relatedness of Concepts. In *Demonstration Papers at the 2004 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 38–41, 2004.
- [237] David Bamman and Noah A. Smith. Contextualized sarcasm detection on Twitter. In *Proceedings of the 9th International AAAI Conference on Web and Social Media*, pages 574–77, 2015.
- [238] Ashwin Rajadesingan, Reza Zafarani, and Huan Liu. Sarcasm Detection on Twitter: A Behavioral Modeling Approach. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining*, pages 97–106, 2015.
- [239] Debanjan Ghosh, Weiwei Guo, and Smaranda Muresan. Sarcastic or Not: Word Embeddings to Predict the Literal or Sarcastic Meaning of Words. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1003–1012, 2015.

- [240] Sam Ransbotham, Gerald C. Kane, and Nicholas H. Lurie. Network Characteristics and the Value of Collaborative User-Generated Content. *Marketing Science*, 31(3):387–405, 2012.
- [241] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C. Aggarwal, and Thomas S. Huang. Heterogeneous Network Embedding via Deep Architectures. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 119–128, 2015.
- [242] Palash Goyal and Emilio Ferrara. Graph Embedding Techniques, Applications, and Performance: A Survey. *arXiv preprint arXiv:1705.02801*, 2017.
- [243] Mikhail Belkin and Partha Niyogi. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation*, 15(6):1373–1396, 2003.
- [244] Sam T. Roweis and Lawrence K. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, 290(5500):2323–2326, 2000.
- [245] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed Large-scale Natural Graph Factorization. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 37–48, 2013.
- [246] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale Information Network Embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077, 2015.
- [247] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric Transitivity Preserving Graph Embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1105–1114, 2016.
- [248] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 701–710, 2014.
- [249] Aditya Grover and Jure Leskovec. Node2Vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864, 2016.
- [250] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural Deep Network Embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1225–1234, 2016.
- [251] Xiao Huang, Jundong Li, and Xia Hu. Accelerated Attributed Network Embedding. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 633–641, 2017.

- [252] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. Tri-Party Deep Network Representation. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pages 1895–1901, 2016.
- [253] Quoc V. Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. In *Proceedings of the 31th International Conference on Machine Learning*, pages 1188–1196, 2014.
- [254] Elisabetta Fersini, Federico Alberto Pozzi, and Enza Messina. Approval network: a novel approach for sentiment analysis in social networks. *World Wide Web*, 20(4):831–854, 2017.
- [255] Zellig S. Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.
- [256] Emily M. Jin, Michelle Girvan, and Mark E.J. Newman. The structure of growing social networks. *Physical review E*, 64(4):046132, 2001.
- [257] Sophia R. Goldberg, Hannah Anthony, and Tim S. Evans. Modelling citation networks. *Scientometrics*, 105(3):1577–1604, 2015.
- [258] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. ArnetMiner: Extraction and Mining of an Academic Social Network. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 990–998, 2008.
- [259] Kar Wai Lim and Wray L. Buntine. Bibliographic Analysis with the Citation Network Topic Model. In *Asian Conference on Machine Learning*, pages 142–158, 2015.
- [260] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective Classification in Network Data. *AI Magazine*, 29(3):93–106, 2008.
- [261] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for Hyperparameter Optimization. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011*, pages 2546–2554, 2011.
- [262] James Bergstra, Daniel Yamins, and David D. Cox. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 115–123, 2013.
- [263] Jui-Ting Huang, Jinyu Li, Dong Yu, Li Deng, and Yifan Gong. Cross-Language Knowledge Transfer using Multilingual Deep Neural Network with Shared Hidden Layers. In

- Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7304–7308, 2013.
- [264] Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. Multi-Way, Multilingual Neural Machine Translation with a Shared Attention Mechanism. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 866–875, 2016.
- [265] Eytan Adar and Christopher Ré. Managing Uncertainty in Social Networks. *IEEE Data Engineering Bulletin*, 30(2):15–22, 2007.
- [266] Arijit Khan and Lei Chen. On Uncertain Graphs Modeling and Queries. *Proceedings of the VLDB Endowment*, 8(12):2042–2043, 2015.
- [267] Muccheol Kim and Sangyong Han. Cognitive social network analysis for supporting the reliable decision-making process. *The Journal of Supercomputing*, 2016.
- [268] Salma Ben Dhaou, Kuang Zhou, Mouloud Kharoune, Arnaud Martin, and Boutheina Ben Yaghlane. The advantage of evidential attributes in social networks. In *Proceedings of the 20th International Conference on Information Fusion*, pages 1–8, 2017.
- [269] Kaiquan Xu, Jiexun Li, and Stephen Shaoyi Liao. Sentiment Community Detection in Social Networks. In *Proceedings of the 2011 iConference*, pages 804–805, 2011.
- [270] William Deitrick and Wei Hu. Mutually Enhancing Community Detection and Sentiment Analysis on Twitter Networks. *Journal of Data Analysis and Information Processing*, 1(03):19, 2013.
- [271] Mirko Lai, Marcella Tambuscio, Viviana Patti, Giancarlo Ruffo, and Paolo Rosso. Extracting Graph Topological Information and Users’ Opinion. In *Proceedings of the International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 112–118, 2017.
- [272] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. DynGEM: Deep Embedding Method for Dynamic Graphs. *CoRR*, abs/1805.11273, 2018.