DEGREE PROJECT IN INFORMATION AND COMMUNICATION TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, *SWEDEN 2018*

Deep Text Mining of Instagram Data Without Strong Supervision

KIM HAMMAR

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

KTH Information and
Communication Technology

Deep Text Mining of Instagram Data Without Strong Supervision

KIM HAMMAR

Master's Thesis at KTH Information and Communication Technology
Dept. of Software and Computer Systems
Supervisor: Shatha Jaradat
Examiner: Mihhail Matskin

TRITA-EECS-EX-2018:138

Abstract

With the advent of social media, our online feeds increasingly consist of short, informal, and unstructured text. This data can be analyzed for the purpose of improving user recommendations and detecting trends. The grand volume of unstructured text that is available makes the intersection of text processing and machine learning a promising avenue of research. Current methods that use machine learning for text processing are in many cases dependent on annotated training data. However, considering the heterogeneity and variability of social media, obtaining strong supervision for social media data is in practice both difficult and expensive. In light of this limitation, a belief that has put its marks on this thesis is that the study of text mining methods that can be applied without strong supervision is of a higher practical interest.

This thesis investigates unsupervised methods for scalable processing of text from social media. Particularly, the thesis targets a classification and extraction task in the fashion domain on the image-sharing platform Instagram. Instagram is one of the largest social media platforms, containing both text and images. Still, research on text processing in social media is to a large extent limited to Twitter data, and little attention has been paid to text mining of Instagram data. The aim of this thesis is to broaden the scope of state-of-the-art methods for information extraction and text classification to the unsupervised setting, working with informal text on Instagram. Its main contributions are (1) an empirical study of text from Instagram; (2) an evaluation of word embeddings for Instagram text; (3) a distributed implementation of the FastText algorithm; (4) a system for fashion attribute extraction in Instagram using word embeddings; and (5) a multi-label clothing classifier for Instagram text, built with deep learning techniques and minimal supervision.

The empirical study demonstrates that the text distribution on Instagram exhibits the long-tail phenomenon, that the text is just as noisy as have been reported in studies on Twitter text, and that comment sections are multi-lingual. In experiments with word embeddings for Instagram, the importance of hyperparameter tuning is manifested and a mismatch between pre-trained embeddings and social media is observed. Furthermore, that word embeddings are a useful asset for information extraction is confirmed. Experimental results show that word embeddings beats a baseline that uses Levenshtein distance on the task of extracting fashion attributes from Instagram. The results also show that the distributed implementation of FastText reduces the time it takes to train word embeddings with a factor that scales with the number of machines used for training. Finally, our research demonstrates that weak supervision can be used to train a deep classifier, achieving an F_1 score of 0.61 on the task of classifying clothes in Instagram posts based only on the associated text, which is on par with human performance.

Keywords— Natural Language Processing, Information Extraction, Machine Learning

Referat

I och med uppkomsten av sociala medier så består våra online-flöden till stor del av korta och informella textmeddelanden, denna data kan analyseras med syftet att upptäcka trender och ge användarrekommendationer. Med tanke på den stora volymen av ostrukturerad text som finns tillgänglig så är kombinationen av språkteknologi och maskinlärning ett forskningsområde med stor potential. Nuvarande maskinlärningsteknologier för textbearbetning är i många fall beroende av annoterad data för träning. I praktiken så är det dock både komplicerat och dyrt att anskaffa annoterad data av hög kvalitet, inte minst vad gäller data från sociala medier, med tanke på hur pass föränderlig och heterogen sociala medier är som datakälla. En övertygelse som genomsyrar denna avhandling är att textutvinnings metoder som inte kräver precis övervakning har större potential i praktiken.

Denna avhandling undersöker oövervakade metoder för skalbar bearbetning av text från sociala medier. Specifikt så täcker avhandlingen ett komplext klassifikations- och extraktions- problem inom modebranschen på bilddelningsplattformen Instagram. Instagram hör till de mest populära sociala plattformarna och innehåller både bilder och text. Trots det så är forskning inom textutvinning från sociala medier till stor del begränsad till data från Twitter och inte mycket uppmärksamhet har givits de stora möjligheterna med textutvinning från Instagram. Ändamålet med avhandlingen är att förbättra nuvarande metoder som används inom textklassificering och informationsextraktion, samt göra dem applicerbara för oövervakad maskinlärning på informell text från Instagram. De primära forskningsbidragen i denna avhandling är (1) en empirisk studie av text från Instagram; (2) en utvärdering av ord-vektorer för användning med text från Instagram; (3) en distribuerad implementation av FastText algoritmen; (4) ett system för extraktion av kläddetaljer från Instagram som använder ord-vektorer; och (5) en flerkategorisk kläd-klassificerare för text från Instagram, utvecklad med djupinlärning och minimal övervakning.

Den empiriska studien visar att textdistributionen på Instagram har en lång svans, att texten är lika informell som tidigare rapporterats från studier på Twitter, samt att kommentarssektionerna är flerspråkiga. Experiment med ord-vektorer för Instagram understryker vikten av att justera parametrar före träningsprocessen, istället för att använda förbestämda värden. Dessutom visas att ord-vektorer tränade på formell text är missanpassade för applikationer som bearbetar informell text. Vidare så påvisas att ord-vektorer är effektivt för informationsextraktion i sociala medier, överlägsen ett standardvärde framtaget med informationsextraktion baserat på syntaktiskt ordlikhet. Resultaten visar även att den distribuerade implementationen av FastText kan minska tiden det tar att träna ord-vektorer med en faktor som beror på antalet maskiner som används i träningen. Slutligen, vår forskning indikerar att svag övervakning kan användas för att träna en klassificerare med djupinlärning. Den tränade klassificeraren uppnår ett F_1 resultat av 0.61 på uppgiften att klassificera kläddetaljer av bilder från Instagram, baserat endast på bildtexten och tillhörande användarkommentarer, vilket är i nivå med mänsklig förmåga.

Nyckelord— Språkteknologi, Informationsextraktion, Maskinlärning

Acknowledgments

I would like to thank my supervisor Shatha Jaradat for giving me the opportunity to join her team and get to know her research ideas. This thesis would not have been the same without her support and guidance. Shatha, you have provided me with a work environment that has been inspiring and I have learned a lot of from you!

In my research I have had the fortune to be working with Prof. Mihhail Matskin (Misha!), who I have a lot to thank for. He gave us valuable counseling in this project and provided his expertise to improve my work. Thank you!

Nima Dokoohaki, Ph.D, offered his precious time to proofread my work and give advice. Nima has been present from the beginning of this project and played an important role in the making of this thesis, for which I am grateful. Thanks for the great discussions, Nima!

Moreover, I wish to thank my fellow students and colleagues in this project, Ummul Wara and Mallu Goswami, whom it was a pleasure to collaborate with. To my recurring lab partner during these two years, Konstantin Sozinov, thanks for all the great projects and knowledge exchange!

I am thankful to the organizations behind Hops Hadoop[1] for supporting this research with computational resources.

Finally, I want to thank my family and friends. Especially my parents and siblings, who always supports me in life. To my mother who encouraged me to commence this study program, and to my father who will always be with me.

Stockholm, June 2018
Kim Hammar

[1] `hops.site`

Contents

List of Figures

List of Tables

Acronyms

API Application Programming Interface. 11, 25, 56, 64

BSP Bulk Synchronous Parallel. 37

CBOW Continuous Bag-of-Words. 13–15, 44, 60

CNN Convolutional Neural Network. 2, 16, 17, 20, 49, 52–55, 63

LSA Latent Semantic Analysis. 13, 15

NER Named-Entity Recognition. 10, 22

NLP Natural Language Processing. 1, 2, 4, 5, 7–10, 12, 13, 16, 17, 25, 56, 62

OOV Out-Of-Vocabulary. 15, 28

PCA Principal Component Analysis. 42

POS Part-Of-Speech. 8

ReLU Rectified Linear Unit. 50, 53, 54

RNN Recurrent Neural Network. 16, 17

Skip-gram Continuous Skip-gram. 13–15, 33–35, 44, 60

SVM Support Vector Machine. 11, 20

TF-IDF Term Frequency-Inverse Document Frequency. 11, 40

Chapter 1

Introduction

This thesis presents methods for accurate and scalable text mining in social media without access to annotated training data. In this introductory chapter, I motivate the research, introduce the area under study, and provide a roadmap for the remainder of the thesis.

1.1 Motivation

Text mining is the means of extracting meaningful information and detecting patterns in unstructured text, often using a synthesis of linguistics, Natural Language Processing (NLP), and machine learning. Currently, more data are being generated than ever before, causing a growing need for intelligent ways to process text. One of the data types that has enlarged the most in the present time is unstructured text, stemming from social media. While social media fosters the development of a new type of applications, it also brings with it its own set of challenges due to the informal language. The informal language in social media has made many established NLP techniques obsolete. Following the increased data volumes and velocity of its generation, as well as the informal language in social media, text processing systems are subject to new requirements.

Despite the tremendous complexity and variety of natural language, many facets of it can be captured by formal rules. One can say that natural language *"makes infinite use of finite means"* (Humboldt 1836). However, although many aspects can be covered by rules, it is debatable whether the natural languages we use today ever can be described in its entirety with rules, it has at least proven to be extremely difficult (Shieber 1987). Considering that the language used in social media is even more complex than formal languages (Baldwin et al. 2013), this thesis investigates a different approach. I examine data-driven methods to learn implicit knowledge about text corpora, studying the *intersection* between NLP and machine learning.

A frequent machine learning model for NLP tasks is the neural network. Neural networks are distributed processing systems where a collection of connected processing units are trained jointly to learn some task. In this context, knowledge is

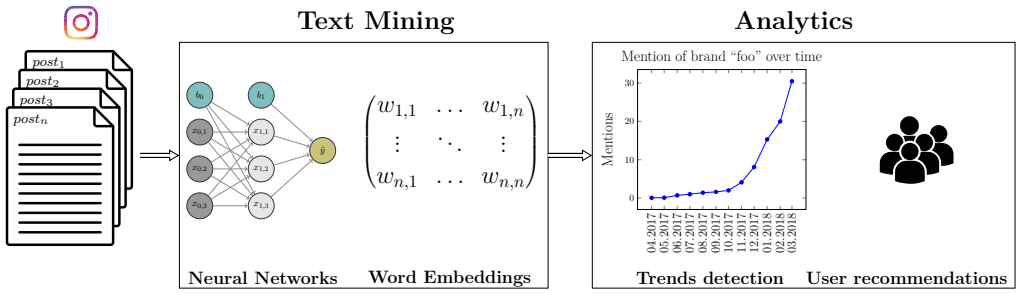

Figure 1.1: An overview of the research problem.

not localized or explicit, rather it consists of the connections among the units in the network (D. E. Rumelhart, McClelland, and PDP Research Group 1986). In this thesis, I experiment with multiple neural network models for language understanding. Such as, networks for learning word embeddings that can capture the semantics of words using the *distributional hypothesis* (Harris 1954), and deep Convolutional Neural Network (CNN)s (Lecun et al. 1998) for text classification.

Several success stories from research on the intersection of NLP and machine learning follow the supervised learning paradigm, and require large amounts of manually annotated data, a requirement which is not always satisfiable. Considering the vast amount of unlabeled data that is accessible today, there is a profound incentive for studying models in the paradigms of *unsupervised* and *weakly-supervised* learning. In this thesis, I explore the boundaries of machine learning models that can be used for text mining *without* requiring annotated training data.

Instagram[1] is one of the largest social media platforms and is redefining many consumption-driven industries by constituting as a new platform for marketing, for trends detection, and for user recommendations (Berthon et al. 2012). In our research, we develop methods for extracting fashion details from text on Instagram and classifying Instagram posts based on fashion attributes. This technology can enable a new type of user recommendation in the fashion domain (Fig. 1.1[2]). Existing methods are sub-optimal for our use-case by often requiring annotated training data (Severyn and Moschitti 2015), not scaling to large data volumes (Bojanowski et al. 2016), not being compatible with social media text (Vine et al. 2015), or not being optimized for the Instagram domain (Ritter, Clark, et al. 2011).

The work presented in this thesis is part of a larger research project for elaborating the state-of-the-art in fashion recommendation (Jaradat 2017). The text processing methods presented in this thesis are meant to be integrated with computer vision models in the project.

We believe that there is a value in the text on Instagram that currently is unutilized. We consider the text that is resident on Instagram as an opportunity to apply text mining methods to and extract information that can be used for

[1] `Instagram.com`

[2] The Instagram logo is a registered trademark of Instagram.

predictive modeling and analytics. The thesis' contribution includes:

- An empirical study of Instagram text.
- An evaluation of word embeddings for Instagram text.
- A system for scalable training of word embeddings with the FastText algorithm.
- A system for unsupervised extraction of fashion attributes from Instagram text.
- A novel pipeline for multi-label clothing classification of the text associated with Instagram posts using weak supervision and the *data programming* paradigm (A. J. Ratner et al. 2016).

The empirical study provides one of the few available studies on Instagram text. Furthermore, although I apply established algorithms for training word embeddings, I provide the first distributed variant of one of the algorithms, together with benchmarks demonstrating its scalability. Moreover, by using extensive quantities of unlabeled data, domain knowledge, and weak supervision, I propose a solution to classification and extraction problems working with Instagram text, that does not require hand-labeled data.

1.2 Background

Just as other consumption-driven industries, the fashion industry has been influenced by the emergence of social media. Social media is progressively getting more attention by fashion brands and retailers as a source for detecting trends, adapting user recommendations, and for marketing purposes (Berthon et al. 2012). To give an example, the image-sharing platform Instagram, with over 700M users (Instagram 2017), has become a popular medium for fashion branding and community engagement (Alter 2016). Consequently, fashion attribute extraction and classification on Instagram is an important task for several modern applications working with user recommendation and detection of fashion trends.

Although characterized as an image-sharing platform, Instagram enclose large volumes of user generated text as well. Specifically, an Instagram post can be associated with an image caption written by the author of the post, by comments written by other users, and by "tags" in the image that refer to other users. Despite being a platform rich of text, little prior work has paid attention to the promising applications of text mining with Instagram data.

In the context of fashion recommendation, we did a case study on Instagram posts in the fashion community. In the case study, we found that clues about clothing attributes on an image can be found in the associated text, an example of this is given in Fig. 1.2. We observed that the text can include both information

Figure 1.2: An Instagram post from our dataset.

about obvious image attributes such as "I love that *coat*", and more nuanced image details that are harder to infer from the image alone, such as "this coat is from *H&M*".

1.3 Problem

Text in social media is unstructured and has a more informal and conversational tone than text from conventional media outlets. The informal text reduces the effectiveness of traditional text processing tools. In addition, Instagram as a source of social media text is relatively unexplored. Moreover, using methods that rely on strong supervision narrows the set of possible application domains.

This thesis addresses the problem of extracting knowledge and learning from text in Instagram using methods that do not have the prerequisite of strong supervision. *How can we perform accurate and scalable text mining of Instagram data without strong supervision?*

1.4 Purpose

The purpose of this research is to to bridge the gap between current solutions for text mining and domains where annotated training data are not available. Additionally, the project aspires to advance the current NLP methods for processing *informal* text in a *scalable* manner, both in terms of computation and data.

1.5 Goal

The goal of this thesis is to present methods for text mining in social media without strong supervision. This includes an assessment of current solutions, explaining

specific requirements that arise in social media, and presenting potential resolutions that are evaluated in terms of accuracy and scalability.

1.5.1 Ethics and Sustainability

This research supports a sustainable development by enabling a new type of applications based on social media activities, which benefits several elements in society. Text classification of users' social media behavior can be used by recommendation systems to enhance recommendations, which is of interest both for users and companies. Moreover, applications with intelligent text processing capabilities have promising applications to make web contents more accessible. For instance, automatic text classification can improve the web experience for users with disabilities and can remove language barriers.

This thesis is a transparent knowledge contribution of possible ways that an organization can make use of user generated text. An ethical prerequisite for processing user generated text is that the processing comply with privacy laws and respects the integrity of users. The data used to produce results presented in this thesis are non-confidential and processed solely for scientific purposes. The data collection was external to my work and is out of scope of this thesis.

1.6 Research Methodology

The methodology is of an experimental nature. The research method consists of quantitative evaluations and comparisons with state-of-the-art solutions. As no established theory exists on the topic of our work, the conducted research is inductive, with the goal of providing new theories and solutions within our field of research.

1.7 Outline

The remainder of this thesis is structured as follows. Chapter 2 describes the relevant theory in the field of NLP and machine learning, as well as prior work that is related to our research. Chapter 3 summarizes the research strategy and the datasets that have been used. Next, my research on an end-to-end solution for information extraction and text classification can be divided into two parts with respective evaluations. The first part surveys information extraction and feature learning of unlabeled data, and is presented in Chapter 4. The second part, that is presented in Chapter 5, builds upon the methods for information extraction to train a deep text classifier using weak supervision. Lastly, Chapter 6 contains the author's interpretation and conclusions from the results, as well as suggestions for future research directions.

Replication and Open Source The code used for experiments in this thesis is open source, and all datasets for which publication is not prohibited for privacy reasons, are publicly available[3].

[3] `https://github.com/shatha2014/FashionRec`

Chapter 2

Natural Language Understanding in Social Media

Considering that social media is an important medium for modern communication, the call for methods to process text in social media is high. Text from social media generally does not adhere to standard grammar rules, making linguistic methods fragile. For this reason, it is common to instead turn to statistical methods for understanding text from social media.

This chapter covers background theory and related work from the two interdisciplinary fields of text processing and machine learning. In Section 2.1, I introduce the field of NLP from our research perspective, the challenges with text in social media, and a repertoire of available methods for text processing. Moreover, an outline of methods for learning from text is covered in Section 2.2, including foundational techniques that our research builds upon, such as word embeddings and models for text classification.

2.1 Natural Language Processing

When computer systems are built for processing with natural language in mind, taking into account language specific properties, such as tokenization and capitalization, they are referred to as NLP systems. NLP is related to the study of language (Section 2.1.1 and Section 2.1.4), as well as challenges faced when processing text, such as text normalization (Section 2.1.2), and downstream applications, like information extraction (Section 2.1.3). Historically, NLP has been divided into *symbolic NLP*, inspired from symbolic artificial intelligence, and *statistical NLP* (Jurafsky and Martin 2000). This thesis focuses on statistical NLP, with the goal of exploiting massive data volumes to build intelligent text processing systems.

2.1.1 Linguistics

The study of language can be divided into six linguistic categories: phonology, morphology, syntax, lexical semantics, pragmatics, and discourse (Jurafsky and Martin 2000). Where morphology, syntax, and lexical semantics, are directly pertinent to our processing task . For instance, tokenization and processing of individual words is linked to morphology, that models the structure of words. As an example of morphological parsing, the word "skirts" consists of the root morpheme "skirt" and the inflectional morpheme "-s". In the same manner, extracting *meaning* from text is related to syntax and lexical semantics, that models word order and the semantics of words.

Typically, NLP systems are composed of multiple processing stages, where the purpose of initial processing stages is to improve the performance of downstream tasks. Text processing methods in the early stages, such as, tokenization, lemmatization, syntactic annotation, and Part-Of-Speech (POS) tagging, make use of linguistic properties of the text. For instance, POS tagging is a standardized way to categorize words based on their role in a language, and syntactic annotation is generally enabled by linguistic grammars. Grammars are used to express syntactic knowledge about text, such as rules of how words are grouped together and how valid sequences of words can be formed. Once a language is described in grammars, it can be used to *parse* text to understand its structure, and to *generate* text that is valid according to the grammar.

Context-free grammar is a particular formalism to express grammars and consists of a four-tuple $\langle N, \Sigma, P, S \rangle$, where N denotes a set of non-terminal symbols, Σ denotes a set of terminal symbols, P denotes a set of productions (rules), and S is the start symbol (ibid.). Rules can constitute both as a lexicon to constrain the set of valid words in a language, and as a way to define constituent structure of phrases, such as how words can be grouped together. Some examples are given in Eq. (2.1), where S is short for "sentence", NP is short for "noun phrase", VP is short for "verb phrase", and "|" can be read as "or". Terminal symbols refer to simple words, like *hello* in the example given in Eq. (2.1). Non-terminals refer to generalizations or clusters of other symbols, such as NP that is a generalization for noun phrases. A context-free rule always has a non-terminal to the left of the arrow symbol ($\rightarrow$) and an ordered list of one or more terminals and non terminals to the right (ibid.).

$$\begin{aligned} S &\rightarrow \quad NP \quad VP \\ Noun &\rightarrow \quad hello \quad | \quad world \end{aligned} \tag{2.1}$$

Grammar rules can be hierarchical and defined at different levels. For instance, rules can define how valid phrases can be formed, and how sentences can be constructed from valid phrases. Alternative grammar formalisms to context-free grammars are dependency grammars and probabilistic context-free grammars.

Even though solutions for ubiquitous text processing tasks, such as POS-tagging, are mature at this stage, the advent of social media has turned things around. Traditional NLP tools are designed for newswire text and experiments demonstrate

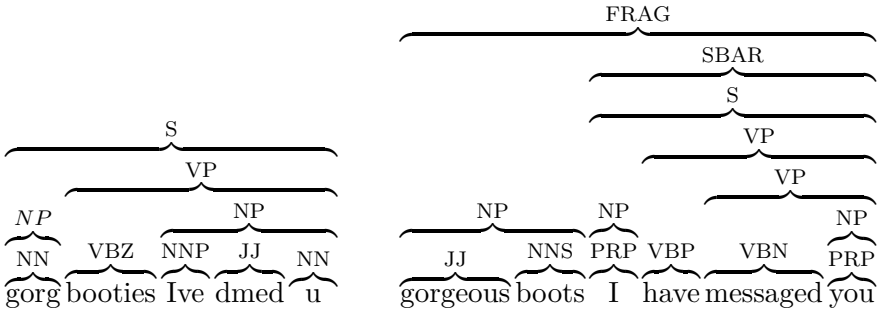

Figure 2.1: Comparison between parse trees for informal and formal text using standard NLP tools.

their inferior performance on social media text (Finin et al. 2010; Ritter, Clark, et al. 2011). This is why recent research efforts have tried to adapt NLP tools to the social media domain (Gimpel et al. 2011; Derczynski et al. 2013). To demonstrate the worsened performance of many NLP tools for social media text, I applied the Stanford parser (Klein and Christopher D Manning 2003) to a typical user comment from Instagram and compared the detected parse tree with the parse tree for the formal spelling of the comment. The result is shown in Fig. 2.1.1, notable is that with the original and informal spelling, the parser erroneously detected "gorg" as a noun (NN), "booties" as a verb (VBZ), and "dmed" as an adjective (JJ).

In addition to studying properties of text at word and phrase level, text can be studied on the level of corpora. Perhaps the most famous result in linguistic statistics is *Zipf's law* (Eq. (2.2)), that stems from a study on relative word frequency in text corpora (Zipf 1949).

$$f(w) \propto \frac{1}{r(w)} \tag{2.2}$$

Zipf's law describes the relationship between word frequency $f(w)$ and word rank $r(w)$, where the rank of a word refers to its relative frequency compared to other words in the corpora. The law states that the word frequency is inversely proportional to the word's rank. Although initially thought as a law for linguistics, time has revealed that Zipf's law is present in a plethora of domains, and belongs to a larger family of power law probability distributions.

2.1.2 Text Normalization

Text normalization is a collective name for methods to prepare text for processing. Text normalization is generally the first task in text processing systems, and is not an objective in itself, rather its purpose is to improve the effectiveness of downstream processing tasks. Common text normalization methods are (1) lemmatization, that converts words to their lemma, with the purpose of reducing morphological variation; (2) stemming, a variant of lemmatization that converts words to

their word stem; (3) stopword removal, in large corpora, the most frequent words can dominate the corpora statistics, reducing the meaning of statistics such as word co-occurrence, since it may appear that every word co-occur with only stopwords; (4) tokenization, that is the means of dividing text into tokens; (5) clustering words and grouping similar words together, with the purpose of reducing lexical variation; (6) spell-checking, the procedure of correcting miss-spelled words; (7) filtering and substitution, where tokens that are of no value for later analysis are removed or substituted, and (8) word segmentation, the task of dividing a string of composite text into a list of words.

Text normalization should be decided based on the purpose of the downstream task. A diverse set of text normalization strategies can be found in the literature. When training word embeddings, it is common to omit stopwords and to exclude all words that only occur occasional times in the corpora (Mikolov, Sutskever, et al. 2013). Although stopword removal, stemming, and lemmatization can be considered as universal techniques known to improve performance on many processing tasks (Silva and Ribeiro 2003). Some results indicate that such text normalization methods can remove important information and *degrade* the performance of certain downstream tasks (Riloff 1995). Finally, the tokenization procedure can look different for various domains. In social media, online-specific tokens, like hashtags, emojis, URLs, and user-handles can be treated as individual tokens, or as textual noise to be removed.

One approach to text processing in social media is to adapt existing tools to the social media domain (Gimpel et al. 2011; Derczynski et al. 2013). Another approach is to convert the social media text into a more formal language, that is suited for traditional NLP tools (Han and Baldwin 2011; Han, Cook, and Baldwin 2012). The latter approach requires additional text normalization to correct ill-formed words, such as spell correction and word substitution.

2.1.3 Information Extraction

Automatic extraction of structured information from unstructured pieces of text is referred to as *information extraction*. The canonical example of information extraction is the task of extracting named entities from unstructured text. Named-Entity Recognition (NER) can be realized by matching unstructured text to linguistic rules, which is a form of unsupervised information extraction. A major drawback of this method when applied to social media is that social media text often is not grammatically correct, reducing the utility of linguistic rules. For instance, many NER systems rely on correct capitalization, which is fragile in social media where the capitalization is unreliable. Another approach to information extraction is to use statistical methods. In general, the choice of method is dictated by the access to annotated training data. When annotated data are available, statistical methods are often preferred.

In the context of our research, the task of extracting clothing items, brands, materials, patterns, and styles from unstructured text associated with Instagram posts,

can be formulated as an information extraction task. Prior work has reported that information extraction systems developed for newswire text may not be effective in the social media domain due to the informal language (Ritter, Mausam, et al. 2012; Bhargava, Spasojevic, and G. Hu 2017; Habib and van Keulen 2014; Cherry and Guo 2015a).

Unsupervised techniques for information extraction includes clustering (Cherry and Guo 2015a), semantic word matching using word embeddings and cosine similarity (ibid.), edit distance matching (Reis et al. 2004), use of distant supervision sources such as knowledge bases and Application Programming Interface (API)s (Tabassum, Ritter, and W. Xu 2016), using Term Frequency-Inverse Document Frequency (TF-IDF) to identify the most characterizing words in a document based on corpora statistics (Seki 2003), and ontology-based information extraction (Wimalasuriya and Dou 2010). Moreover, supervised techniques for information extraction include machine learning models such as the Support Vector Machine (SVM) (Li, Bontcheva, and Cunningham 2005), and hidden Markov models (Bikel et al. 1997).

2.1.4 Natural Language in Social Media

Use of natural language in social media is different than in newswire text. Emojis, spelling errors, multi-lingual text, emoticons, grammatical errors, abbreviations, and hashtags are part of normal language use in social media. Furthermore, social media text tend to be diverse in style, as compared to corpora of newswire text, that are homogeneous in comparison. Throughout this thesis, I will refer to text with the mentioned characteristics as *noisy*.

Prior research on social media text is dominated by research on Twitter data. Although I anticipate the main research results from Twitter to be useful in my research on Instagram, there are also noticeable differences between the two domains. The most prevalent discrepancies being that Instagram is an image-sharing platform while Twitter is a micro-blogging platform, and that Twitter has a character-limit per tweet.

Baldwin et al. (2013) empirically compare social media text sources like Twitter, forums, and blogs to a corpus of edited English text. In summary, the findings reinforce the accepted notion that social media text is more noisy than other sources of text, having more grammatical errors, and being more heterogeneous. With that said, the authors too conclude that text normalization techniques can be effective in reducing the noise. Similar results have been reported by (Gouws, Hovy, and Metzler 2011) and (Baeza-Yates and Rello 2011).

One of the few quantitative studies on data from *Instagram* is provided by Y. Hu, Manikonda, and Kambhampati (2014). In their study, eight dominant photo categories were identified with the help of computer vision techniques and human expertise. Fashion was one of the eight identified categories, among other popular categories such as food, friends, and selfies.

2.1.5 Challenges

Text processing and understanding is a complex endeavor. The main difficulty for machines working with text can be ascribed to the *ambiguity* of natural language. We say that a piece of text is ambiguous if it can be processed and interpreted in multiple ways. Indeed, a substantial part of NLP is concerned with resolving ambiguities, from syntactic to semantic ambiguity. In addition to ambiguity, natural language is compositional, meaning that the semantic of a word depends on its context. Moreover, natural language is sparse, symbolic and discrete, referring to the fact that an intractable amount of word combinations exists, and individual words have meanings that cannot be traced down to their structure or statistical properties. As evident of the difficulty, natural language understanding is believed to be an AI-complete problem (Yampolskiy 2013). Besides the inherent challenges with natural language, the peculiarities of social media text adds to the complexity.

To sum up this section on background theory in the field of NLP, the field includes both linguistic methods and data-driven methods. Processing of text from social media has special needs that are not necessarily fulfilled by NLP methods that are effective for newswire text. This has led to the development of new methods for processing social media text. A general characteristic of these methods is a reduced emphasis on linguistic aspects and more emphasis of statistical text properties.

2.2 Learning Mechanisms

The goal of machine learning is to use a mixture of algorithms and data to train models capable of generalizing to new instances of some task. Although the lines have been blurred over time, machine learning applied to NLP falls into the category of statistical NLP, to distinguish it from symbolic NLP that deal with methods such as grammars, logic, and formal rules.

Learned semantic representations of words can drive information extraction and improve text classifiers (Section 2.2.1). Moreover, deep neural networks can learn to recognize patterns in text to perform complex classifications (Sections 2.2.2 and 2.2.4). When labeled data are a shortage, weak- and semi- supervised methods can be an alternative to supervised methods (Section 2.2.3). Finally, beyond the positive aspects of deep learning are also practical difficulties (Section 2.2.5).

2.2.1 Word Embeddings

Using embeddings to represent words is an old idea in NLP, with a recent upswing in attention due to progress in algorithms for deriving the embeddings. Word embeddings reduce common NLP tasks to mathematical vector operations. To understand the attractive properties of word embeddings, it is relevant to consider what is the alternative to word embeddings. The universal means of representing words in a numerical format is the *one-hot* encoding. In this type of encoding, a

word w_i with index i in a vocabulary V is represented as a vector $\vec{w_i} \in \mathbb{R}^{|V|}$, where all entries of $\vec{w_i}$ are set to 0 except for the entry with index i, that is set to 1. One-hot encoding is a *local* word representation in a discrete space. In one-hot encoding, a word is represented by a single binary value in a sparse vector. The issue with the one-hot encoded representation is that it does not convey any relation of word similarity. Given two one-hot encoded vectors, $\vec{w_i}$ and $\vec{w_j}$, it is possible to tell if the vectors represent the same word or not, but the word representations do not disclose any further information about the relation between the two words.

Word embeddings are distributed representations of words in a continuous space, represented by dense vectors $\vec{w_i} \in \mathbb{R}^d$ of a fixed dimension d. A typical choice of dimension is 300, which usually is many order of magnitude smaller than $|V|$. The word embeddings are either learned by use of optimization methods, or derived with spectral methods from word co-occurrence statistics. The idea of word embeddings is conceptually related to the *distributional hypothesis* (Harris 1954), saying that the meaning of a word is dictated by the contexts where it occurs, concisely described as *"You shall know a word by the company it keeps"* (Firth 1957). By incorporating this idea in the practice of deriving word representations, words that occur in similar contexts will obtain similar representations (Mikolov, Sutskever, et al. 2013). Thus, word embeddings can be used for automatic understanding of words, which is an important task in NLP. In addition, usage of word embeddings as generic word representations in extrinsic tasks, such as text classification, information extraction, and machine translation, is manifold (Habibi et al. 2017; B. Hu et al. 2014; Mesnil et al. 2013; Y. Zhang et al. 2016; Vora, Khara, and Kelkar 2017).

Several algorithms exist for computing word embeddings. The early work on word embeddings include Latent Semantic Analysis (LSA) (Deerwester et al. 1990), and probabilistic LSA (Hofmann 1999), that are based on matrix factorization methods for embedding words in a d-dimensional space. More recently, context window approaches for deriving word embeddings, practiced in (Collobert and Weston 2008) and extended by Word2vec (Mikolov, Sutskever, et al. 2013), GloVe (Pennington, Socher, and Christopher D. Manning 2014), and FastText (Bojanowski et al. 2016), have achieved unbeaten results on several metrics.

Neural Network Methods Lately, learning word embeddings based on predictive models within local context windows, often implemented as neural networks, has become the dominant approach to learn word embeddings. Such a model was popularized by Bengio et al. (2003), that proposed a neural network model where the input is a k-gram of words, and the output is a probability distribution over the next word. This idea was further developed by Collobert and Weston (2008), and more recently Mikolov, Chen, et al. (2013) introduced the Continuous Skip-gram (Skip-gram) model, and the Continuous Bag-of-Words (CBOW) model, which are the foundation for modern algorithms for learning word embeddings.

The Skip-gram model is a model for predicting the context of a word. The number of words to predict depends on the context window size c. A typical choice

is $c = 2$, meaning that the task of the model is to predict the two words in front of the center word, w_{t-1}, w_{t-2}, and the two words after the center word, w_{t+1}, w_{t+2}. The accuracy of the prediction is determined by the objective in Eq. (2.3), that should be maximized, where T is the number of words in the training corpus.

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) \tag{2.3}$$

The CBOW model does the reverse of Skip-gram, and predicts the center word w_t, given a set of context words $w_{t+2}, w_{t+1}, w_{t-1}, w_{t-2}$, intending to maximize the objective in Eq. (2.4).

$$\frac{1}{T}\sum_{t=1}^{T}\log p(w_t|w_{t+1}, w_{t+2}, w_{t-1}, w_{t-2}) \tag{2.4}$$

Both Skip-gram, and CBOW can be implemented as shallow neural networks that are efficient to train with gradient optimization (Fig. 2.2). In the neural network implementations, the objective functions are realized through the softmax function ($\sigma(\vec{z})_j = \frac{e^{\vec{z}_j}}{\sum_{k=1}^{K} e^{z_k}}$). The softmax operation can be interpreted as computing the probability of each word in the vocabulary V occurring in the context, conditioned on the center word (or vice-verse for CBOW). However, as the vocabulary of words V tend to be large, softmax becomes a computationally expensive operation. For this reason, most implementations use approximations of softmax, such as hierarchical softmax (Morin and Bengio 2005) and negative sampling (Mikolov, Sutskever, et al. 2013). During training, the parameters are updated to minimize the loss between the predicted probabilities and the actual words in the context window.

Evaluating Word Embeddings Word embeddings are evaluated in two ways, with extrinsic and intrinsic evaluations. Extrinsic evaluation refers to evaluation of word embeddings with respect to the performance on a downstream task, such as information extraction. As the downstream task generally is the end goal, this type of evaluation is the most significant, though more costly than the intrinsic counterpart.

Intrinsic evaluations are shallow assessments of the quality of word embeddings. Examples of intrinsic evaluations include evaluating the embeddings capability of inferring meaningful semantic relations between words, and word analogy solving. Word analogy solving alludes to the task of finding the missing word in puzzles such as "king is to man as queen is to X". For comparable evaluations of algorithms for learning word embeddings, there exist public datasets for intrinsic evaluation (Hill, Reichart, and Korhonen 2015; Finkelstein et al. 2002). Besides the standardized datasets for intrinsic evaluation, domain-specific datasets can be used (Tixier, Vazirgiannis, and Hallowell 2016).

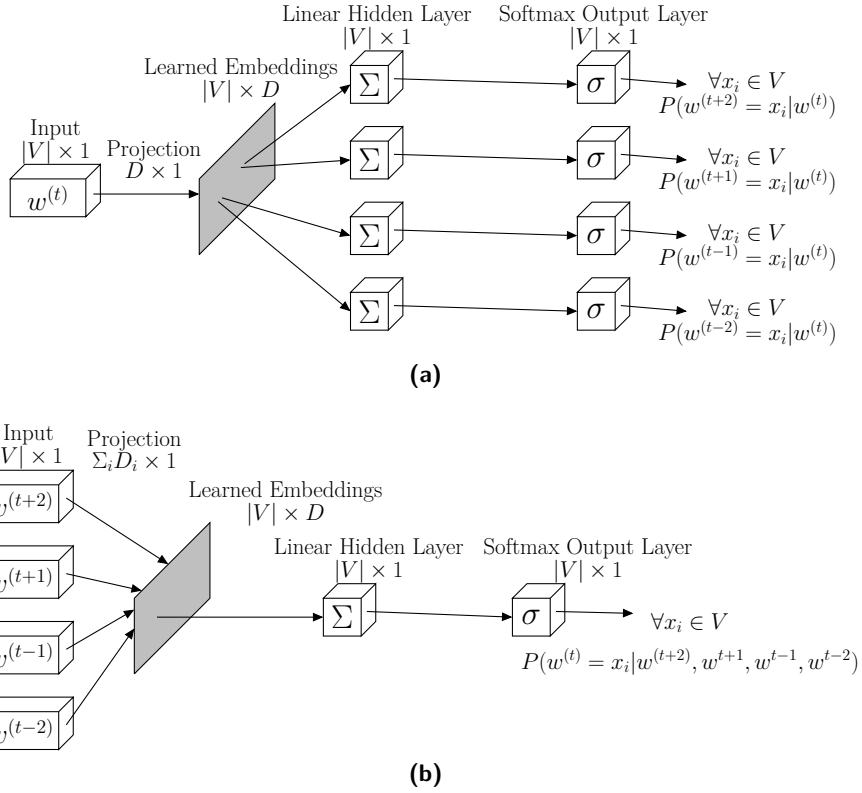

Figure 2.2: Skip-gram (Fig. 2.2a) and CBOW (2.2b). V denotes the vocabulary, and D denotes the embeddings' dimension.

State-of-the-art Algorithms Word2vec (Mikolov, Sutskever, et al. 2013) is a software package that implements both the Skip-gram model and the CBOW model. It stands out from prior work for being faster to train than prior implementations, owing to a technique called negative sampling. By using local context windows of words, Word2vec misses certain global statistics of the corpus. This shortcoming of Word2vec motivated the development of GloVe (Pennington, Socher, and Christopher D. Manning 2014), that combines the global corpus statistics used in spectral methods like LSA, with local context windows that are used in Word2vec.

Both Word2vec and GloVe assign distinct word embeddings to each individual word in the training corpora. Unlike Word2vec and Glove, in the FastText algorithm for learning word embeddings, each word is described by a bag of character n-grams. In FastText, an embedding representation is associated with *each n-gram* (Bojanowski et al. 2016). The advantage of using a more fine-grained assignment of embeddings is that the embeddings are not as sensitive to Out-Of-Vocabulary (OOV) words, which is particularly useful for languages that are rich on morphology. Alternative algorithms for learning word embeddings that are less used in practice are WordRank (Ji et al. 2015), and Eigenwords (Dhillon, Foster, and Ungar 2015).

2.2.2 Deep Learning

Deep learning is a subset of machine learning, where the input is modeled as a *deep* hierarchy of representations, hence the name. In deep learning, initial levels in the system learn representations of the input that are suitable for more sophisticated learning further down the hierarchy. The deep learning approach to learning has achieved unbeaten results in domains such as computer vision and NLP.

Deep learning has had a renaissance in recent years due to an increasing amount of data and computational power, which are important ingredients for deep learning pipelines. In addition, improvements in algorithms for training deep learning systems have contributed to the success. The neural network is the most common model for deep learning. Since the breakthrough in 2006 on how to circumvent the difficulty of training deep neural networks (G. Hinton and Salakhutdinov 2006), several success stories using deep learning have been reported.

Broadly speaking, neural networks consists of a set of connected computing elements (neurons) and activation functions φ. To make a network recognize a certain pattern, the parameters in the network are updated iteratively according to some objective based on truth labels. This procedure can be seen as a form of automatic programming, also called *learning* (Minsky and Papert 1988).

A *deep* neural network is a network where the computing elements are structured in several layers stacked on top of each other. Structuring the network in a deep hierarchy have shown to increase the network's expressive power and to be exponentially more effective than shallow networks (Mhaskar, Liao, and Poggio 2016).

With the expressive power comes also challenges. Deep neural networks are notoriously hard to train compared with less complex models, and often require parameters that can only be found via experimentation and empirical research. However, a generic trait of deep learning systems is that they work best when trained on vast amounts of data (Sun et al. 2017).

Types of Deep Neural Networks Neural networks are distributed processing devices that come in many shapes and forms. Two of the most frequent architectures are the CNN (Lecun et al. 1998) and the Recurrent Neural Network (RNN).

Traditionally, CNNs have reached most success on computer vision tasks, dominating the the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al. 2015). CNNs have an architecture inspired from the visual cortex in human brains, with built-in assumptions that the input has a grid-like shape, and that the recognition should be equivariant to the input representation. The network consists of a combination of convolutional, pooling, and fully connected layers. The convolutional layers make use of localized feature detectors, also called *filters*, that are connected to smaller regions of the input. Although computer vision is the dominating use case of CNNs, in recent times, CNNs have been used for NLP tasks as well. In particular, the CNN has been fruitful for text classification (Kim 2014; Conneau et al. 2016). When applied to text, CNNs achieve context awareness

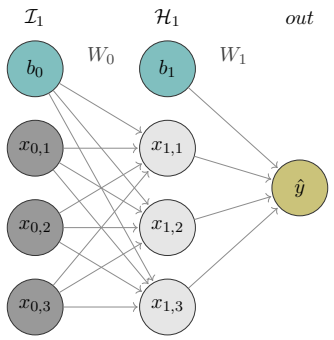

Figure 2.3: A two-layered feed-forward neural network.

through the filters that focus on n-grams of the input.

As opposed to the CNN architecture, RNNs are not common in computer vision. RNNs excel at sequence learning and are a common tool for NLP tasks that can be formulated as sequence prediction tasks. Such as, machine translation (Bahdanau, Cho, and Bengio 2014) and question answering (Iyyer et al. 2014). The architecture of RNNs have a built-in assumption that the input is in a sequence-format. In RNNs, the prediction at one part of the input sequence depends on the previous parts of the sequence, modeled with a loop-back connection in the neural network.

Backpropagation The dominant optimization technique for training deep neural networks is gradient descent, where the gradient is computed with a procedure called backpropagation (D. E. Rumelhart, G. E. Hinton, and Williams 1986). When training a neural network with gradient descent, after each iteration, the parameters are updated based on the partial derivatives with respect to the loss function. Backpropagation implements the computation of the partial derivatives in neural networks in an efficient manner by propagating the errors backwards in the network and caching intermediate values.

I now turn to an example to illustrate the workings of backpropagation using computational graphs. A computational graph is an abstraction that can be used to represent the flow of computations in neural networks. A node in the computational graph refers to an operation, and inputs to operations are denoted with arrows.

To exemplify how neural networks are trained in a supervised manner, consider the network in Fig. 2.3. If I assume that the network uses the logistic sigmoid ($\sigma(x) = \frac{1}{1+e^{-x}}$) as the activation function for both hidden and output units, and that the network is trained to minimize the mean squared error with respect to truth labels y ($\mathcal{L}(y, \hat{y}) = \frac{1}{2}\sum_i (y_i - \hat{y})^2$). Then, Fig. 2.4, and Fig. 2.5 shows the corresponding computational graphs for the neural network for forward, and backward propagation, respectively, where $\vec{x} \in \mathbb{R}^3, W_0 \in \mathbb{R}^{3\times3}, b_0 \in \mathbb{R}^3, W_1 \in \mathbb{R}^3, \hat{y} \in \mathbb{R}$.

Let $\vec{x}$ be the input data and y be the truth label, then the forward propagation of the network computes the composite function $\hat{y} = f(\vec{x}) = \sigma(W_1^T \sigma(W_0^T \vec{x} + \vec{b_0}) + \vec{b_1})$,

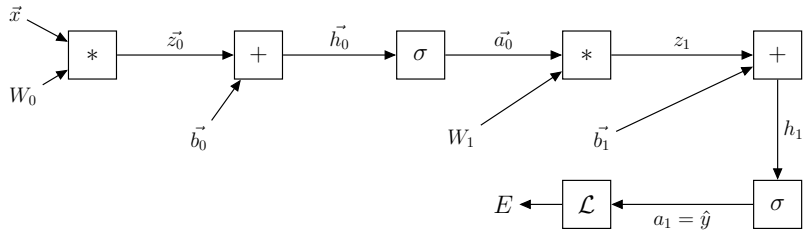

Figure 2.4: Computational graph for forward propagation of the network in Fig. 2.3.

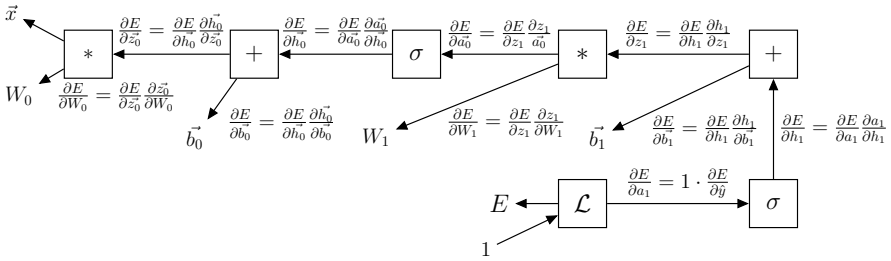

Figure 2.5: Computational graph for backward propagation of the network in Fig. 2.3.

and the backward propagation computes the gradient $\nabla_\theta \mathcal{L}(y, \hat{y}) = (\frac{\partial E}{\partial W_{1,1}}, \ldots, \frac{\partial E}{\partial W_{n,n}})$, where θ is the set of parameters in the network. The forward propagation can be broken down into seven operations: two additions, two dot products, two sigmoid operations (applied element-wise to vectors), and one application of the loss function. The results of these operations are in turn stored in seven variables (Eq. (2.5)).

$$\begin{gathered} \vec{z_0} = W_0^T \vec{x} \in \mathbb{R}^3, \quad \vec{h_0} = \vec{z_0} + \vec{b_0} \in \mathbb{R}^3, \quad \vec{a_0} = \sigma(\vec{h_0}) \in \mathbb{R}^3, \\ z_1 = W_1^T \vec{a_0} \in \mathbb{R}, \quad h_1 = \vec{z_1} + \vec{b_1} \in \mathbb{R}, \quad \hat{y} = \vec{a_1} = \sigma(\vec{h_1}) \in \mathbb{R}, \\ E = \frac{1}{2} \sum_i (y_i - \hat{y})^2 \in \mathbb{R} \end{gathered} \tag{2.5}$$

The backward propagation seeks to compute the gradient $\nabla_\theta \mathcal{L}(\vec{x}, y)$, that contains the partial derivatives of all of the variables in the network, θ. For completeness, the symbolic derivatives of the example network in Fig. 2.3 are given in Eq. (2.6). The derivatives are derived by recursive application of the chain-rule from calculus to the composite function that the forward propagation computes. The partial derivatives of two vectors $\frac{\partial \vec{v_1}}{\partial \vec{v_2}}$, where $\vec{v_1} \in \mathbb{R}^n$, $\vec{v_2} \in \mathbb{R}^m$, and $g(\vec{v_1}) \to \vec{v_2}$ refers to the $n \times m$ Jacobian matrix of g. To see how this notation can be generalized to higher-order tensors, I refer to (Goodfellow, Bengio, and Courville 2016).

$$\begin{gathered} \frac{\partial E}{\partial \vec{b_1}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_1} \frac{\partial h_1}{\partial \vec{b_1}}, \quad \frac{\partial E}{\partial W_1} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial W_1}, \\ \frac{\partial E}{\partial \vec{b_0}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial \vec{a_0}} \frac{\partial \vec{a_0}}{\partial \vec{h_0}} \frac{\partial \vec{h_0}}{\partial \vec{b_0}}, \quad \frac{\partial E}{\partial W_0} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial \vec{a_0}} \frac{\partial \vec{a_0}}{\partial \vec{h_0}} \frac{\partial \vec{h_0}}{\partial \vec{z_0}} \frac{\partial \vec{z_0}}{\partial W_0} \end{gathered} \tag{2.6}$$

In the symbolic derivatives above (Eq. (2.6)), notice that several sub-expressions are computed repeatedly. For large neural networks, training time can be reduced drastically by caching the computation of intermediate sub-expressions. Backpropagation provides a principled way for doing exactly this.

Finally, once the gradient is computed using backpropagation, gradient descent can be used to perform "learning". This is done by updating the parameters in the network using the partial derivatives and a learning rate η to minimize the loss, as demonstrated in Eq.(2.7).

$$\begin{aligned} \Delta_{\vec{b_1}} &= -\eta \frac{\partial E}{\partial \vec{b_1}}, \quad \Delta_{W_1} = -\eta \frac{\partial E}{\partial W_1}, \\ \Delta_{\vec{b_0}} &= -\eta \frac{\partial E}{\partial \vec{b_0}}, \quad \Delta_{W_0} = -\eta \frac{\partial E}{\partial W_0} \end{aligned} \tag{2.7}$$

2.2.3 Weak- and Semi- Supervised Learning Techniques

Supervised machine learning has enabled several success stories for machine understanding of text. For instance, machine translation (Bahdanau, Cho, and Bengio 2014), text classification (Kim 2014), and question answering (Iyyer et al. 2014). However, acquisition of labeled training data is expensive and not always feasible.

Due to the scarcity of labeled training data, research on exploiting unlabeled data for training has received attention. For certain tasks, completely unsupervised learning is enough, such as the task of learning word embeddings. For other tasks, a blend of supervised and unsupervised learning is appropriate. Semi-supervised and weakly-supervised learning are two approaches to learning with limited amount of supervision, while having access to an abundant amount of *unlabeled* data.

In semi-supervised learning, even though it is assumed that a smaller amount of labeled training data are available, the goal is to combine that data with a larger portion of unlabeled data. To train with unlabeled data, semi-supervised learning makes use of assumptions about the data, such as the data distribution. With the right assumptions, semi-supervised learning algorithms are able to relate the unlabeled data with the labeled data to drive the learning process.

Weakly supervised learning methods rely on availability of weak-supervision signals and do not assume that any labeled data are available. A weak supervision signal can for instance be in the form of an external API, a crowdworker, or a domain heuristic. As opposed to strong supervision, weak supervision seldom has perfect accuracy or coverage.

The Data Programming Paradigm With the data programming paradigm (A. J. Ratner et al. 2016), weak supervision is encoded with *labeling functions*. A labeling function is any function $\lambda_i : x \rightarrow y$, that takes as input a training example x, and outputs a label y. A labeling function is typically realized through some domain heuristic and labels only a subset of the data. Naturally, labels produced by such functions are less accurate than labels produced by human annotators.

However, the idea of weak supervision is that weak labels can be complementary to each other. Different weak labels can be combined with the purpose of obtaining more accurate labels. The innovative part of data programming is the way that it learns a generative model of the labeling process in an unsupervised fashion.

Formally, a labeling function λ_i has a probability β of labeling an input, and refrain from labeling an input with probability $1-\beta$. Similarly, a labeling function has a probability α of labeling an input correctly. The combination of functions can be modeled as a generative model $\pi_{\alpha,\beta}(\Lambda, Y)$. Where Λ is the output matrix after applying all of the labeling functions to the unlabeled data ($\Lambda_{i,j} = \lambda_j(x_i)$), and Y is the true classes, modeled as latent variables. In Λ, an empirical probability $\hat{p}_{i,j}$ that two labeling functions λ_i and λ_j agrees can be inferred. By using the observed probabilities of overlap, the accuracy of each labeling function can be *estimated* using maximum likelihood estimation. Consequently, the problem of finding the parameters α and β that best describe the observed overlaps among labeling functions can be phrased as the optimization problem defined in Eq. (2.8), where S denotes the training set (A. J. Ratner et al. 2016).

$$(\hat{\alpha}, \hat{\beta}) = \arg\max_{\alpha,\beta} \sum_{x \in S} \log P_{(\Lambda,Y)\sim\pi_{\alpha,\beta}}(\Lambda = \lambda(x)) \tag{2.8}$$

Finally, after estimating α and β, the parameterized generative model is used to engender probabilistic training labels $p(Y|\Lambda)$ from the unlabeled data and the output of the labeling functions. When producing the probabilistic labels, more weight is given to accurate labeling functions, and the uncertainty of each label is indicated by the probability. If labeling functions disagree on a training example, this is encoded as an uncertainty by giving the corresponding label a lower probability. Subsequently, the probabilistic training labels can be used to train a discriminative machine learning model in a supervised manner. As the labels are probabilistic and not binary, a *noise-aware* loss function is used when training a discriminative model with such labels. A noise-aware loss function is a loss function for minimizing the expected loss with respect to the probabilistic labels.

2.2.4 State-of-the-Art in Text Classification

Traditionally, the naive Bayes classifier, and the SVM have been popular models for supervised text classification tasks, and are common baselines when evaluating new models (Wang and Christopher D. Manning 2012). Presently, deep learning models have reached state-of-the-art results on several text classification benchmarks. In particular, variants of the CNN architecture (Kim 2014; Conneau et al. 2016; Johnson and T. Zhang 2014), and the recurrent architectures (Lai, L. Xu, et al. 2015) have been effective. Another deep model with comparable results is the recursive deep neural network (Socher et al. 2013). As opposed to supervised text classification, standardized benchmarks do not exist for weakly supervised and semi- supervised models.

2.2.5 Challenges

Among the most prevalent challenges in machine learning is lack of labeled training data, shortage of computational resources for training, and *overfitting*. If a model is able to fit the training data and still has a high test error, the model is considered to overfit the training data. By controlling the model's capacity, overfitting can be reduced. The mechanism to control the model's capacity is called *regularization*.

In addition to overfitting, a challenge inherent in many domains is the *curse of dimensionality*. The curse of dimensionality refers to the phenomenon that the number of possible input configurations quickly blow up as problems get complex, making learning intractable. The most effective machine learning systems surmount the curse of dimensionality by relying on assumptions about the data, but it also reduces the generality of the system.

With this I complete the presentation of learning mechanisms for textual data that relate to our research. Below is a summarization of key concepts from this section that have implications on the choice of methods for our purpose.

Word embeddings are learned representations of words that capture word semantics based on word co-occurrence statistics. Unlike training of word embeddings, the bulk of machine learning models for text processing requires labeled data for training, something that is not available in our domain. Machine learning paradigms that can be applied without labeled data includes semi- and weakly- supervised learning, both of which traditionally have been less successful than supervised learning. Finally, *data programming* is a recent method for learning with weak supervision, where weak supervision is encoded with labeling functions.

2.3 Related Work

There exists a large amount of research on NLP and machine learning applied to text from social media. A major part of this research has been made on data from Twitter, because of their liberal data policy. This section presents the related work that the research presented in this thesis builds upon. The section covers prior work in the areas of scalable learning of domain-specific word embeddings, information extraction in social media, and weakly supervised classification.

Domain-Specific Word Embeddings The practice of constructing word embeddings targeted to a specific domain is a new field of research. Some examples are (Tixier, Vazirgiannis, and Hallowell 2016), where embeddings were trained for the construction domain, and (Major, Surkis, and Aphinyanaphongs 2017; Chiu et al. 2016) that trained embeddings for the biomedical domain. In summary, the results indicate that domain-specific embeddings can be beneficial over generic embeddings for certain tasks, and that hyperparameter tuning is important when training new embeddings. No prior experiments have been made with embeddings for the fashion domain on Instagram.

Distributed Training of Word Embeddings For scaling out the training of word embeddings, the main inspiration comes from Ordentlich et al. (2016), that scaled out training of Word2vec using the parameter server architecture, and the data-parallel Word2vec implementation in Spark (Zaharia et al. 2010). My work focuses on distributing the training of FastText, for which no distributed implementation or benchmark exists.

Unsupervised Information Extraction Ritter, Mausam, et al. (2012) devise an approach to event extraction and categorization that uses a supervised tagger to identify events in tweets. Next, the extracted events are categorized using latent variable models, that can make use of unlabeled data. Results demonstrate an improved accuracy compared with a supervised baseline. Their work resembles ours in that they attempt to classify and extract information from noisy text, and try to make use of unlabeled data. However, it has some important differences compared to our setting. In event categorization, the categories are unclear a priori, which fits well into the latent variable model approach. In contrast, our extraction problem has a pre-defined set of classes. Moreover, in their proposed solution, they assume access to an annotated dataset for training a tagger to recognize events in tweets, a corresponding dataset is not available in our domain.

Numerous research efforts have been made on the line of coarse-grained classification in social media using latent variable models (Ritter, Cherry, and Dolan 2010; Ritter, Clark, et al. 2011). These studies differ from our work in two ways. First, most of the work is focused on Twitter. Second, in our research, the goal is a complex multi-label extraction with pre-defined classes, while the aforementioned work typically target more general extraction tasks, often without pre-defined outputs.

Word embeddings have shown to be a great asset for information extraction. In (Vine et al. 2015), the authors evaluate how useful word embeddings are for clinical concept extraction and in (Cherry and Guo 2015b), the utility of word embeddings for NER on Twitter is evaluated. Both results demonstrate improvements when using word embeddings compared with baseline methods.

Text Classification with Weak Supervision For text classification, our research builds primarily on results from supervised machine learning. The success of this paradigm of machine learning has traditionally been coupled to large annotated datasets. Notable results in supervised text classification are (Severyn and Moschitti 2015; Kim 2014; Conneau et al. 2016), all of which differ from our research in that they rely on large annotated text corpora for training the classifier.

More recently, weakly supervised approaches have been used for text classification and information extraction. Specifically, the data programming paradigm presented in (A. J. Ratner et al. 2016), has achieved promising results. Data programming has been applied to binary and multinomial text extraction and classification tasks (A. Ratner et al. 2017). To the best of my knowledge, it has neither been applied to multi-label classification tasks, nor to social media text.

Chapter 3

Approach

This chapter covers the strategy for this research. In Section 3.1, I give an overview of how the research was framed and how it relates to the problem statement from Section 1.3. Moreover, Section 3.2 describes the Instagram corpora that have been used for experiments and analysis. Finally, Section 3.3 presents the methods in more detail, how they are related to each other, and the motivation for using them.

3.1 Overview

As stated in the problem statement from Chapter 1, In this research, my goal has been to study how we can perform accurate and scalable text mining of Instagram data. The experiments in this thesis can be divided into four categories.

- Training of domain-specific word embeddings on an Instagram corpora.
- Distributed training of word embeddings.
- Unsupervised text mining of Instagram data using domain knowledge.
- Deep text classification of Instagram text with weak supervision.

On a holistic perspective, the experiments are designed to exercise two approaches to the same problem. In text mining, there is a balance between knowledge engineering (examined in Chapter 4), and training with data (covered in Chapter 5). While the former approach has the disadvantage of being dependent on domain knowledge, it is not limited by access to labeled training data, as the latter approach is.

3.2 Data

This section describes the Instagram corpora that have been processed to obtain the results in this thesis. The section also describes a smaller annotated dataset that has been used for evaluation purposes.

3.2.1 Instagram Corpora

Experiments presented in this thesis have been conducted on a provided dataset, consisting of Instagram posts from a community of users in the fashion domain. The data are in the form of a corpora consisting of image captions, user comments, and usertags associated with each post. In entirety, the corpora consists of 143 accounts, 200K posts, 9M comments, and 62M tokens, out of which 2M are unique. The numbers were computed before any pre-processing, except applying the NLTK (Loper and Bird 2002) TweetTokenizer and removing user-handles.

3.2.2 Ground Truth

For evaluation purposes, a dataset of 200 annotated Instagram posts was used. Each annotated post includes an annotation for each fashion item in the associated image. Annotations comprise of item-category, fabric, pattern, style, and brand. The annotation was a collective work by four participants in our research group. The average number of annotated clothing items per post is 3. Moreover, noteworthy is that the truth labels are based on the image associated with the text. In that sense, evaluation using this dataset is unfavorable for the text-based analysis. Since the labels are decided by the image, certain posts can have labels that cannot be inferred from the text alone, degrading the measured performance of the developed text mining models.

3.3 Methods

The research was initiated by an empirical study of the Instagram corpora (Section 4.1). The purpose of the study was to better understand Instagram as a source of text. The results of the empirical study were helpful in deciding appropriate methods for text mining of Instagram data.

Following the results of the empirical study, I have identified *word embeddings* as a key component for text mining of social media data. With word embeddings, the semantics of words can be uncovered based on word co-occurrence statistics, rather than relying on synonym lists or exact linguistic methods. This approach to text understanding is robust to noise and just as applicable to informal text as it is to newswire text. However, in the analysis of the Instagram corpora, it was found that nearly half of the tokens in our Instagram corpora do not have an embedding representation in the pre-trained word embeddings that are available. This is because most of the off-the-shelf word embeddings have been trained on formal newswire text, that has a different vocabulary than social media text. Due to this mismatch, I have trained domain-specific word embeddings using our Instagram corpora (Section 4.2).

Training word embeddings is time-consuming. Moreover, the majority of existing implementations for training word embeddings are not scalable. This motivated the development of FastTextOnSpark, an implementation of FastText that can

scale out and train word embeddings using a cluster of machines. The implementation of FastTextOnSpark is presented in Section 4.3.

Finally, using word embeddings as a central component, two systems for text mining of Instagram data was developed. The first implementation uses word embeddings in combination with domain knowledge to extract fashion details from Instagram text (Section 4.4). The second implementation uses weak supervision from open APIs and NLP methods to label a dataset of Instagram posts. After the labeling process, the labeled data are used to train a deep text classifier for predicting clothing items in Instagram posts (Section 5.2).

Chapter 4

Unsupervised Text Mining with Word Embeddings

This chapter describes collected results from analyzing an Instagram corpora and experimenting with unsupervised methods for text mining. The chapter covers data exploration to quantify the characteristic of the corpora (Section 4.1), an assessment of word embeddings trained on Instagram text (Section 4.2), and a presentation of a scalable system for training with the FastText algorithm (Section 4.3). Finally, the chapter describes an information extraction system (Section 4.4) that uses the trained embeddings to extract fashion details from Instagram posts in an unsupervised manner.

4.1 Data Analysis

Data analysis was done to better understand Instagram as a data domain of text. In Section 4.1.1 the experimental setup of the data analysis is presented and Section 4.1.2 presents the results.

4.1.1 Experimental Setup for an Empirical Study of Instagram Text

Of special interest in the data analysis was to quantify how the corpora differs from newswire text, as it affects the choice of processing methods. To measure the fraction of emojis, hashtags, and user-handles, the NLTK (Loper and Bird 2002) TweetTokenizer was used to tokenize the text, and regular expressions were applied to extract the desirable tokens. To quantify the amount of Out-Of-Vocabulary (OOV) words, two vocabularies were used, the Google-news vocabulary (Google 2013), and GNU aspell v0.60.7[1]. The Google-news vocabulary was chosen to illuminate the mismatch between off-the-shelf word embeddings and the Instagram corpora, and the aspell dictionary was used to enable comparison with related work.

[1] `http://aspell.net/`

Table 4.1: Measurements of lexical noise in the corpora.

Text Statistic	*Fraction of corpora size*	*Average/post*	*St. Dev*	*Min/post*	*Max/post*
Emojis	0.15	48.63	141.15	0	17938
Hashtags	0.03	9.14	12.48	0	1325
User-handles	0.06	18.62	232.74	0	46208
Google-OOV words	0.46	145.02	477.89	0	58832
Aspell-OOV words	0.47	147.61	486.27	0	58859

Finally, `langid.py` (Lui and Baldwin 2012) was used to capture the distribution of languages in the corpora.

4.1.2 Results from the Empirical Study

In this section, results from exploratory data analysis of the Instagram corpora is presented.

What Characterizes Instagram as a Source of Text? Table 4.1 contains statistics that capture the distinctive properties of the Instagram corpora compared with newswire text. Removing all online-specific tokens (hashtags, user-handles, emojis, URLs) results in an OOV fraction of 0.30 based on the aspell dictionary, that can be compared with the fraction 0.25 that was obtained by Baldwin et al. (2013) on a Twitter corpora using the same pre-processing and dictionary.

How Multi-Lingual is Instagram Text? Although all Instagram posts in the corpora are from English accounts, the comments sections are often multi-lingual. Applying `langid.py` (Lui and Baldwin 2012) on the set of 9 million comments reveals that 52% of the comments are primarily written in English. The Language identified as the second most common was Chinese on 6.5%, followed by Japanese on 5%, German on 3%, and Spanish on 2%. In total, 97 languages were identified in the set of comments.

How Is the Text Distributed on Instagram? The number of comments associated with Instagram posts is varying. Data analysis reveals that the distribution of comments and amount of text associated with posts exhibit the long tail phenomenon. The frequencies of number of comments roughly follows a *power law* relationship (Fig. 4.1). Some posts have no comments at all, while others have a few thousand comments. The mean length of captions and comments in the corpora is 29, and 6 tokens, respectively.

4.2 Learning Domain-Specific Word Embeddings

Considering the substantial ratio of OOV words (Table 4.1), and the informal language used in social media, in this section I survey the benefit of training new

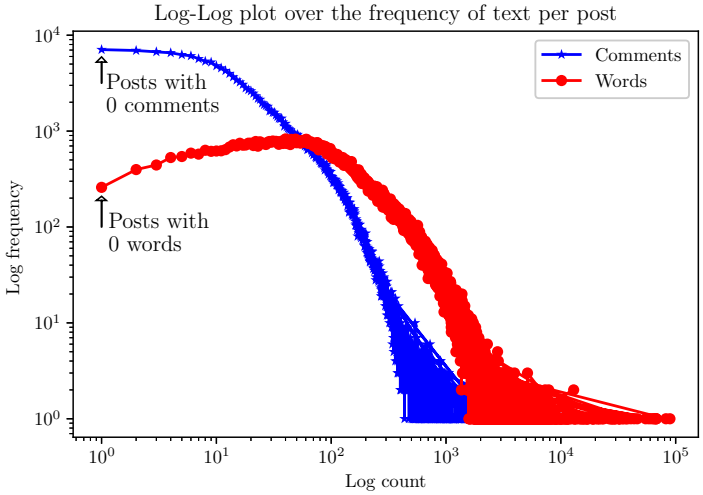

Figure 4.1: The text distribution in the corpora.

embeddings for the fashion domain on Instagram. Furthermore, I provide an evaluation of embeddings trained on our corpora of Instagram posts using Word2vec, Glove, and FastText, with varying hyperparameters.

4.2.1 Experimental Setup for Training and Evaluating Word Embeddings

This section outlines the setup that was used to train and evaluate word embeddings for Instagram text.

Baselines To examine the difference between domain-specific word embeddings and generic word embeddings, the embeddings trained on the Instagram corpora was compared with the state-of-the-art off-the-shelf embeddings, provided by Google, Facebook, and Stanford's NLP group. Specifically, the baselines were: (1) FastText-Wiki, consisting of embeddings pre-trained with the FastText algorithm on a corpus of Wikipedia articles, published by Facebook (Bojanowski et al. 2016); (2) Word2Vec-GNews, consisting of embeddings pre-trained with the Word2vec algorithm on a corpus of Google news articles, published by Google (Google 2013); (3) and (4) GloVe-Wiki, and GloVe-Twitter, consisting of embeddings pre-trained with the GloVe algorithm respectively on a corpus of Wikipedia articles, and tweets, published by the Stanford NLP group (Pennington, Socher, and Christopher D. Manning 2014).

Hyperparameters To find the best algorithm and hyperparameters for training word embeddings on the Instagram corpora, word embeddings with varying dimension and context window size were trained. Moreover, the evaluation included all

of the state-of-the-art algorithms for training word embeddings. When training embeddings with GloVe, the GloVe-python implementation was used[2], for FastText, the official C++ implementation was used[3], and for Word2vec, the gensim implementation was used[4]. I have made the best performing embeddings publicly available[5]. Parameters that were not tuned in the evaluation, were kept to their default values, listed in Table 4.2.

Table 4.2: Default parameters used when training word embeddings.

Parameter	*Value*
Iterations	15
MinCount	5
Learning rate	0.025
Learning rate update rate	100
Minimum n-gram (FastText)	3
Maximum n-gram (FastText)	6
Output layer	Hierarchical softmax
Max count (GloVe)	100

Evaluation Datasets Three datasets were used to evaluate trained word embeddings on the word similarity task, (1) WordSim353, introduced by (Finkelstein et al. 2002), is a dataset consisting of 353 word pairs with accompanying relatedness scores; (2) SimLex-999, presented in (Hill, Reichart, and Korhonen 2015), is a dataset of 999 word pairs and similarity labels; and (3) FashionSim, an open-source[6] dataset consisting of 307 fashion related words and relatedness scores, collectively annotated by our research group in cooperation with fashion experts.

Evaluation Metric The relative word rankings were measured with Spearman's rank correlation coefficient, ρ. The similarities on the evaluation datasets were compared to the cosine similarity between corresponding word embeddings. Spearman's rank correlation is suitable for measuring monotonic relationships between variables, that fit well with the word similarity task, and enables comparison with prior results.

Pre-processing Before training word embeddings, the Instagram corpora was pre-processed by converting the text to lowercase, tokenizing the text with the TweetTokenizer in NTLK (Loper and Bird 2002), removing user-handles, removing all URL's, as well as removing stopwords. The stopwords were removed based on a custom stopword list, consisting of standard English stopwords listed in the

[2] `https://github.com/maciejkula/glove-python`
[3] `https://github.com/facebookresearch/fastText`
[4] `https://github.com/RaRe-Technologies/gensim`
[5] `https://github.com/shatha2014/FashionRec`
[6] `https://github.com/shatha2014/FashionRec`

NLTK stopword list (ibid.), as well as domain-specific stopwords. Finally, all words were lemmatized and lemmas that occurred fewer than 5 times in the corpora were removed before training the word embeddings.

The TweetTokenizer was preferred as it is designed to recognize social media text. Removal of urls, stopwords and rare words, as well as lemmatization, was motivated by previous results reporting that it improved the usefulness of the final embeddings (Mikolov, Sutskever, et al. 2013; Sugathadasa et al. 2017). Moreover, user-handles were removed from the corpora based on experimentation. It was found that user-handles occur frequently in the corpora, yet bear little, if any, predictive power for word co-occurrence statistics.

Significance Testing Significance testing was done by applying the t-test and calculating the two-sided p-value for the null-hypothesis that the similarities provided by a set of embeddings are uncorrelated with the ground-truth in the evaluation dataset. To regard the result as a significant correlation, a p-value below 0.01 was required.

4.2.2 Results from an Intrinsic Evaluation of Word Embeddings

In this section, word embeddings trained on the Instagram corpora are examined. The experiments include a comparison between Instagram embeddings and off-the-shelf embeddings, as well as hyperparameter tuning of embeddings trained on Instagram text.

How compatible are Off-the-shelf Embeddings for Social Media? Off-the-shelf embeddings outperform the domain-specific embeddings on general evaluation metrics such as Simlex-999 (Hill, Reichart, and Korhonen 2015), and Wordsim353 (Finkelstein et al. 2002). However, on the FashionSim evaluation dataset the reversed relationship occurs (Fig. 4.2). To exemplify, the embeddings FastText-Fashion had the lowest score on the Simlex-999 evaluation dataset and the highest score on the FashionSim dataset.

What are Suitable Hyperparameters? It can be observed that FastText and Word2vec are highly dependent on the hyperparameter settings, while Glove is stable in comparison (Fig. 4.3). FastText demonstrated the best results on the given task. With FastText, the top accuracy was achieved with Skip-gram and context window size 2. A prevalent trend in the results is that CBOW performed better with larger window sizes, as opposed to Skip-gram that achieved the highest results with smaller context windows. Additionally, a substantial boost in accuracy was observed when increasing the vector dimension from 50 to 100, and then a less significant increase when further raising the dimension up to 300. When the dimension is increased above 300 there is a diminishing return of increased accuracy relative to the increased dimension.

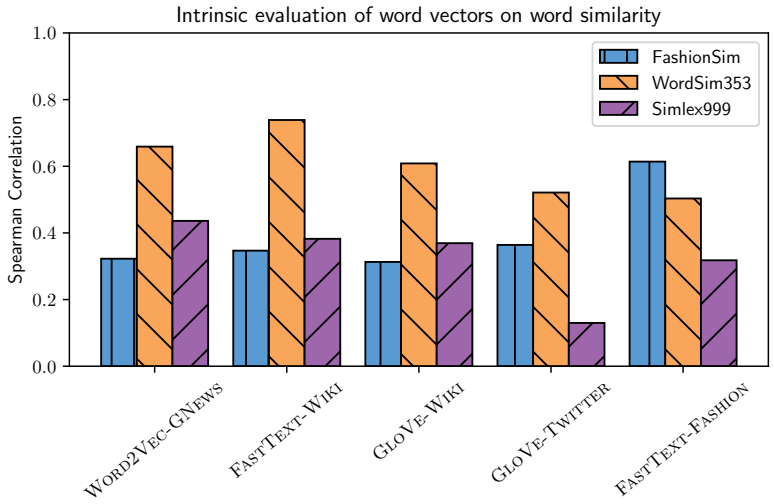

Figure 4.2: Intrinsic evaluation on the word similarity task (p-value < 0.001).

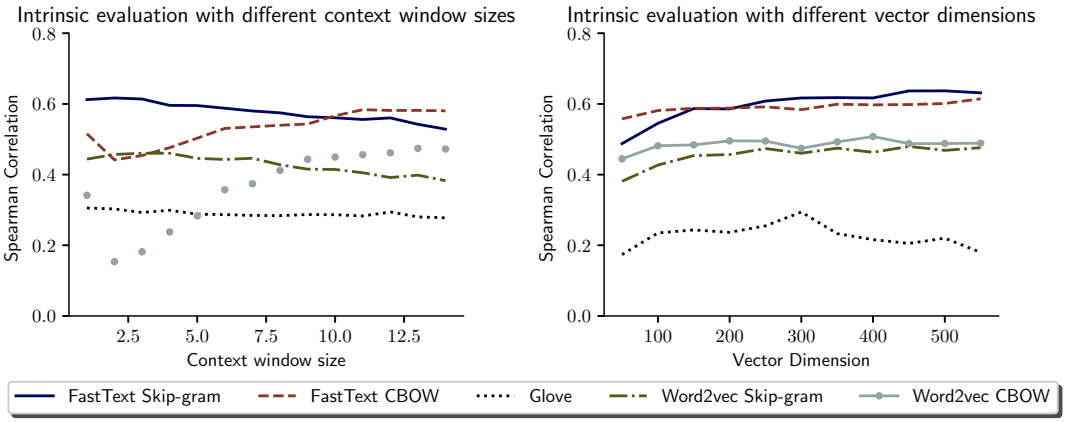

Figure 4.3: Hyperparameter tuning on the FashionSim evaluation dataset (p-value $< 1.76\text{e}{-5}$). When tuning the context-window size the dimension was 300. When tuning the dimension the fine-tuned window sizes $2, 11, 12, 3, 13$ for FastText Skip-gram, FastText CBOW, Glove, Word2vec Skip-gram,Word2vec CBOW, was used.

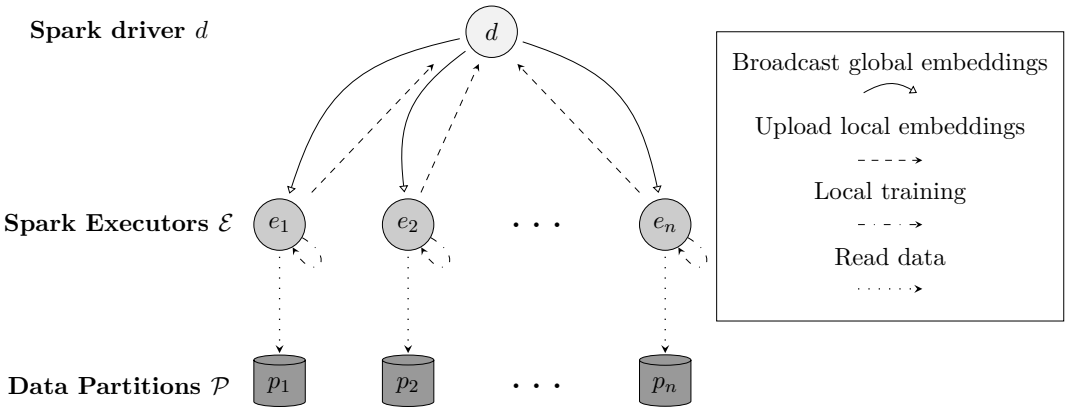

Figure 4.4: Architecture of FastTextOnSpark.

4.3 Distributed Training of Word Embeddings

Training of word embeddings is a data-driven undertaking, where the quality of the resulting embeddings is dependent on the amount of data and length of training (Lai, Liu, et al. 2015). Therefore, *scalability* is a concern. As evidence, the state-of-the-art word embeddings are all trained on corpora consisting of several billion words. Moreover, as the experiments in the previous section testify to, finding appropriate embeddings involves time-consuming empirical experiments.

The official C++ implementation of FastText is multi-threaded but limited to a single machine, making it a bottleneck in big data pipelines. With this background, I have ported the official FastText implementation for training word embeddings to run in a distributed setting on the Spark engine (Zaharia et al. 2010), inspired by the Word2vec counterpart. This section describes the implementation, analyzes it theoretically, and evaluates it with empirical experiments.

The implementation, FastTextOnSpark, is open source[7], and can scale out training of FastText word embeddings to a cluster of machines to reduce the training time. FastTextOnSpark is *data-parallel* and partitions the training data over the set of available machines, denoting one machine as the driver d, and the rest as executors $\mathcal{E}$. For each training iteration, every executor gets a local copy of the word embeddings, and updates the embeddings with gradient-based optimization using its partition of the training set. After every iteration, each executor sends its modified embeddings to the driver. Finally, the driver updates the global embeddings by adding the modified embeddings together. Once the global embeddings have been updated, the driver broadcasts the updated embeddings to the executors, who proceed to the next iteration. The system is illustrated in Fig. 4.4.

For sake of understanding, a formal definition of the Skip-gram model as it is used in FastTextOnSpark is given below. I first describe the general Skip-gram model (Section 4.3.1), and then how it is extended with subword information by the

[7] `https://github.com/shatha2014/FashionRec`

FastText algorithm (Section 4.3.2), and finally how the computation is distributed in FastTextOnSpark to achieve close to linear scaling (Section 4.3.3).

4.3.1 The Continuous Skip-gram Model

Let the sequence of words $w_1, \ldots, w_T$ denote the training corpus. Then, if we recall the definition from Chapter 2, the Skip-gram model aspires to maximize the objective in Eq. (4.1).

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) \tag{4.1}$$

When implemented as a neural network, the computation in Skip-gram for each word in the training corpus is as follows. The center word w_t is first projected into its embedding representation $\vec{v_{w_t}} \in \mathbb{R}^d$, where d is the vector dimension (as visualized in Fig. 2.2a). Next, the predicted probability of each word w_c in the vocabulary V occurring in the context window is computed. The probabilities are obtained with softmax. Input to softmax is the dot product between the center word vector $\vec{v_{w_t}}$ and the weights between the projection layer and the hidden layer for the context word. For example, the conditioned probability of word $w_c \in V$ occurring in the context is obtained by taking the softmax of the dot product between the center word vector $\vec{v_{w_t}}$ and the weight vector for word w_c, denoted as $\vec{u_{w_c}} \in \mathbb{R}^d$ (Eq. (4.2)). To explain this computation, it is convenient to think of the dot product as a rough similarity measure. A larger dot product indicates a higher similarity between the two vectors, and that in turn yields a higher probability by softmax.

$$p(w_c|w_t) = \frac{e^{\vec{u_{w_c}}^T \vec{v_{w_t}}}}{\sum_{w_k \in V} e^{\vec{u_{w_k}}^T \vec{v_{w_t}}}} \tag{4.2}$$

The parameters in the Skip-gram model consists of two vectors of dimension d for each word in the vocabulary. One vector in the projection layer and one in the weights between the projection layer and the hidden layer. The loss function is the negative log-likelihood (Eq. (4.3)), where $\theta = [u_{w_1}, v_{w_1}, \ldots, u_{w_{|V|}}, v_{w_{|V|}}] \in \mathbb{R}^{2d|V|}$ denotes a concatenated vector of model parameters.

$$L(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log\frac{e^{\vec{u_{w_{t+j}}}^T \vec{v_{w_t}}}}{\sum_{w_k \in V} e^{\vec{u_{w_k}}^T \vec{v_{w_t}}}} \tag{4.3}$$

Having defined the loss function, parameters of the model can be optimized with gradient-based optimization and a learning rate η. Each parameter in θ is updated according to its partial derivative with respect to the loss, $\theta' = \theta - \eta\nabla_\theta$.

4.3.2 The FastText Algorithm - Modeling Subword Information

The Skip-gram model is extended in the FastText algorithm (Bojanowski et al. 2016) by incorporating subword information in the model. In FastText, the

projection layer includes a vector representation for *each n-gram* in the vocabulary. The projected representation of a word is the sum of the vector representations of all of its n-grams. For example, setting $n = 6$, the word "#londonfashion" contains the n-grams:

#londo, london, ondonf, ndonfa, donfas, onfash, nfashi, fashio, ashion

In addition to the regular set of n-grams, FastText uses two special boundary symbols that marks the beginning and end of words, < and >. With these symbols, the model can distinguish prefixes and suffixes from other character sequences. Moreover, a special n-gram that includes the entire word is included. Hence, for the example with word "#londonfashion", three additional n-grams are added:

<#lond, shion>, <#londonfashion>

The updated Skip-gram model used in FastText is defined in Eq. (4.4), where the set of n-grams for word w_t is denoted as $\mathcal{G}_{w_t}$.

$$p(w_c|w_t) = \frac{e^{\sum_{g \in \mathcal{G}_{w_t}} \vec{u_{w_c}}^T \vec{v_{w_g}}}}{\sum_{w_k \in V} e^{\sum_{g \in \mathcal{G}_{w_t}} \vec{u_{w_k}}^T \vec{v_g}}} \tag{4.4}$$

The intention with the n-gram computations in FastText is that it helps the model to learn representations that are shared across words. The subword information improves word generalization for languages with a lot of morphological variation. For instance, by using the FastText approach, the word "#londonfashion" will obtain an embedding representation that is similar to both the word family of "london", and to the word family of "fashion". Thus, even if the literal token "#londonfashion" only occur occasional times in the training corpus, the model can capture its semantics by using the learned representations of its common subwords "london" and "fashion".

4.3.3 FastTextOnSpark

In FastTextOnSpark, for each iteration, the training corpus is split into m partitions $P = \{p_i | p_i = [s_{(i-1)*(T/m)+1}, \ldots, s_{i*T/m}] \wedge i \in 1, \ldots, m\}$, where s_i denotes the i-th sentence in the training corpus. Next, the parameters of the model, θ, are broadcasted to every executor $e_i \in \mathcal{E}$, and the partitions are distributed evenly over the executors. Once this data shuffling is completed, each executor independently performs one pass of Skip-gram training and gradient optimization on its partitions of the dataset. After completing the training, the executors upload their modified parameters $\hat{\theta}^{(i)}$ to the driver d. Finally, the driver then sums the modified embeddings from the executors to update the global parameters θ. When this step is completed, the system proceeds to the next iteration and the procedure is repeated

when the driver re-broadcasts θ. This computation is defined in Eq. (4.5), where $\hat{\theta}^{(i)} = t(e_i, \vec{p_i}, \theta)$ denotes the updated model parameters after Skip-gram training by executor e_i on its assigned partitions $\vec{p_i}$ and parameters θ. θ_k denotes the global model parameters for iteration k. Pseudocode of the algorithm is given in listings 1 and 2.

$$\begin{aligned} \hat{\theta}_k^{(i)} &= t(e_i, \vec{p_i}, \theta_{k-1}) \\ \theta_k &= \theta_{k-1} + \sum_i \hat{\theta}_k^{(i)} \end{aligned} \tag{4.5}$$

Algorithm 1 FastTextOnSpark: Driver node computation

procedure FTOS(D) ▷ Input training corpus D
  $S \leftarrow split(D)$ `/* split corpus into sentences */`
  $V \leftarrow vocab(D)$
  $\theta_0 \leftarrow random(\mathbb{R}^{2d|V|+dn})$ `/* initialize parameters */`
  $P \leftarrow partition(S)$
  $distribute(P)$
  for $i \in \{1, \ldots, iter_{max}\}$ **do** `/* iterative learning */`
    $\vec{\hat{\theta}_i} \leftarrow broadcast(\theta_{i-1}, \mathcal{E})$ `/* broadcast θ to executors */`
    $\theta_i \leftarrow \theta_{i-1} + \sum \hat{\theta}_i^j \quad \forall \hat{\theta}_i^j \in \vec{\hat{\theta}_i}$ `/* update model */`
  end for
  return $\theta_{iter_{max}}$ ▷ Learned embeddings
end procedure

Algorithm 2 FastTextOnSpark: Executor node computation

procedure FTOS($\vec{p_j}$, j) ▷ Input partitions and executor id
  for $i \in \{1, \ldots, iter_{max}\}$ **do**
    $\theta_{i-1} \leftarrow receive(\theta_{i-1})$ `/* receive parameters from driver*/`
    $\hat{\theta}_i^j \leftarrow t(e_j, \vec{p_j}, \theta_{i-1})$ `/* skip-gram training*/`
    $send(\hat{\theta}_i^j)$ `/* send modified embeddings to driver */`
  end for
end procedure

Implementation Details of FastTextOnSpark To make local training at each executor more efficient, the implementation makes use of a handful of well known optimizations. Hierarchical softmax (Morin and Bengio 2005) is used to approximate the computationally expensive softmax operation over large vocabularies. Moreover, the implementation hashes n-grams to bound memory requirements (Bojanowski et al. 2016), and uses efficient implementations of linear algebra operations from the BLAS library (Blackford et al. 2002). Finally, the implementation also holds a pre-

computed cache of exponentials in physical memory to speed up the computation of sigmoid operations used in hierarchical softmax.

Fault-tolerance is provided by the underlying Spark platform. Spark performs the computations in memory and models the computation as a lineage graph, with explicit dependencies between sub-tasks. By checkpointing the lineage to disk periodically, failures can be recovered quickly by re-computation, without risking inconsistencies and without having to rely on replication, for details see (Zaharia et al. 2010).

Scalability Analysis The implementation has two conceivable bottlenecks. The first bottleneck scenario is lack of memory. Although the computation in FastTextOnSpark can scale with the number of machines, every machine must be able to fit all word embeddings in physical memory to achieve tolerable training times. To exemplify, with a vocabulary of words V, a vector dimension d, model parameters θ, and using 4-byte floating point numbers, the baseline Word2vec implementation requires $2 \cdot d \cdot 4 \cdot |\mathcal{V}|$ bytes of RAM for every machine to be able to fit $\theta \in \mathbb{R}^{2d|V|}$ in memory. Similarly, FastText requires $2 \cdot d \cdot 4 \cdot |\mathcal{V}| + 4 \cdot d \cdot n$ bytes to fit $\theta \in \mathbb{R}^{2d|V|+dn}$ in memory, where n is the number of n-grams.

The second potential bottleneck of FastTextOnSpark is network bandwidth. The implementation follows the Bulk Synchronous Parallel (BSP) framework (Valiant 1990). In this framework, the computation can be divided into a series of supersteps, one for each iteration. The BSP model makes the execution free of race conditions and deadlocks. However, the BSP model is sensitive to stragglers and network communication.

In the sequel, I give an asymptotic time complexity analysis of FastTextOnSpark. The training corpus is denoted as D, the number of iterations as k, and the number of executors as $|\mathcal{E}|$. C refers to the total context window size, n denotes to the number of n-grams, V is the vocabulary of words, and d is the dimension of embeddings.

Computation Time The time complexity when $k >> 1$ (startup complexity is ignored) for the driver node is:

$$\underbrace{\mathcal{O}(3D + 2d|V| + dn)}_{\text{constant time}} + \underbrace{\mathcal{O}(k|\mathcal{E}|(2d|V| + dn))}_{\text{iterative}} = \mathcal{O}(k|\mathcal{E}|(2d|V| + dn))$$

Here I use the worst case analysis. Meaning that I assume that every executor modifies all parameters in the model at every iteration, which entails that the driver needs to sum over $|\mathcal{E}|$ complete parameter sets for each iteration. Similarly, the time complexity for executors is $\mathcal{O}(k\frac{D}{|\mathcal{E}|}C(d + d\log_2|V|))$. Where $\mathcal{O}(d + d\log_2|V|)$ represents the complexity of hierarchical softmax.

Communication Cost As depicted in Fig. 4.4, the distributed training in FastTextOnSpark makes use of two costly network transfers: (1) sending the modified

parameters from every executor to the driver node; and (2) broadcasting updated model parameters from the driver to the executors. In the worst case analysis, all executors modify every parameter in every iteration. Thus, the communication cost is $\mathcal{O}(k2|\mathcal{E}|(2d|V| + dn))$.

Communication Time The communication time depends on the available bandwidth. Without loss of generality, I assume that the time to send $2 \cdot d \cdot 4 \cdot |\mathcal{V}| + 4 \cdot d \cdot n$ bytes between the driver and every executor takes p time. The messages from executors to the driver are done in parallel (assuming no stragglers) and hence the effective communication time is in the order of $\mathcal{O}(2kp)$.

Proposition 4.3.1. *Let $|\mathcal{E}|$ denote the number of executors and let p denote the network latency. Assume that the model parameters $\theta \in \mathbb{R}^{2d|V|+dn}$ fits in memory of each machine, and that the effect of stragglers is negligible. Then, if startup complexity is ignored,* FastTextOnSpark *gives an asymptotic speedup as $D \rightarrow \infty \wedge |V| \not\rightarrow \infty$ in the order of $\mathcal{O}(|\mathcal{E}|)$ compared to the single machine implementation, minus the added communication cost of $\mathcal{O}(2kp)$.*

Proof. With the assumptions and notation used above, the complexity for the single machine implementation is $\mathcal{O}(kDC(d + d \log_2 |V|)$, denoted as T_1. Similarly, the complexity of FastTextOnSpark, denoted as $T_{|\mathcal{E}|}$, is on the order of $\mathcal{O}(k|\mathcal{E}|(2d|V| + dn)) + \mathcal{O}(k\frac{D}{|\mathcal{E}|}C(d + d \log_2 |V|))$. As $D \rightarrow \infty$, the leading term is $\mathcal{O}(k\frac{D}{|\mathcal{E}|}C(d + d \log_2 |V|)$. Then it follows that in the limit, $\frac{T_1}{T_{|\mathcal{E}|}} = |\mathcal{E}|$. Thus, the theoretical speedup as $D \rightarrow \infty$ is on the order of $\mathcal{O}(|\mathcal{E}|)$ minus a communication cost of $\mathcal{O}(2kp)$. This is essentially an example of Gustafson's law (Gustafson 1988), where the computation and communication time is constant and the problem size is variable. □

4.3.4 Experimental Setup for Distributed Training of Word Embeddings Using FastTextOnSpark

This section describes the experiments with FastTextOnSpark and how the implementation was deployed on a cluster of machines to evaluate its empirical scalability.

Evaluation Metrics FastTextOnSpark was evaluated to measure its empirical scalability and how well it preserves the embeddings' accuracy despite the distribution. The implementation was compared to the official C++ implementation of FastText[8]. To measure the quality of the embeddings, the WordSim353 (Finkelstein et al. 2002) dataset was used. The empirical scalability was measured by the runtime, the speedup (Eq. (4.6)), and the efficiency (Eq. (4.7)). T_s denotes the

[8] `https://github.com/facebookresearch/fastText`

runtime of the original implementation that is limited to a single machine, and T_n denotes the runtime of FastTextOnSpark with n machines.

$$Speedup_n = \frac{T_s}{T_n} \tag{4.6}$$

$$Efficiency_n = \frac{T_s}{nT_n} \tag{4.7}$$

Cluster Configuration All experiments with FastTextOnSpark were run on a medium-loaded Spark cluster[9]. The experiments used a variable number of executors, where each executor was allocated 8 CPU cores and 76GB RAM. The dataset used for training was a corpus of text from English Wikipedia that consists of 371M tokens, with a vocabulary of size 430K. The Wikipedia corpus is publicly available[10]. When training, the dataset was stored distributed in HopsFS[11] (Niazi et al. 2016) on the cluster.

For each execution, the number of partitions of the dataset was configured to be 2× the number of total available cores for training, as this was shown to give a desirable balance between parallelism and overhead of task management in Spark. The official C++ implementation was tested on a local machine with 8 CPU cores and 16GB RAM. The FastText parameters used for training with both implementations were the default parameters listed in Table 4.2, with dimension set to 300 and window size set to 8. The implementation, results, dataset, and the commands used for training, are publicly available[12].

4.3.5 Results From Evaluation of FastTextOnSpark

This section contains the results after training word embeddings using FastTextOnSpark on a cluster of machines.

How Scalable is FastTextOnSpark? FastTextOnSpark scales with the number of machines in the cluster (consistent with Proposition 4.3.1). It reduces training time on the test corpora described in Chapter 3 from 15.5 hours using the official C++ implementation to 4.5 hours by distributing the training procedure on 10 machines (Fig. 4.5). However, the results also indicate that the virtue of increased parallelism decreases as the number of executors reach above 8 for this dataset. Moreover, even though increasing the number of machines to train FastTextOnSpark on did not reduce the accuracy, FastTextOnSpark has a slower convergence when compared with the official implementation (Fig. 4.5).

[9] `hops.site`

[10] `https://github.com/shatha2014/FashionRec`

[11] A fork of the Hadoop distributed file system

[12] `https://github.com/shatha2014/FashionRec`

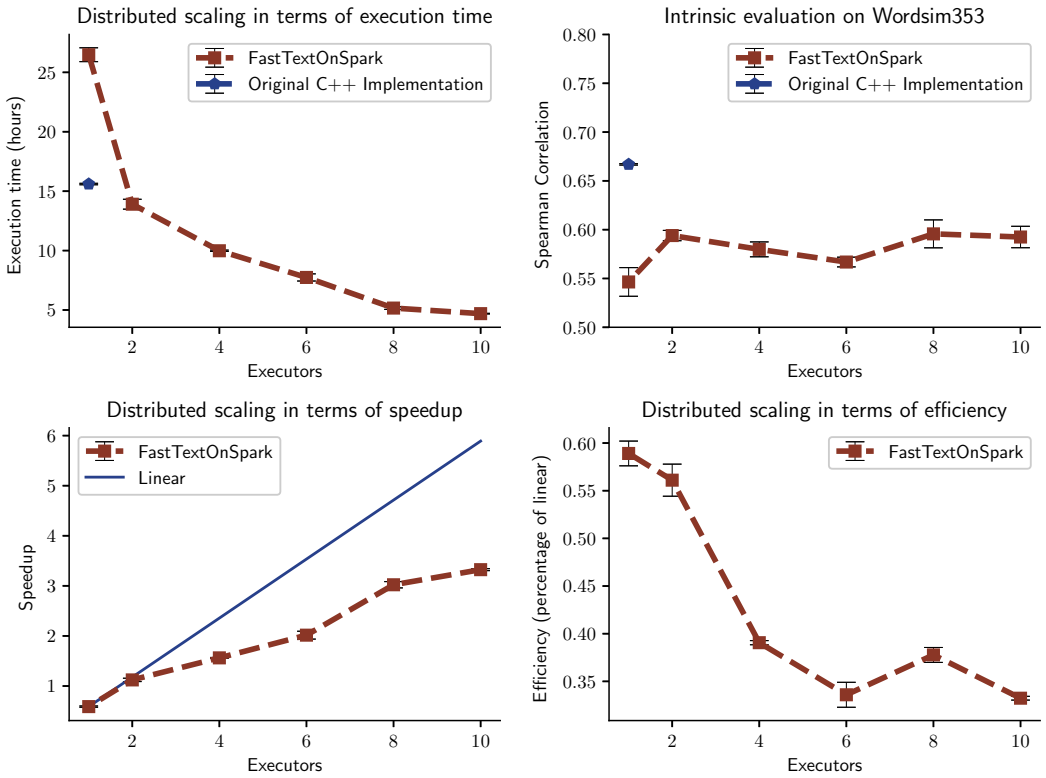

Figure 4.5: Evaluation of FastTextOnSpark in terms of scalability and accuracy (Wordsim353 accuracy with p-value $< 3.6e{-}14$). The number of training iterations and the dataset size was fixed during training. The results are the average of two executions with each configuration.

4.4 Unsupervised Extraction of Fashion Attributes from Instagram

This section presents the workings of a system for unsupervised extraction of fashion attributes from Instagram posts. The system is evaluated on an annotated dataset to measure the accuracy of its extractions.

4.4.1 Fashion Attribute Extraction Using Word Embeddings

I have developed a system for extracting fashion attributes from Instagram posts that is publicly available[13], subsequently referred to as SemCluster. Figure 4.6 illustrates the workings of the system. The extraction in SemCluster is carried out as follows.

[13] `https://github.com/shatha2014/FashionRec`

Table 4.3: Glossary for Eq. (4.8).

Term	*Meaning*
$\mathcal{O}$	Ontology
$\mathcal{P}$	Set of all Instagram posts
p	An Instagram post
$cos(\vec{w_i}, \vec{o_j})$	Cosine similarity between embeddings
$lev(w_i, o_j)$	Levenshtein distance between words
$tfidf(w_i, p, \mathcal{P})$	TF-IDF statistic for word w_i
$h(o_j)$	Probase lookup of ontology term o_j
$t(w_i)$	Term-score of word w_i
γ, η, α	Scaling factors
$\mathcal{V}$	The Vocabulary of word embeddings

Text Normalization To begin with, the text of a single post is tokenized with NLTK's (Loper and Bird 2002) TweetTokenizer, that is designed to recognize text from social media (a tokenizer that can handle online-specific tokens such as emojis, emoticons, and the like). Then the text is normalized by lemmatizing and lowercasing all tokens as well as removing stopwords. Moreover, hashtags, emojis, and user-handles are extracted using regular expressions, and hashtags are segmented using the segmenter presented in (Baziotis, Pelekis, and Doulkeridis 2017).

Ontology Mapping Using Word Embeddings After normalizing the text, it is mapped to a domain ontology that includes fashion brands, items, patterns, materials, and styles. The mapping is based on semantic similarity matching via word embeddings and the cosine similarity metric, as well as syntactic matching through the Levenshtein distance metric (Levenshtein 1966). Furthermore, each word's contribution to the rankings of the categories in the ontology is scaled by its TF-IDF score, and its term-score. Where the term-score has a different weight depending on if the word occurred in the caption, a usertag, a hashtag, or in a comment. After this mapping with the ontology, the k highest ranked entities from the ontology are extracted together with their respective scores.

Ambiguity Resolution The results are re-ranked based on a source of distant supervision, Probase (Wu et al. 2012). Probase is an API that, for a given word, returns an estimated probability that the word has a certain meaning. For instance, the homonym "felt" is both a clothing fabric and a common English word, implying that it will receive a lower rank than a less ambiguous word, like "polyester". Hence, Probase is used by SemCluster to resolve word ambiguities.

Linear Combination The different components in the pipeline are combined in a final ranking $\vec{r}$ through a linear combination defined in Eq. (4.8) using the glossary from Table 4.3. The scaling factors, as well as the number of results to return, k, are hyperparameters that are best determined through experimentation.

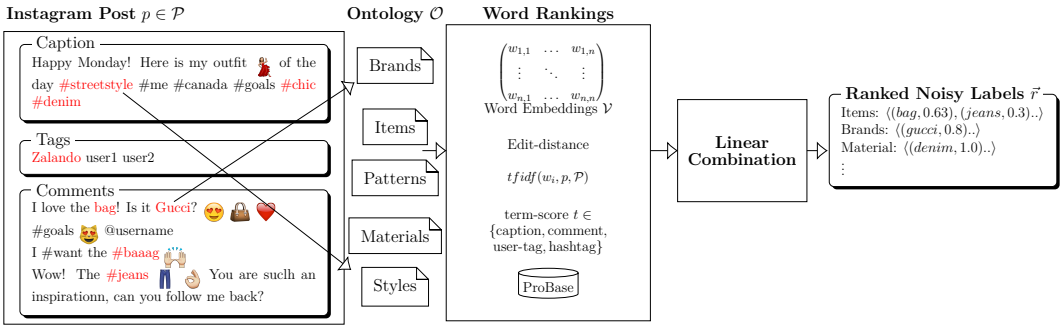

Figure 4.6: SEMCLUSTER, a system for extracting fashion attributes from social media text.

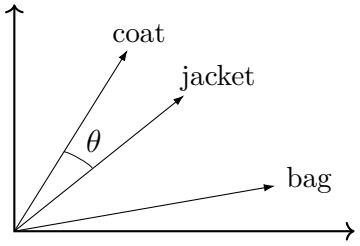

Figure 4.7: Illustrative image of cosine similarity between word embeddings.

$$
\begin{aligned}
&\forall(w_i, o_j) \quad w \in p, o \in \mathcal{O} \\
&r(w_i, o_j) = \begin{cases} t(w_i) + \gamma h(o_j) + \eta(tfidf(w_i, p, \mathcal{P})) + \alpha(cos(\vec{w_i}, \vec{o_j})) & \text{if } w_i \in \mathcal{V} \wedge o_j \in \mathcal{V} \\ t(w_i) + \gamma h(o_j) + \eta(tfidf(w_i, p, \mathcal{P})) + \alpha(lev(w_i, o_j)) & \text{else} \end{cases} \\
&\vec{r} = \underset{s_j}{top_k}(\{(o_j, s_j) | o_j \in \mathcal{O} \wedge s_j = \sum_i r(w_i, o_j)\})
\end{aligned}
\tag{4.8}
$$

Effectively, the information extraction may be seen as a form of clustering, where clusters are seeded with terms from an ontology, and the k most salient clusters are returned and re-ranked based on distant supervision.

4.4.2 Capturing The Semantics of Text with Word Embeddings

SEMCLUSTER uses the cosine similarity metric (Eq. (4.9)) to estimate the similarity between words in text associated with the Instagram post (w_i) and terms in the ontology (o_j). Figure 4.7 depicts an intuitive example of how the similarities of words can be interpreted from the angle between their corresponding embeddings. A smaller cosine of the angle (θ) indicates a higher similarity. Figure 4.7 is an oversimplification by visualizing embeddings of only two dimensions. In practice, two dimensions is not sufficient to capture fine-grained semantics of words. Word embeddings are usually of dimension $\mathbb{R}^{300}$.

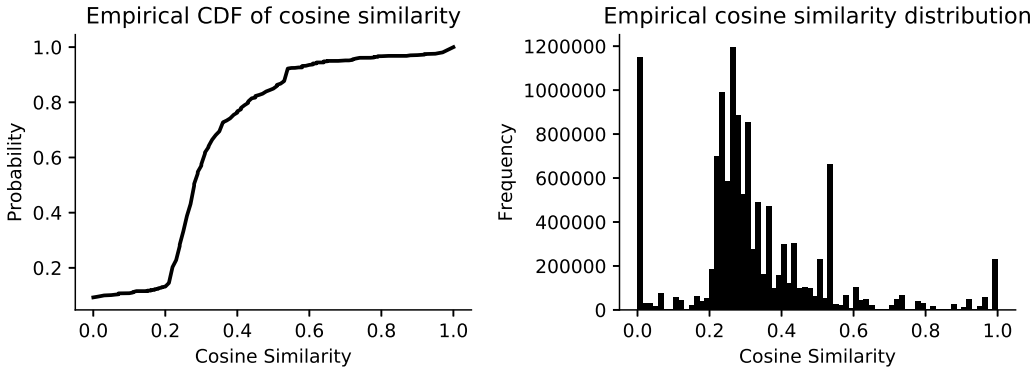

Figure 4.8: The distribution of cosine similarities between a corpora of 70K Instagram posts and an ontology of fashion items.

$$cos(\theta) = \frac{\vec{w_i}\vec{o_j}}{\|\vec{w_i}\|\|\vec{o_j}\|} \tag{4.9}$$

In SemCluster, the word embeddings are used in a way that resembles clustering. The cosine similarity between each word in the input and each term in the ontology are accumulated and then the highest scoring ontology terms are returned. Another approach is to use a similarity threshold for the extraction. However, using a threshold can be brittle depending on how accurate the threshold is.

Figure 4.8 visualizes the distribution of cosine similarities between words in our corpora of Instagram posts and terms in the ontology. The distribution was measured after taking the cosine similarity between each token in a sub-corpora of 70K Instagram posts and terms in the ontology under the category "Clothing items". For each word, the highest cosine similarity out of the similarities with all clothing items in the ontology was used. The embeddings Word2Vec-Fashion were used for this measurement. As can be seen in Fig. 4.8, most similarities are in the range from 0.0 to 0.6. The range $[0, 0.6]$ is likely to represent the similarities of stopwords and other tokens that are not directly related to the ontology. Whereas cosine similarities in the range $(0.6, 1]$ probably indicate words that describe some clothing item. As an example, two cosine similarities computed with the embeddings Word2Vec-Fashion are given in Eq. (4.10).

$$\begin{aligned} cos(\vec{w}_{\text{coat}}, \vec{w}_{\text{trench}}) &= 0.72 \\ cos(\vec{w}_{\text{coat}}, \vec{w}_{\text{amazing}}) &= 0.37 \end{aligned} \tag{4.10}$$

To inspect the semantics of word embeddings, they can be projected to an euclidean space for visualization purposes. Figure 4.9 contains two plots of word embeddings from Word2Vec-Fashion projected with Principal Component Analysis (PCA). It is inevitable that in the process of reducing the dimension, some information about the word semantics is lost. However, as PCA preserves the most significant

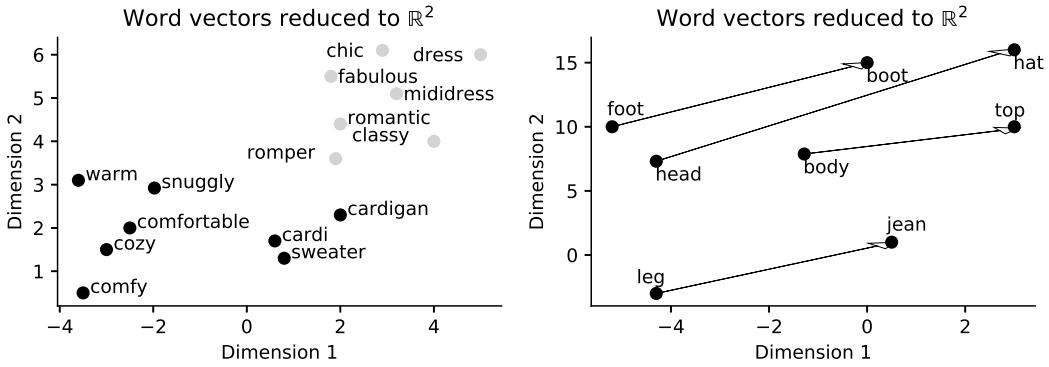

Figure 4.9: Word embeddings trained on the Instagram corpora with Word2vec projected to an euclidean space using PCA.

dimensions of the embeddings, the projected embeddings can still be used to infer coarse grained word semantics, as is demonstrated in Fig. 4.9.

4.4.3 Experimental Setup for Evaluating SemCluster

This section describes the setup that was used to evaluate the information extraction system and the utility of word embeddings for information extraction of Instagram text.

Evaluation & Ontology The system for information extraction was evaluated by comparing its extraction with the ground truth described in Section 3.2. The extractions of SemCluster were evaluated with the metrics precision at k (P@K), average normalized discounted cumulative gain at k (NDGC@K), and mean average precision (MAP). The ontology used in SemCluster consists of fashion-items, brands, clothing-attributes, styles, clothing-patterns, and clothing-fabrics. The ontology was constructed in collaboration with domain experts.

Extrinsic Evaluation An extrinsic evaluation of word embeddings was done with respect to the task of information extraction. The extrinsic evaluation intended to compare domain-specific embeddings to off-the-shelf embeddings. Seven embeddings were compared in this evaluation. The four off-the-shelf embeddings described in Section 4.2.1, and FastText-Fashion, Word2Vec-Fashion, and GloVe-Fashion. The latter set of embeddings consist of word embeddings trained on the Instagram corpora that performed best on the intrinsic evaluation (Fig. 4.3), using FastText, Word2vec, and Glove, respectively. FastText-Fashion was trained using Skip-gram with window size 2 and dimension 300, Word2Vec-Fashion was trained using CBOW with window size 13 and dimension 300, and GloVe-Fashion was trained with window size 12 and dimension 300.

Baseline To highlight the utility of word embeddings for information extraction, the built system, SemCluster, was evaluated against a baseline, that I refer to as SynCluster. The baseline follows the same extraction method as SemCluster except that it uses syntactic matching through Levenshtein distance (Levenshtein 1966), instead of the method with word embeddings used in SemCluster.

Hyperparameters In all of the experiments with SemCluster, the term-score was set to $2, 1, 1, 3$ for caption, comments, tags, and hashtags, respectively. With the motivation that we believe that clothing descriptions provided by the author of a post are more accurate than descriptions that occur in user comments. Moreover, the relative weighting among semantic, syntactic, TF-IDF and Probase was kept equal and k was set to 10.

Significance Testing When comparing two systems for information extraction, a pairwise t-test on the recorded results was made to measure if the difference between the results is significant. The null-hypothesis in the test was that the results were produced by the same system, and that deviations in the results occurred by chance. The significance testing was done against a p-value threshold of 0.05.

4.4.4 Results From Extrinsic Evaluation of Word Embeddings and Evaluation of SemCluster

This section outlines the results from the experiments with SemCluster.

Extrinsic Evaluation: Which Set of Word Embeddings Are Best for Information Extraction? The relative performance of the said embeddings is not consistent across the different sub-tasks of extracting items, styles, patterns, materials, and brands (Table 4.4). For instance, Glove-Fashion do well on the task of extracting styles but worse on the task of extracting patterns. In general, off-the-shelf embeddings performed comparable to domain-specific embeddings. The domain-specific embeddings surpassed the performance of the off-the-shelf embeddings on the tasks of extracting items and styles from the text, while on the tasks of extracting patterns, materials, and brands, the off-the-shelf embeddings did better. For example, Word2Vec-Fashion achieved a MAP score of 0.733 on extraction of clothing items, and a MAP score of 0.373 on the task of extracting clothing materials, while FastText-Wiki achieved a MAP score of 0.696 on extraction of clothing items and a score of 0.441 on extraction of clothing materials. Furthermore, the highest performance overall is achieved on the task of extracting clothing items from the text (best MAP score was 0.733). The task of extracting fashion brands from the text was the most difficult, yielding the lowest scores (best MAP score was 0.203).

Are Word Embeddings Better Than a Syntactic Baseline for Information Extraction? A comparison between SemCluster and the baseline, Syn-

Table 4.4: Extrinsic evaluation of word embeddings. Significant performance degradation for off-the-shelf embeddings in comparison with the domain-specific counterpart (same algorithm) is denoted with $(-)$, with p-value ≤ 0.05.

Embeddings/Category	*NDGC@1*	*NDGC@3*	*NDGC@5*	*NDGC@10*	*P@1*	*P@3*	*P@5*	*P@10*	*MAP*
FastText-Wiki/Item	0.789	0.626	0.655	0.767	0.789	0.513	0.397	0.268$^-$	0.696$^-$
Word2Vec-GNews/Item	0.795	0.629$^-$	0.647$^-$	0.748$^-$	0.795	0.514$^-$	0.386	0.266$^-$	0.697$^-$
GloVe-Wiki/Item	0.789	0.598	0.617	0.746	0.789	0.482	0.369	0.271	0.670
GloVe-Twitter/Item	0.789	0.576	0.592	0.726	0.789	0.461	0.35	0.267	0.662
FastText-Fashion/Item	0.825	0.652	0.683	0.796	0.825	**0.546**	0.429	0.299	0.714
Word2Vec-Fashion/Item	**0.833**	**0.658**	**0.691**	**0.807**	**0.833**	**0.546**	**0.454**	**0.309**	**0.733**
Glove-Fashion/Item	0.773	0.574	0.598	0.733	0.773	0.466	0.366	0.273	0.655
FastText-Wiki/Style	0.367	0.480	0.496	0.516	0.367	0.187	0.120	0.066	0.499
Word2Vec-GNews/Style	0.367	0.436	0.472	0.508	0.367	0.161$^-$	0.114	0.068	0.484$^-$
GloVe-Wiki/Style	0.398$^-$	0.505$^-$	0.515	0.529	0.398$^-$	0.193	0.120	0.065	0.519$^-$
GloVe-Twitter/Style	0.414	0.503$^-$	0.509$^-$	0.538	0.414	0.187	0.116	0.067	0.521$^-$
FastText-Fashion/Style	0.407	0.520	0.539	0.550	0.417	0.207	0.143	0.071	0.528
Word2Vec-Fashion/Style	0.399	0.505	0.519	0.548	0.417	0.204	0.139	0.069	0.539
Glove-Fashion/Style	**0.496**	**0.571**	**0.586**	**0.589**	**0.496**	**0.231**	**0.148**	**0.074**	**0.565**
FastText-Wiki/Pattern	0.047	0.052	0.089$^-$	0.300$^-$	0.047	0.031	0.043	0.098	0.181
Word2Vec-GNews/Pattern	0.110	0.156	0.366	0.447	0.110	0.089	0.178	**0.118**	0.298
GloVe-Wiki/Pattern	0.094	0.135	0.242	0.421	0.094	0.081	0.109	**0.118**	0.268
GloVe-Twitter/Pattern	**0.150**	**0.322**	**0.461**	**0.519**	**0.150**	**0.184**	**0.189**	**0.118**	**0.385**
FastText-Fashion/Pattern	0.091	0.078	0.287	0.447	0.091	0.064	0.071	**0.118**	0.284
Word2Vec-Fashion/Pattern	0.087	0.179	0.353	0.444	0.087	0.110	0.169	**0.118**	0.296
Glove-Fashion/Pattern	0.094	0.100	0.313	0.422	0.094	0.060	0.157	**0.118**	0.265
FastText-Wiki/Material	**0.477**	**0.397**	**0.403**	**0.456**	**0.477**	**0.339**	**0.245**	**0.181**	**0.441**
Word2Vec-GNews/Material	0.125$^-$	0.183$^-$	0.204$^-$	0.264$^-$	0.125$^-$	0.177$^-$	0.148$^-$	0.117$^-$	0.282$^-$
GloVe-Wiki/Material	0.203	0.237	0.284	0.350	0.203	0.214	0.202	0.147	0.343
GloVe-Twitter/Material	0.391	0.297	0.359	0.446	0.391	0.245	0.234	0.174	0.403
FastText-Fashion/Material	0.454	0.378	0.386	0.435	0.384	0.313	0.244	0.168	0.422
Word2Vec-Fashion/Material	0.296	0.286	0.324	0.393	0.286	0.264	0.233	0.165	0.373
Glove-Fashion/Material	0.163	0.222	0.286	0.382	0.163	0.233	0.186	0.169	0.347
FastText-Wiki/Brand	0.062	0.064	0.077	0.046	0.062	0.051	0.029	0.034	0.199
Word2Vec-GNews/Brand	**0.078**	**0.094**	**0.096**	**0.079**	**0.078**	**0.078**	**0.038**	**0.049**	**0.203**
GloVe-Wiki/Brand	0.046	0.053	0.058	0.035	0.046	0.046	0.013	0.039	0.191
GloVe-Twitter/Brand	0.030	0.051	0.083	0.035	0.030	0.046	0.019	0.019	0.188
FastText-Fashion/Brand	0.062	0.061	0.062	0.064	0.032	0.051	0.026	0.039	0.194
Word2Vec-Fashion/Brand	0.062	0.066	0.062	0.064	0.032	0.056	0.036	0.039	0.194
Glove-Fashion/Brand	0.030	0.039	0.040	0.040	0.000	0.011	0.013	0.025	0.176

Cluster, is presented in Table 4.5. The embeddings Word2Vec-Fashion, that achieved among the best results in the extrinsic evaluation (Table 4.4), were used in SemCluster for this experiment. SemCluster beats the baseline in four out of the five sub-tasks when using Word2Vec-Fashion, and on all sub-tasks when taking into account the best set of embeddings for each sub-task.

4.4.5 Error Analysis

The main cause of error in the extraction is text sparsity. Since the system relies solely on text for information extraction, its performance degrades when the text is insufficient. The aforementioned problem is the main reason that extracting brands is harder than extracting clothing items, as brands are rarely mentioned in the text. Additionally, before introducing Probase for word disambiguation, extraction of homonym words was an issue.

The baseline, SynCluster, performs comparable with SemCluster on posts

Table 4.5: Performance comparison between SemCluster and SynCluster. Significant performance degradation of the baseline, SynCluster, in comparison to SemCluster is denoted with $(-)$, with p-value ≤ 0.05.

Method/Category	*NDGC@1*	*NDGC@3*	*NDGC@5*	*NDGC@10*	*P@1*	*P@3*	*P@5*	*P@10*	*MAP*
SemCluster/Item	**0.833**	**0.658**	**0.691**	**0.807**	**0.833**	**0.546**	**0.454**	**0.309**	**0.733**
SynCluster/Item	0.781	0.581^{-}	0.607^{-}	0.767^{-}	0.781	0.474^{-}	0.370^{-}	0.296	0.641^{-}
SemCluster/Style	**0.399**	**0.505**	**0.519**	**0.548**	**0.417**	**0.204**	**0.139**	**0.069**	**0.539**
SynCluster/Style	0.367	0.415^{-}	0.425^{-}	0.507	0.367	0.130^{-}	0.123	0.069	0.474^{-}
SemCluster/Pattern	0.087	0.179	0.353	0.444	0.087	0.110	0.169	**0.118**	0.296
SynCluster/Pattern	**0.108**	**0.413**	**0.498**	**0.512**	**0.108**	**0.221**	**0.193**	**0.117**	**0.395**
SemCluster/Material	**0.296**	**0.286**	**0.324**	**0.393**	**0.286**	**0.264**	**0.233**	**0.165**	**0.373**
SynCluster/Material	0.113^{-}	0.104^{-}	0.137^{-}	0.209^{-}	0.113^{-}	0.107^{-}	0.109^{-}	0.092^{-}	0.227^{-}
SemCluster/Brand	**0.062**	**0.066**	**0.062**	**0.064**	**0.032**	**0.056**	**0.036**	**0.039**	**0.194**
SynCluster/Brand	0.016	0.010	0.010	0.010	0.016	0.005	0.003	0.002	0.159

that contain words that have direct mappings to words in the fashion ontology. However, for posts where the clothing details is not as obvious to infer from the text, the performance of SynCluster degrades in comparison with SemCluster. The intuition behind this result is that for Instagram posts where the text comprise of "noisy" tokens that describe clothing attributes, the word embeddings are still able to relate the text pretty well to the ontology. On the other hand, Levenshtein distance (Levenshtein 1966) is less useful when the text has no syntactic relation to the ontology.

In this chapter I have empirically analyzed a corpora of Instagram posts. The chapter have also presented an evaluation that compares word embeddings trained on text from Instagram posts with off-the-shelf embeddings trained on newswire text. Furthermore, the chapter described a distributed implementation of the FastText algorithm that was benchmarked against the original FastText implementation.

To extract fashion attributes from Instagram posts I took a practical approach and developed an unsupervised information extraction system for Instagram posts that uses a domain ontology. The system was described and evaluated in this chapter. In its extractions, the system accounts for the text structure on Instagram, the semantic and syntactic meaning of noisy text, and the ambiguity of natural language in a coherent way. The key idea to take away from this chapter is that word embeddings in combination with domain knowledge can enable information extraction on noisy text without supervision.

Chapter 5

Deep Text Classification with Weak Supervision

The information extraction system presented in Chapter 4 uses complex feature engineering and depends on domain knowledge. This chapter explores methods to learn from raw data. Section 5.1 introduces the classification task under study, and Section 5.2 introduces a deep weakly supervised text classifier for Instagram posts.

5.1 The Classification Task

Although multiple classifications are of interest in our research, such as brand classification, and fabric classification, we focus initially on the clothing item classification problem. This task is a multi-label multi-class classification problem with 13 classes. The classes are as follows: dresses, coats, blouses & tunics, bags, accessories, skirts, shoes, jumpers & cardigans, jeans, jackets, tights & socks, tops & t-shirts, and trouser & shorts.

5.2 Deep Clothing Classification of Text using Data Programming

This section presents a pipeline for weakly supervised classification that I have applied to our corpora of Instagram posts. The pipeline includes steps devoted to labeling a dataset with weak supervision (Section 5.2.1), combining weak labels with data programming to produce probabilistic labels (Section 5.2.2), and training a discriminative model using the probabilistic labels (Section 5.2.3).

5.2.1 Weak Supervision for Fashion Attributes in Instagram Posts

I used seven labeling functions to label a dataset of 30K Instagram posts with fashion attributes. The purpose of using several functions is that I expect that the

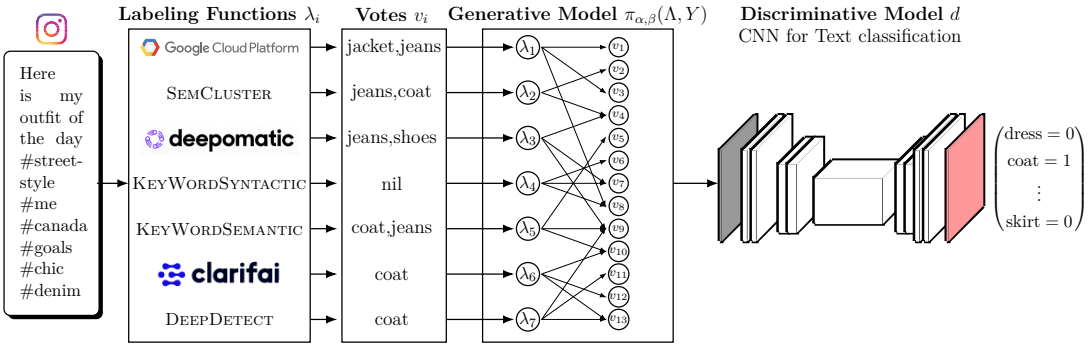

Figure 5.1: A pipeline for weakly supervised text classification.

combination of functions will improve the accuracy of the supervision compared to what each function in isolation would provide. The functions are as follows.

1. λ_1, a function that uses Google's Cloud Vision API[1] to classify the image associated with the text.

2. λ_2, the system for information extraction, SemCluster.

3. λ_3, a function that uses the Deepomatic[2] API for computer vision to classify the image associated with the text.

4. λ_4, a function that uses keyword matching to the fashion ontology with Levenshtein distance (Levenshtein 1966).

5. λ_5, a function that uses keyword matching to the fashion ontology with the word embeddings Word2Vec-Fashion.

6. λ_6, a function that uses the Clarifai "Apparel" model[3] to classify the image associated with the text.

7. λ_7, a function that uses a pre-trained image-classifier provided by DeepDetect[4].

It should be clear that the kind of supervision provided by the aforementioned labeling functions is scalable and extremely cheap in comparison with supervision in the form of human annotations. Figure 5.1[5] illustrates the pipeline I have used for training a weakly supervised text classifier using this method.

[1] `https://cloud.google.com/vision/`
[2] `https://www.deepomatic.com/`
[3] `https://www.clarifai.com/`
[4] `https://www.deepdetect.com/`
[5] The logos of Instagram, Google cloud platform, Deepomatic, and Clarifai are registered trademarks of the respective companies.

5.2.2 Combining Weak Multi-Labels with Data Programming

In the original data programming paper, a binary classification scenario is studied and it is assumed that labeling functions are binary (A. J. Ratner et al. 2016). The labeling functions presented in this thesis differ from the binary model by having several outputs, rather than a scalar output, as defined in Eq. (5.1). In the notation used, C denotes the set of classes, -1 denotes a negative label, 0 means "no label", and 1 denotes a positive label.

$$\lambda_j(x_i) = \vec{z} \in \mathbb{R}^{|C|} \wedge z_k \in \{-1, 0, 1\} \tag{5.1}$$

To make use of the data programming paradigm for multi-label classification, I model the labeling process with one generative model for each class. With this approach, the combination of generative models is able to represent separate accuracy estimates of the labeling functions for each class.

Formally, the generative model $\pi_{\alpha,\beta}(\Lambda^{(k)}, Y^{(k)})$ is trained using the observed overlaps between the labeling functions applied to the unlabeled data for class k. In this notation, $\Lambda^{(k)}_{i,j} = (\lambda_j(x_i))_k$, and $Y^{(k)}$ is the truth labels for class k, modeled as latent variables. Once trained, the parameters learned by the generative models are used to produce probabilistic labels $p(Y^{(k)}|\Lambda^{(k)}) \in \mathbb{R}^n \wedge p(Y^{(k)}|\Lambda^{(k)})_i \in [0, 1]$, for each class k and training example $i \in \{1, \dots n\}$. The probabilistic labels for each class then constitute as column vectors in a matrix of probabilistic labels $p(Y|\Lambda) \in \mathbb{R}^{n \times |C|}$, that can be used to train a multi-label classifier (Eq. (5.2)).

$$p(Y|\Lambda) = \begin{pmatrix} p(Y^{(1)}|\Lambda^{(1)})_1 & \dots & p(Y^{(|C|)}|\Lambda^{(|C|)})_1 \\ \vdots & \ddots & \vdots \\ p(Y^{(1)}|\Lambda^{(1)})_n & \dots & p(Y^{(|C|)}|\Lambda^{(|C|)})_n \end{pmatrix} \tag{5.2}$$

In my experiments, I used the Snorkel[6] implementation (A. Ratner et al. 2017) to train the generative models on the unlabeled data. For completeness, the definition of the training procedure in Snorkel is presented below.

First, the labeling functions are applied to the unlabeled data $\Lambda^{(k)}_{i,j} = (\lambda_j(x_i))_k$. Then the generative model is encoded by using a vector $\phi^{(k)}_i(\Lambda^{(k)}, Y^{(k)})$ of factors for each unlabeled data point x_i and class k. The vector contains concatenated values representing the labeling propensity (encoded with a 1 for each labeling function that labeled x_i), estimated accuracy of each labeling function (encoded with a 1 if the function agrees with the estimated label), and pairwise correlations of labeling functions (a 1 is added if two functions agree with each other). Using these vectors for each data point and labeling function, as well as a vector of model parameters $w^{(k)}$, the model can be defined as in Eq. (5.3) (ibid.). Where $Z_{w^{(k)}}$ is a normalizing constant, and m is the number of unlabeled data points.

$$p_{w^{(k)}}(\Lambda^{(k)}, Y^{(k)}) = Z^{-1}_{w^{(k)}} exp\left(\sum_{i=1}^{m} (w^{(k)})^T \phi^{(k)}_i(\Lambda^{(k)}, y^{(k)}_i)\right) \tag{5.3}$$

[6] `https://github.com/HazyResearch/snorkel`

The parameters of the model $w^{(k)}$ are learned by minimizing the negative log marginal likelihood based on $\Lambda^{(k)}$ and the latent variables $Y^{(k)}$:

$$\hat{w}^{(k)} = \arg\min_{w^{(k)}} - \log \sum_{Y^{(k)}} p_{w^{(k)}}(\Lambda^{(k)}, Y^{(k)}) \tag{5.4}$$

The implementation uses an interleaving of stochastic gradient descent and Gibbs sampling to maximize the objective (A. Ratner et al. 2017). After training, the predictions of the model constitute as the probabilistic labels $p_{\hat{w}^{(k)}}(Y^{(k)}|\Lambda^{(k)})$ for class k.

5.2.3 A Deep Neural Network for Text Classification

I have trained the CNN model for text classification presented in (Kim 2014) using the probabilistic labels produced by the pipeline in Fig. 5.1. This model was used as it is recognized as one of the leading text classifiers on general benchmarks (ibid.). However, nearly any model could have been used, the only requirement is that the loss function can be modified.

The neural network architecture in (ibid.) consists of an embedding input layer, a convolutional layer, and a fully-connected layer of softmax or sigmoid output units. Moreover, the architecture employs max-over-time pooling to detect keywords in the input. The architecture is illustrated in Fig. 5.2 and defined mathematically below.

Embedding and Convolutional Layers Let $p = \langle w_1 \oplus w_2, \oplus \ldots \oplus w_m \rangle \in \mathbb{R}^m$ represent an Instagram post index-encoded with respect to a vocabulary V and padded or chopped off to a fixed length m. The symbol $\oplus$ denotes the concatenation operator, and w_i denotes the i-th word in the concatenated text of caption, usertags, and user comments.

The first layer is the embedding layer, that serves as a lookup step, where each word w_i is encoded as its corresponding word embedding $\vec{w_i} \in \mathbb{R}^d$, and d is the dimension of the embeddings. The embeddings are updated as part of training and can either be initialized randomly or with pre-trained embeddings. Let $x = [\vec{w_1}, \ldots, \vec{w_m}] \in \mathbb{R}^{m \times d}$ denote the output of the embedding layer E.

Next is the convolutional layer that performs two-dimensional convolutions over the sequence of embeddings. The convolutional layer consists of n filter windows $W_1 = [r_1, \ldots, r_n]$ of variable sizes $[k_1, \ldots, k_n]$, $r_i \in \mathbb{R}^{d \times k_i}$. Each filter is slided across the input, x, to produce new *local* feature representations $[c_1, \ldots, c_n]$, called *feature maps*. During every step when traversing the input, filter r_i is applied to a k-gram of length k_i. For each k-gram, the filter applies a non-linearity φ (such as Rectified Linear Unit (ReLU) or tanh) to the weighted sum of the embeddings of the k-gram and the filter weights plus a bias term b_1, producing a scalar output v_j.

Let $x_{i:i+k-1} \in \mathbb{R}^{d \times k}$ denote a k-gram of consecutive words encoded as their embeddings $\vec{w_i}, \vec{w_{i+1}}, \ldots \vec{w_{i+k-1}}$. With this notation, a filter of size k that is slided over the entire input of m words will cover the k-grams $G =$

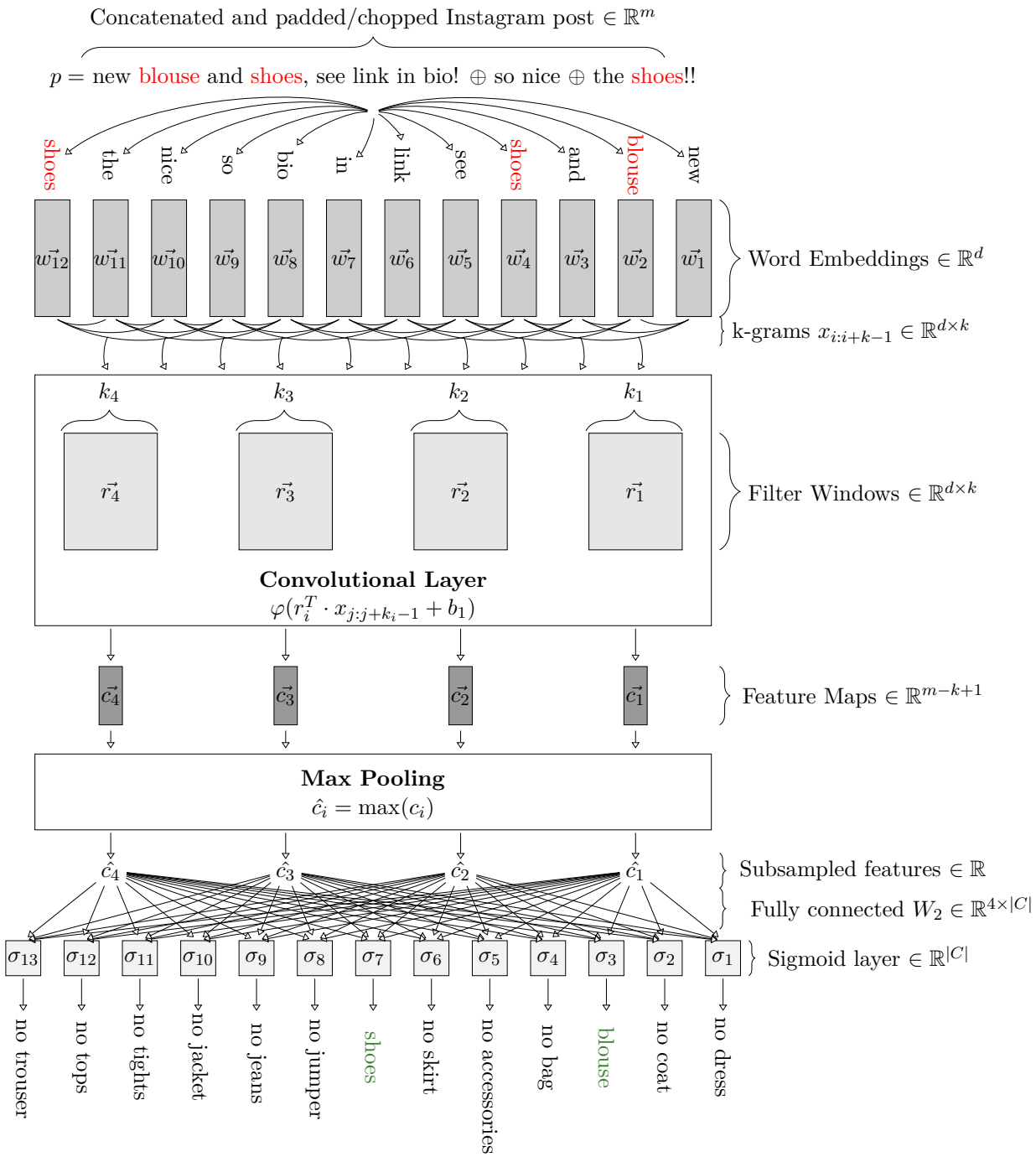

Figure 5.2: CNN for multi-label text classification.

$\{x_{1:k}, x_{2:k+1}, \ldots, x_{m-k+1:m}\} \wedge |G| = m - k + 1$. Following these definitions, the steps to compute a feature map $c_i \in \mathbb{R}^{|G|}$ with filter r_i can be defined as in Eq. (5.5).

$$\begin{aligned} v_j &= \varphi(r_i^T \cdot x_{j:j+k_i-1} + b_1) \\ c_i &= [v_1, \ldots, v_{|G|}] \end{aligned} \tag{5.5}$$

Output and Loss Layer After the convolutional layer, max-over-time pooling (Collobert, Weston, et al. 2011) is applied to the feature maps. The max-over-time pooling yields new subsampled features $[\hat{c_1}, \ldots, \hat{c_n}]$, one for each of the n filters, where $\hat{c_i} = \max(c_i)$. Intuitively, this operation captures the most significant features. Finally, these features are input to a fully-connected layer of $|C|$ softmax or sigmoid output units, one for each class.

The original architecture in (Kim 2014) is designed for the multi-class setting and uses a softmax output layer. I have extended the network to the multi-label setting working with probabilistic labels by switching out the loss function with a noise-aware loss function for multi-label classification. The loss is defined as the cross-entropy over sigmoid outputs with respect to probabilistic labels (Eq. (5.6)). $|C|$ is the number of classes, $p(Y^{(k)}|\Lambda^{(k)})$ is the probabilistic labels for class k, σ is the logistic sigmoid function ($\sigma(x) = \frac{1}{1+e^{-x}}$), $W_2 \in \mathbb{R}^{n \times |C|}$ is the weights between the max-pooled features and the output layer, b_2 is a bias term for the fully connected layer, $\vec{z} \in \mathbb{R}^n$ is the vector of max-pooled features, $\hat{y^{(k)}}$ is the logits for class k, and $\theta = \langle E, W_1, W_2, b_1, b_2 \rangle$ denotes the model parameters.

$$\begin{aligned} \vec{z} &= [\hat{c_1}, \ldots, \hat{c_n}] \\ \hat{y} &= W_2^T \vec{z} + b_2 \end{aligned}$$

$$L(\theta) = \frac{1}{|C|} \sum_{k=0}^{|C|} -(p(Y^{(k)}|\Lambda^{(k)}) \log(\sigma(\hat{y^{(k)}})) + ((1 - p(Y^{(k)}|\Lambda^{(k)})) \log(1 - \sigma(\hat{y^{(k)}}))) \tag{5.6}$$

Multi-channel Inputs The described CNN model uses one input channel. However, it can be extended to use multiple input channels. I have experimented with a variant of the model that splits the input into two channels instead of one, as illustrated in Fig. 5.3. Specifically, in this model, the image caption and usertags are concatenated to one input channel p_1, and the user comments are concatenated to a second input channel p_2. In this architecture, there is a separate set of filters for each input channel. By dividing the inputs into two separate channels, the model should easier be able to distinguish between tokens from the author of the Instagram post, and user comments, which can be helpful for the classification.

Model Analysis There are a few key concepts that characterizes the CNN architecture for text classification. Most prevalent is the assumption that a smaller

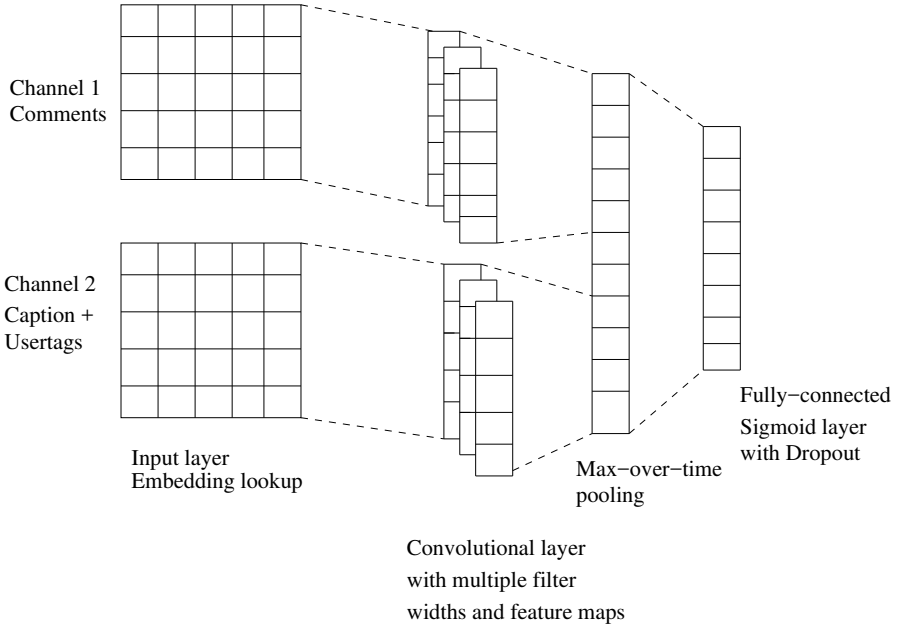

Figure 5.3: CNN model with two input channels.

amount of tokens in the input are decisive for classification. This assumption is expressed both with the max-over-time pooling and by using ReLU activations, that have a sparsity effect on the network. Moreover, since all the neurons inside a single filter share weights, each filter can be seen as a *feature-learner*, that looks for a certain feature in the input. As weights are not shared across filters, increasing the number of filters can allow the network to learn to detect more distinct features in the input. The training procedure will cause the filters to learn different features to minimize the loss. How many filters to use depends on the task. If too many filters are used, some filters typically become so called "dead filters" that never activate and always output zeros.

5.2.4 Experimental Setup for Training and Evaluating Deep Models Using Weak Supervision

This section describes the setup I used to train and evaluate the described CNN models.

Training Dataset When training classifiers, a dataset of 30K Instagram posts annotated with weak labels produced by the labeling functions described in Section 5.2.1 was used. The dataset was split into a 90% train set and a 10% dev set. The train set was used for training and the dev set was used hyperparameter tuning and debugging.

Evaluation Classifiers were evaluated using the annotated dataset (described in Section 3.2) and the metrics, accuracy, precision, recall, F_1, and hamming loss. When the F_1 score is averaged over all training examples it is referred to as micro-averaged. When the F_1 score is computed for each class individually and then averaged, it is referred to as macro-averaged.

Data Programming Compared with Majority Voting To examine how effective data programming is as a method to combine weak labels, it was compared with majority voting. When comparing data programming with majority voting, the base CNN model from (Kim 2014) with a single input channel and randomly initialized embeddings was trained first with probabilistic labels obtained with data programming (CNN-DataProgramming) and then with the majority votes as the labels (CNN-MajorityVote).

Baselines and Variants of the CNN Model As a second experiment, the base CNN model was tweaked and compared with two baselines in the form of the information extraction system (SemCluster), and a human benchmark (DomainExpert). The human benchmark was obtained by taking the average performance on the classification task of three people from our research group. Human test participants were faced with the same task as the other models, namely to classify Instagram posts based solely on the text.

The base CNN model from Kim (ibid.) that uses randomly initialized embeddings and concatenates the text into a single input channel is referred to as CNN. CNN-MultiChannel refers to the CNN model that uses two input channels. CNN-PreTrained refers to a variant of the CNN model trained with a single input channel and *pre-trained* embeddings rather than randomly initialized embeddings. The embeddings Word2Vec-Fashion, that achieved among the best results in the extrinsic evaluation (Table 4.4), were used in both SemCluster and CNN-PreTrained for this experiment. All of the models in this experiment, except the baselines, were trained using weak supervision and probabilistic labels obtained with data programming.

Regularization and Implementation Details The models were regularized with dropout (G. E. Hinton et al. 2012), and weight-decay regularization using the l_2 norm of the weight matrix. For training the models, gradient descent and the Adam optimizer (Kingma and Ba 2014) were used. As the model expects a fixed-size input, shorter texts were padded with zeros and longer texts were chopped down to a fixed size of $m = 2000$ tokens (size selected by experimentation).

Hyperparameters Limited hyperparameter tuning was done prior to the experiments. We used 128 filter windows of size 3, 4, and 5, and a mini-batch size of 256. Moreover we used a vector dimension in the embedding layer of 300 with randomly initialized embeddings updated as part of training. For regularization

Table 5.1: Evaluation of two weakly supervised classifiers. The results are the average of three training runs.

Model	*Accuracy*	*Precision*	*Recall*	*Micro-F_1*	*Macro-F_1*	*Hamming Loss*
CNN-DataProgramming	**0.797** $\pm$ 0.01	**0.566** $\pm$ 0.05	0.678 $\pm$ 0.04	**0.616** $\pm$ 0.02	**0.535** $\pm$ 0.01	**0.195** $\pm$ 0.02
CNN-MajorityVote	0.739 $\pm$ 0.02	0.470 $\pm$ 0.06	**0.686** $\pm$ 0.05	0.555 $\pm$ 0.03	0.465 $\pm$ 0.05	0.261 $\pm$ 0.03

we used a dropout keep probability of 0.7 and a l_2 constraint of 0. Finally, ReLU ($f(z) = \max(0, z)$) was used as the activation function, and the padding strategy was set to VALID and the learning rate to 0.01. The values were chosen based on hyperparameter tuning using random-search on the dev set.

5.2.5 Results from Evaluation of Deep Models Trained with Weak Supervision

This section outlines the results from the experiments with deep text classification of Instagram text using weak supervision. The code used for training is open source[7].

Is Data Programming a Better Way to Combine Weak Labels Than Majority Voting? Table 5.1 compares results from the base CNN model trained with weak labels combined through majority voting, with results from the same model trained with probabilistic labels obtained by using data programming. The data programming approach beats the majority vote model on nearly all metrics.

How Is the Labeling Functions' Accuracy Modeled in the Generative Models? Figure 5.4 visualizes the relative accuracy among labeling functions that was learned by the generative models in CNN-DataProgramming. The keyword-functions were given the highest accuracy overall, indicating that when the keywords are found in the text it tend to be telling for the image contents. This implies that the keyword functions often agrees with the majority in their votes, which in turn gives them a high estimated accuracy. In general, the relative accuracy among labeling functions differed from class to class. The spikes in the accuracy of Clarifai, Deepomatic, and DeepDetect on the classes of "bags" and "shoes" indicate that the APIs' are especially consistent in their predictions on those classes.

Which CNN Model Is Best? and How Do the CNN Models Compare with Baselines? Table 5.2 contains the results after evaluating different variants of the CNN model and comparing the results with the two baselines. The results demonstrate that the CNN model with a single input channel and randomly initialized embeddings (CNN) achieved the highest F_1 score, on par with the human benchmark, beating both CNN-MultiChannel and CNN-PreTrained. All

[7] `https://github.com/shatha2014/FashionRec`

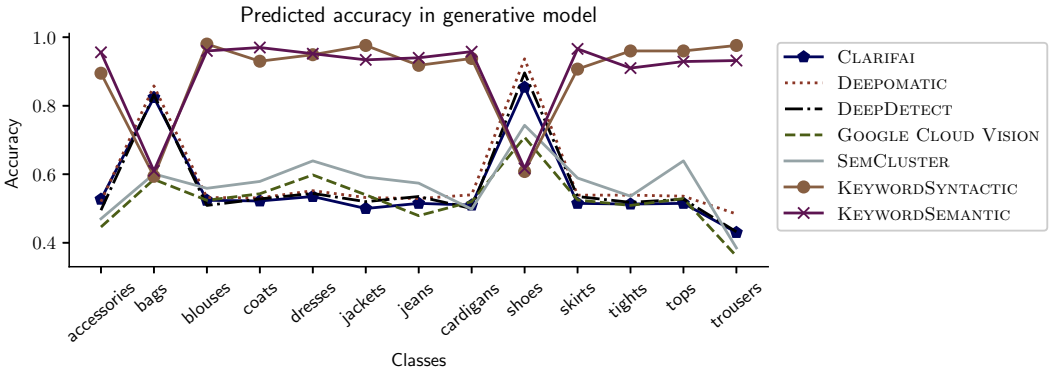

Figure 5.4: Accuracy for labeling functions learned by the generative models.

Table 5.2: Evaluation of several discriminative models for classification, compared against two baselines, SemCluster and DomainExpert. The results of CNN models are the average of three training runs.

Model	*Accuracy*	*Precision*	*Recall*	*Micro-F_1*	*Macro-F_1*	*Hamming Loss*
CNN	0.797 ± 0.01	0.566 ± 0.05	0.678 ± 0.04	$\mathbf{0.616} \pm 0.02$	$\mathbf{0.535} \pm 0.01$	0.195 ± 0.02
CNN-MultiChannel	0.756 ± 0.01	0.512 ± 0.02	0.660 ± 0.03	0.574 ± 0.02	0.490 ± 0.01	0.235 ± 0.03
CNN-PreTrained	0.765 ± 0.02	0.519 ± 0.01	$\mathbf{0.703} \pm 0.03$	0.594 ± 0.03	0.527 ± 0.02	0.226 ± 0.01
SemCluster	0.719	0.541	0.453	0.493	0.438	0.279
DomainExpert	**0.807**	**0.704**	0.529	0.604	0.534	**0.184**

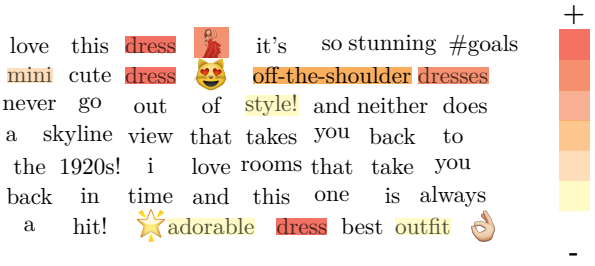

Figure 5.5: Heatmap of a sample Instagram text, where a higher heat indicates a larger logit in the trained CNN model.

CNN models outperformed the baseline SemCluster. The human benchmark had a higher precision but a lower recall than the CNN models.

How Do the CNN Model Make Its Predictions? After training the CNN model, the learned weights of the model can be frozen and used for inference. Figure 5.5 illustrates how the trained CNN model infers clothing items from the text. The heatmap in Fig. 5.5 was produced by running each word in the sample text through the trained model and recording the output scores (logits) for the output class "dresses". After recording the scores, they were normalized and put on a scale and visualized.

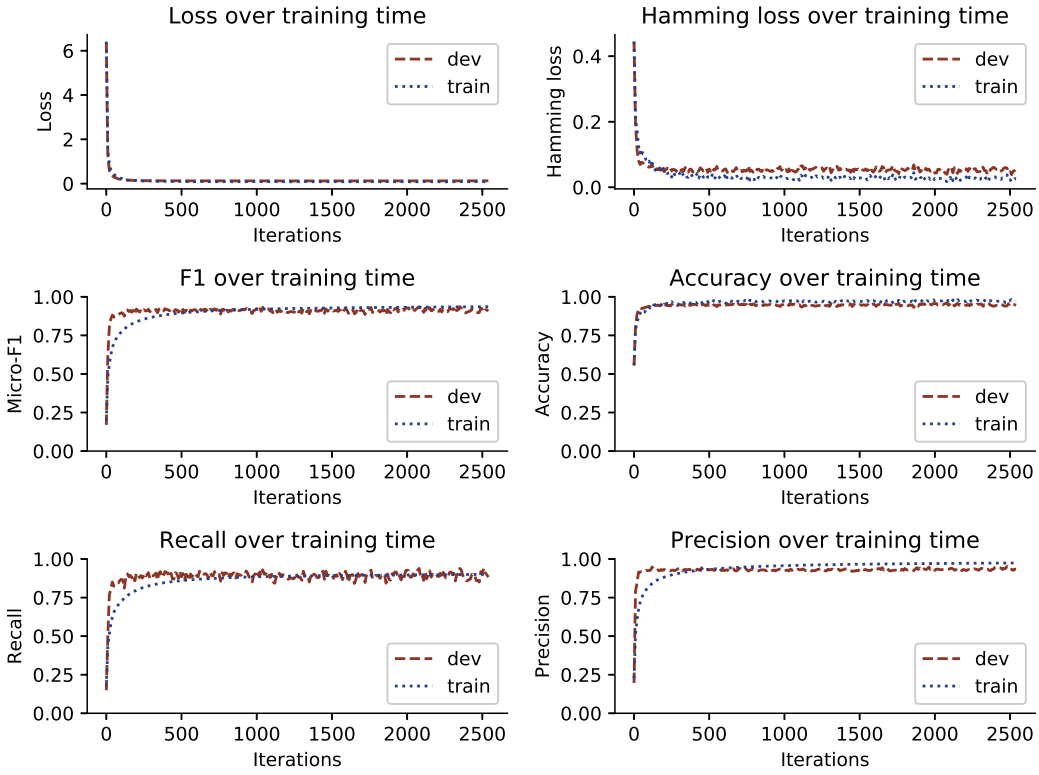

Figure 5.6: Statistics on the dev and train set during training of the CNN model.

5.2.6 Error Analysis

As discussed in Section 4.4.5, a part of the error is attributable to the disparity between the labels in the test set and the text. Since the ground truth is determined based on the image contents, there is an inherent error when information is lacking in the text. This is also evident from the relatively low human benchmark on the task. Moreover, the performance on the dev and train set were significantly better than on the test set for all of the trained models. This is owing to the difference between the weak labels and the ground truth. Figure 5.6 depicts the performance on the dev and train set per training iteration of the CNN model. The performance on the dev and train sets were significantly higher than on the test set. Finally, that CNN-DataProgramming trained with a noise-aware loss function achieved a better result than CNN-MajorityVote indicates that when taking the majority vote for training, potential signals to learn from is lost.

To summarize this chapter, in absence of strong supervision, weak supervision can be used to label large datasets in a cheap and quick manner. This chapter have demonstrated how data programming can be used in a principled way to integrate supervision from open APIs and NLP functions. Moreover, the chapter

has showed how the combined supervision signal can be used to train a deep multi-label text classifier for predicting clothing items in Instagram posts.

Chapter 6

Summary

Recurrent traits of the results presented in this thesis are the informality of text from Instagram, the need for scalable processing methods, and the demand for machine learning methods that do not require strong supervision. In this chapter, I give a summary of the research, the implications of the results, and an outlook on the future. In Section 6.1 I provide my interpretation of the results. Sections 6.2, and 6.3 presents suggestions for applications and future research directions, and finally Section 6.4 outlines the conclusions from this thesis.

6.1 Discussion

I have analyzed an Instagram corpora, performed an intrinsic evaluation of word embeddings for Instagram text, scaled out FastText training, evaluated the utility of word embeddings for information extraction, and trained a weakly supervised model for predicting clothing items in Instagram posts based on associated text. This section contains my interpretation of the results.

What Characterizes Instagram as a Source of Text? In comparison with measurements on Twitter corpora (Baldwin et al. 2013), text from Instagram is just as noisy based on my measurements (Table 4.1). In the results it can be seen that 46% of the tokens in our Instagram corpora are not included in the Google-news vocabulary. This implies that nearly half of the tokens do not have an embedding representation in WORD2VEC-GNEWS. This result illustrates a mismatch between social media text and off-the-shelf word embeddings. Furthermore, when comparing with Twitter, notable is also the high diversity of languages occurring in the comment sections on Instagram and the short length of comments (mean length measured to be 6 tokens).

The long-tail distribution of text on Instagram can be explained with the follower count of the post author and the preferential attachment theory (Zsuzsanna Albert and Barabasi 2001). As an Instagram post attracts a lot of comments, it will get a larger spread on the Instagram platform. This causes a snowball effect, where

a post that already has many comments will be more likely to attract even more comments.

Is Domain-Specific Word Embeddings for Instagram Useful? It was expected that the domain-specific word embeddings would outperform the off-the-shelf embeddings on the FashionSim word similarity task. The FashionSim evaluation is based on fashion concepts and social media tokens which are difficult to capture in large and generic text corpora, such as the corpora that the off-the-shelf embeddings have been trained on. When comparing the state-of-the-art algorithms for training word embeddings, FastText embeddings yielded the most accurate semantics on the intrinsic evaluation. FastText explicitly models the morphology of words by incorporating information about subwords in the embeddings, this is useful for languages that are rich on morphology. According to my results, FastText is also suited for noisy text, as can be found in social media. This is not surprising, as social media language can be characterized as containing a large vocabulary, with many rare words, where the subword embeddings can enhance generalization between words.

The obtained results are specific to our Instagram corpora, and the FashionSim dataset for evaluation. However, similarities can be found in the literature. For example, that CBOW worked best with larger window sizes, whereas Skip-gram achieved the highest accuracy with smaller windows on the intrinsic evaluation is partly in concordance with results reported by Fallgren, Segeblad, and Kuhlmann (2016). Moreover, that all algorithms demonstrated an improved accuracy when the dimension parameter was increased, was expected, as it has been shown in several related results (Pennington, Socher, and Christopher D. Manning 2014; Chiu et al. 2016). Finally, the observed difference between Skip-gram and CBOW was not convincing. With FastText, a slightly better result was obtained with Skip-gram, while with Word2vec a slightly better result was obtained with CBOW. I anticipated that Skip-gram would yield superior results, as it is generally known that Skip-gram works better than CBOW on smaller corpora (Mikolov, Le, and Sutskever 2013). However, this conclusion can not be made from my results. Finally, an unforeseen result from the intrinsic evaluation was that GloVe performed poorly in the intrinsic evaluation, independent of window size.

How Effective are Word Embeddings for Text Mining of Instagram Text? For the extrinsic evaluation I can only speculate why the results turned out as they did. That the off-the-shelf embeddings did well on the tasks of extracting patterns, materials, and brands from the text could be due to the large vocabulary that off-the-shelf embeddings posses. However, as the results on these extraction tasks were so low overall, this result is not very telling and might be due to chance. That the domain-specific embeddings did better on extracting styles and clothing items from the text is likely because those embeddings capture the semantics of more fine-grained fashion concepts, as evident in the intrinsic evaluation (Fig. 4.2).

Overall, the benefit of domain-specific embeddings on the information extraction

task was lower than expected based on the intrinsic evaluation, but in line with the results in (Tixier, Vazirgiannis, and Hallowell 2016). Furthermore, the results do not indicate that one algorithm for training word embeddings is better than another on this information extraction task. Rather, each algorithm achieved the best performance on at least one sub-task. This demonstrates the empirical nature of learning word embeddings, and that there is a lot to gain by finding the right set of embeddings for the task. Another take-away from the extrinsic evaluation is that it contradicts with the intrinsic evaluation, similar to what was observed in (Chiu et al. 2016).

I believe that word embeddings are particularly useful in the social media domain where syntactic word similarity fails to capture the relation between many tokens. This is manifested in the evaluation from Chapter 4 where SemCluster outperformed the baseline, SynCluster, significantly. For instance, using word embeddings trained on the Instagram corpora and the cosine similarity metric, the words most similar to "bag" are "purse", "tote", "clutch", "handbag", "dustbags", "crossbody", and "carryall". This capability of capturing the meaning of words can alleviate the need for curated synonym lists for information extraction.

How Scalable is FastTextOnSpark? The empirical scalability observed in the experiments is consistent with Proposition 4.3.1. Moreover, the results also manifest an upper bound on scalability, as stated by Amdahl's law (Amdahl 1967). Amdahl's law says that the speedup is constrained by the sequential portion of the computation, essentially describing a diminishing return in speedup as more processors are added. In the case of FastTextOnSpark, the sequential portion of the computation is the synchronization of model parameters between training iterations.

Amdahl's law assumes that the *problem size is fixed*. However, FastTextOnSpark scales with the training corpus. Although the quality of embeddings depends on more factors than the corpus size, a common trend is that enlarging the training corpus can improve the quality of embeddings (Lai, Liu, et al. 2015). This implies that the scaled speedup can be arbitrary large in practice (Gustafson 1988). As the training corpus is enlarged, the potential speedup of FastTextOnSpark with respect to the C++ implementation is increased.

The distributed implementation has a slower convergence than the C++ implementation. This is attributable to the less frequent parameter synchronization.

Is Weak Supervision a Competitive Alternative for Doing Text Mining on Instagram? Experimental results demonstrate that a classifier trained with weak supervision in the form of open APIs and keyword detectors is viable. Considering that not all clothing items can be inferred from the text and that the human benchmark on the task is 0.60, the achieved F_1 score of 0.61 is promising. A substantial improvement is to be expected when integrating the text classifier with a model analyzing the image contents.

Combining multiple signals of weak supervision improves the accuracy compared to the baseline system for information extraction. Additionally, when combining the labels by using parameters learned with generative models, rather than majority voting, an increase of six F_1 points was observed. The results are concordant with prior work using data programming (A. J. Ratner et al. 2016; A. Ratner et al. 2017).

With the experiments in this thesis, data programming has been applied to a new domain. In (A. J. Ratner et al. 2016) it is assumed that labeling functions are binary. I propose to extend the base model to the multi-label scenario by learning a separate generative model for each class. Intuitively, labeling functions that produce several labels can have a separate accuracy for each label, which is captured by the proposed method. In my experiments, the relative accuracy of labeling functions differed between classes, strengthening my belief that learning separate generative models for each class is useful.

Unexpectedly, the regular CNN model outperformed both CNN-MultiChannel and CNN-Pretrained. I anticipated the model that used pre-trained embeddings to give a small accuracy boost since that have been reported on similar experiments (Kim 2014). It is hard to say why the model with randomly initialized embeddings outperformed the model with pre-trained embeddings. Both models fine-tune the embeddings as part of training, which might have evened out the differences between the embedding layers. The intuition behind CNN-MultiChannel is that the model should easier be able distinguish between user captions and user comments in the classification. However, splitting the input into two channels was not beneficial in this case. I suspect that one reason for this result is that captions in our corpora tend to be short in comparison with the concatenated user comments.

6.2 Applications

The system for information extraction, SemCluster, is currently being used as hints for human annotators that are crowdsourced via the Amazon Mechanical Turk[1] service for annotating images (Fig. 6.1). In addition, the system is not limited to information extraction of fashion related posts. By exchanging the ontology, the same system can be used to extract information from arbitrary Instagram posts. Further, the weakly supervised classifier presented in Chapter 5 is composable, and the intent is to combine it with a model for computer vision.

6.3 Future Work

The most natural direction for future work is to combine the NLP methods presented in this thesis with a model that analyzes the image contents. Moreover, we are in the process of collecting a larger annotated dataset for training a computer vision

[1] `https://www.mturk.com/`

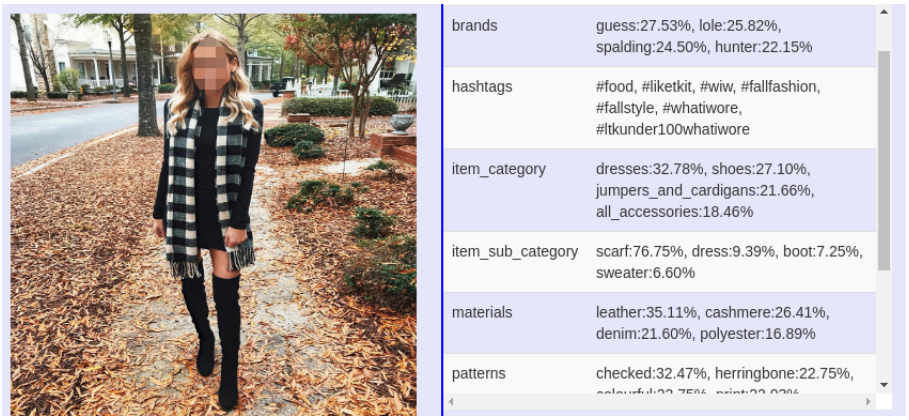

Figure 6.1: Screenshot from the web interface used for crowdworkers. The right panel contains information extracted by SemCluster.

model. The data are collected from crowdsourcing using Amazon Mechanical Turk[2]. When integrating annotations from crowdsourcing, the data programming methods outlined in Section 2.2.3 and extended in Section 5.2.1 can be used.

The word embeddings trained as part of this research can be *retrofitted* to fine-tune word semantics according to some lexicon (Faruqui et al. 2014). Moreover, in future work we might experiment with post-level embeddings rather than learning an embedding for each word. Specifically, I hypothesize post-level embeddings to be useful for the task of predicting the overall clothing style of an Instagram post.

A potential direction for future work on FastTextOnSpark is to move away from the single-driver bottleneck in FastTextOnSpark. The driver's role can be distributed on a cluster of *parameter servers*, similar to the Word2vec training system presented by Ordentlich et al. (2016). Moreover, asynchronous updates is a way to avoid the straggler problem in the current implementation.

Future research to extend on the system for information extraction presented in this thesis includes to experiment with linguistic rules for doing more fine-grained text extraction. This line of work was deprioritized in this research due to the mismatch between standard linguistic rules and noisy text.

When experimenting with text classification models, I focused on the CNN model from Kim (2014), as it is established as one of the best performing models. In future work, it would be interesting to experiment with other models, particularly the deep CNN presented in (Conneau et al. 2016), the bow-CNN model from (Johnson and T. Zhang 2014), and the FastText classifier (Joulin et al. 2016). Moreover, the accuracy of the models used in this thesis can potentially be improved by doing large-scale hyperparameter tuning of the models' hyperparameters.

[2] `https://www.mturk.com/`

6.4 Conclusion

I have quantified a corpora of Instagram posts, demonstrating that the text distribution exhibits the long-tail phenomenon, that the text is just as noisy as have been reported in studies on Twitter text, and that comment sections often contain a multitude of languages. Furthermore, I have presented a system for extracting fashion attributes from social media posts, where word embeddings are a key component in the extraction process. I evaluated word embeddings for the social media domain on both intrinsic and extrinsic tasks, and put the results in relation with off-the-shelf embeddings trained on generic text corpora.

The results demonstrate a mismatch between text in social media and off-the-shelf embeddings. Despite the mismatch, the performance of off-the-shelf embeddings in information extraction from Instagram is remarkable, achieving comparable results with domain-specific embeddings. In summary, the results confirm the belief that word embeddings are able to capture meaningful semantic relationships between words, outperforming a baseline based on Levenshtein distance for the information extraction task.

Using off-the-shelf embeddings is just a glimpse of the many uses of word embeddings. Finding the right set of embeddings for your application is an empirical challenge that includes many parameters to consider, where scalability becomes a necessity. In this context, I have developed and benchmarked a system for distributed training with the FastText algorithm. The implementation scales with the number of machines available for training and provides a significant speedup to the the original single-machine implementation of FastText.

Building upon the system for information extraction, I trained a deep multi-label text classifier using weak supervision in the form of open APIs and keyword detectors. With weak supervision I was able to label a large dataset in hours, something that would have taken months to do with human annotators. The weak supervision signals were combined with the data programming paradigm, which makes for a proof-of-concept of the paradigm in a new domain. Moreover, the original model for binary classification was extended to the multi-label setting by learning a separate generative model for each class. In all measures, combining weak supervision signals with the proposed combination of generative models outperformed a baseline that uses majority voting.

The statement from this thesis is that in absence of annotated data, using large volumes of unlabeled data, domain heuristics, and clever algorithms, is an effective approach to text mining of Instagram data. However, there is still room to expand this area of text mining. To achieve the best results, large data quantities are required, demanding scalable methods. Equally important is empirical research with state-of-the-art solutions in the unsupervised and *weakly* supervised category, which are unproven in comparison with supervised counterparts.

Bibliography

Alter, Nicole (2016). *Four Ways Instagram Is Redefining The Fashion Industry.* https://www.launchmetrics.com/resources/blog/four-ways-instagram-is-redefining-the-fashion-industry. Accessed: 2018-04-09.

Amdahl, Gene M. (1967). "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities". In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference.* AFIPS '67 (Spring). Atlantic City, New Jersey: ACM, pp. 483–485. DOI: 10.1145/1465482.1465560. URL: http://doi.acm.org/10.1145/1465482.1465560.

Baeza-Yates, Ricardo A. and Luz Rello (2011). "How Bad Do You Spell?: The Lexical Quality of Social Media". In: *The Future of the Social Web.* Vol. WS-11-03. AAAI Workshops. AAAI.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *CoRR* abs/1409.0473.

Baldwin, Timothy et al. (2013). "How Noisy Social Media Text, How Diffrnt Social Media Sources?" In: *IJCNLP.* Asian Federation of Natural Language Processing / ACL, pp. 356–364.

Baziotis, Christos, Nikos Pelekis, and Christos Doulkeridis (2017). "DataStories at SemEval-2017 Task 4: Deep LSTM with Attention for Message-level and Topic-based Sentiment Analysis". In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017).* Vancouver, Canada: Association for Computational Linguistics, pp. 747–754.

Bengio, Yoshua et al. (2003). "A Neural Probabilistic Language Model". In: *J. Mach. Learn. Res.* 3, pp. 1137–1155. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=944919.944966.

Berthon, Pierre R. et al. (2012). "Marketing meets Web 2.0, social media, and creative consumers: Implications for international marketing strategy". In: *Business Horizons* 55.3. SPECIAL ISSUE: STRATEGIC MARKETING IN A CHANGING WORLD, pp. 261–271. ISSN: 0007-6813. DOI: https://doi.org/10.1016/j.bushor.2012.01.007. URL: http://www.sciencedirect.com/science/article/pii/S0007681312000080.

Bhargava, Preeti, Nemanja Spasojevic, and Guoning Hu (2017). "Lithium NLP: A System for Rich Information Extraction from Noisy User Generated Text

on Social Media". In: *CoRR* abs/1707.04244. arXiv: 1707.04244. URL: http://arxiv.org/abs/1707.04244.

Bikel, Daniel M. et al. (1997). "Nymble: A High-performance Learning Name-finder". In: *Proceedings of the Fifth Conference on Applied Natural Language Processing*. ANLC '97. Washington, DC: Association for Computational Linguistics, pp. 194–201. DOI: 10.3115/974557.974586. URL: https://doi.org/10.3115/974557.974586.

Blackford, L. S. et al. (2002). "An Updated Set of Basic Linear Algebra Subprograms (BLAS)". In: *ACM Trans. Math. Softw.* 28.2, pp. 135–151. ISSN: 0098-3500. DOI: 10.1145/567806.567807. URL: http://doi.acm.org/10.1145/567806.567807.

Bojanowski, Piotr et al. (2016). "Enriching Word Vectors with Subword Information". In: *CoRR* abs/1607.04606. arXiv: 1607.04606. URL: http://arxiv.org/abs/1607.04606.

Cherry, Colin and Hongyu Guo (2015a). "The Unreasonable Effectiveness of Word Representations for Twitter Named Entity Recognition". In: *HLT-NAACL*. The Association for Computational Linguistics, pp. 735–745.

— (2015b). "The Unreasonable Effectiveness of Word Representations for Twitter Named Entity Recognition." In: *HLT-NAACL*. Ed. by Rada Mihalcea, Joyce Yue Chai, and Anoop Sarkar. The Association for Computational Linguistics, pp. 735–745. ISBN: 978-1-941643-49-5. URL: http://dblp.uni-trier.de/db/conf/naacl/naacl2015.html#CherryG15.

Chiu, Billy et al. (2016). "How to Train good Word Embeddings for Biomedical NLP". In: *BioNLP@ACL*. Association for Computational Linguistics, pp. 166–174.

Collobert, Ronan and Jason Weston (2008). "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: ACM, pp. 160–167. ISBN: 978-1-60558-205-4. DOI: 10.1145/1390156.1390177. URL: http://doi.acm.org/10.1145/1390156.1390177.

Collobert, Ronan, Jason Weston, et al. (2011). "Natural Language Processing (Almost) from Scratch". In: *J. Mach. Learn. Res.* 12, pp. 2493–2537. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1953048.2078186.

Conneau, Alexis et al. (2016). "Very Deep Convolutional Networks for Natural Language Processing". In: *CoRR* abs/1606.01781.

Deerwester, Scott et al. (1990). "Indexing by latent semantic analysis". In: *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE* 41.6, pp. 391–407.

Derczynski, Leon et al. (2013). "Twitter Part-of-Speech Tagging for All: Overcoming Sparse and Noisy Data". In: *RANLP*. RANLP, pp. 198–206.

Dhillon, Paramveer S., Dean P. Foster, and Lyle H. Ungar (2015). "Eigenwords: Spectral Word Embeddings". In: *Journal of Machine Learning Research* 16, pp. 3035–3078. URL: http://jmlr.org/papers/v16/dhillon15a.html.

Fallgren, Per, Jesper Segeblad, and Marco Kuhlmann (2016). "Towards a Standard Dataset of Swedish Word Vectors". In: *Proceedings of the Sixth Swedish Language Technology Conference (SLTC)*. UmeÃ¥, Sweden.

Faruqui, Manaal et al. (2014). "Retrofitting Word Vectors to Semantic Lexicons". In: *CoRR* abs/1411.4166. arXiv: 1411.4166. URL: http://arxiv.org/abs/1411.4166.

Finin, Tim et al. (2010). "Annotating Named Entities in Twitter Data with Crowdsourcing". In: *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*. CSLDAMT '10. Los Angeles, California: Association for Computational Linguistics, pp. 80–88. URL: http://dl.acm.org/citation.cfm?id=1866696.1866709.

Finkelstein, Lev et al. (2002). "Placing search in context: the concept revisited." In: *ACM Trans. Inf. Syst.* 20.1, pp. 116–131.

Firth, J. R. (1957). "A synopsis of linguistic theory 1930-55." In: 1952-59, pp. 1–32.

Gimpel, Kevin et al. (2011). "Part-of-speech Tagging for Twitter: Annotation, Features, and Experiments". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*. HLT '11. Portland, Oregon: Association for Computational Linguistics, pp. 42–47. ISBN: 978-1-932432-88-6. URL: http://dl.acm.org/citation.cfm?id=2002736.2002747.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Google (2013). *word2vec*. URL: https://code.google.com/archive/p/word2vec/ (visited on 02/02/2018).

Gouws, Stephan, Dirk Hovy, and Donald Metzler (2011). "Unsupervised Mining of Lexical Variants from Noisy Text". In: *Proceedings of the First Workshop on Unsupervised Learning in NLP*. EMNLP '11. Edinburgh, Scotland: Association for Computational Linguistics, pp. 82–90. ISBN: 978-1-937284-13-8. URL: http://dl.acm.org/citation.cfm?id=2140458.2140468.

Gustafson, John L. (1988). "Reevaluating Amdahl's Law". In: *Commun. ACM* 31.5, pp. 532–533. ISSN: 0001-0782. DOI: 10.1145/42411.42415. URL: http://doi.acm.org/10.1145/42411.42415.

Habib, Mena Badieh and Maurice van Keulen (2014). "Information Extraction for Social Media". In: *Proceedings of the Third Workshop on Semantic Web and Information Extraction (SWAIE 2014)*. eemcs-eprint-24959. Association for Computational Linguistics, pp. 9–16. ISBN: 978-1-873769-48-5.

Habibi, Maryam et al. (2017). "Deep learning with word embeddings improves biomedical named entity recognition". In: *Bioinformatics* 33.14, pp. i37–i48. DOI: 10.1093/bioinformatics/btx228. eprint: /oup/backfile/content_public/journal/bioinformatics/33/14/10.1093_bioinformatics_btx228/4/btx228.pdf. URL: +%20http://dx.doi.org/10.1093/bioinformatics/btx228.

Han, Bo and Timothy Baldwin (2011). "Lexical Normalisation of Short Text Messages: Makn Sens a #Twitter". In: *Proceedings of the 49th Annual Meeting of*

the Association for Computational Linguistics: Human Language Technologies - Volume 1. HLT '11. Portland, Oregon: Association for Computational Linguistics, pp. 368–378. ISBN: 978-1-932432-87-9. URL: http://dl.acm.org/citation.cfm?id=2002472.2002520.

Han, Bo, Paul Cook, and Timothy Baldwin (2012). "Automatically Constructing a Normalisation Dictionary for Microblogs". In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. EMNLP-CoNLL '12. Jeju Island, Korea: Association for Computational Linguistics, pp. 421–432. URL: http://dl.acm.org/citation.cfm?id=2390948.2391000.

Harris, Zellig S (1954). "Distributional structure". In: *Word* 10.2-3, pp. 146–162.

Hill, Felix, Roi Reichart, and Anna Korhonen (2015). "Simlex-999: Evaluating Semantic Models with Genuine Similarity Estimation". In: *Comput. Linguist.* 41.4, pp. 665–695. ISSN: 0891-2017. DOI: 10.1162/COLI_a_00237. URL: http://dx.doi.org/10.1162/COLI_a_00237.

Hinton, Geoffrey E. et al. (2012). "Improving neural networks by preventing co-adaptation of feature detectors". In: *CoRR* abs/1207.0580. arXiv: 1207.0580. URL: http://arxiv.org/abs/1207.0580.

Hinton, Geoffrey and Ruslan Salakhutdinov (2006). "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786, pp. 504–507.

Hofmann, Thomas (1999). "Probabilistic Latent Semantic Analysis". In: *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*. UAI'99. Stockholm, Sweden: Morgan Kaufmann Publishers Inc., pp. 289–296. ISBN: 1-55860-614-9. URL: http://dl.acm.org/citation.cfm?id=2073796.2073829.

Hu, Baotian et al. (2014). "Convolutional Neural Network Architectures for Matching Natural Language Sentences". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., pp. 2042–2050. URL: http://papers.nips.cc/paper/5550-convolutional-neural-network-architectures-for-matching-natural-language-sentences.pdf.

Hu, Yuheng, Lydia Manikonda, and Subbarao Kambhampati (2014). "What we instagram: A first analysis of instagram photo content and user types". In: *Proceedings of the 8th International Conference on Weblogs and Social Media, ICWSM 2014*. The AAAI Press, pp. 595–598. ISBN: 9781577356578.

Humboldt, W.F. von (1836). *Über die Verschiedenheit des menschlichen Sprachbaues: und ihren Einfluss auf die geistige Entwickelung des Menschengeschlechts*. Druckerei der Königlichen Akademie der Wissenschaften. URL: https://books.google.se/books?id=dV4SAAAAIAAJ.

Instagram (2017). *700 million*. https://instagram-press.com/blog/2017/04/26/700-million/. Accessed: 2018-05-03.

Iyyer, Mohit et al. (2014). "A Neural Network for Factoid Question Answering over Paragraphs". In: *Empirical Methods in Natural Language Processing*. Doha, Qatar.

Jaradat, Shatha (2017). "Deep Cross-Domain Fashion Recommendation". In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. RecSys '17. Como, Italy: ACM, pp. 407–410. ISBN: 978-1-4503-4652-8. DOI: 10.1145/3109859.3109861. URL: http://doi.acm.org/10.1145/3109859.3109861.

Ji, Shihao et al. (2015). "WordRank: Learning Word Embeddings via Robust Ranking". In: *CoRR* abs/1506.02761. arXiv: 1506.02761. URL: http://arxiv.org/abs/1506.02761.

Johnson, Rie and Tong Zhang (2014). "Effective Use of Word Order for Text Categorization with Convolutional Neural Networks". In: *CoRR* abs/1412.1058. arXiv: 1412.1058. URL: http://arxiv.org/abs/1412.1058.

Joulin, Armand et al. (2016). "Bag of Tricks for Efficient Text Classification". In: *CoRR* abs/1607.01759. arXiv: 1607.01759. URL: http://arxiv.org/abs/1607.01759.

Jurafsky, Daniel and James H. Martin (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR. ISBN: 0130950696.

Kim, Yoon (2014). "Convolutional Neural Networks for Sentence Classification". In: *EMNLP*. ACL, pp. 1746–1751.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980. arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

Klein, Dan and Christopher D Manning (2003). "Fast Exact Inference with a Factored Model for Natural Language Parsing". In: *Advances in Neural Information Processing Systems 15*. Ed. by S. Becker, S. Thrun, and K. Obermayer. MIT Press, pp. 3–10. URL: http://papers.nips.cc/paper/2325-fast-exact-inference-with-a-factored-model-for-natural-language-parsing.pdf.

Lai, Siwei, Kang Liu, et al. (2015). "How to Generate a Good Word Embedding?" In: *CoRR* abs/1507.05523.

Lai, Siwei, Liheng Xu, et al. (2015). "Recurrent Convolutional Neural Networks for Text Classification". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI'15. Austin, Texas: AAAI Press, pp. 2267–2273. ISBN: 0-262-51129-0. URL: http://dl.acm.org/citation.cfm?id=2886521.2886636.

Lecun, Y. et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. ISSN: 0018-9219. DOI: 10.1109/5.726791.

Levenshtein, Vladimir Iosifovich (1966). "Binary codes capable of correcting deletions, insertions and reversals." In: *Soviet Physics Doklady* 10.8. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965, pp. 707–710.

Li, Yaoyong, Kalina Bontcheva, and Hamish Cunningham (2005). "SVM Based Learning System for Information Extraction". In: *Proceedings of the First International Conference on Deterministic and Statistical Methods in Machine Learning*. Sheffield, UK: Springer-Verlag, pp. 319–339. ISBN: 3-540-29073-7, 978-

3-540-29073-5. DOI: 10.1007/11559887_19. URL: http://dx.doi.org/10.1007/11559887_19.

Loper, Edward and Steven Bird (2002). "NLTK: The Natural Language Toolkit". In: *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*. ETMTNLP '02. Philadelphia, Pennsylvania: Association for Computational Linguistics, pp. 63–70. DOI: 10.3115/1118108.1118117. URL: https://doi.org/10.3115/1118108.1118117.

Lui, Marco and Timothy Baldwin (2012). "Langid.Py: An Off-the-shelf Language Identification Tool". In: *Proceedings of the ACL 2012 System Demonstrations*. ACL '12. Jeju Island, Korea: Association for Computational Linguistics, pp. 25–30. URL: http://dl.acm.org/citation.cfm?id=2390470.2390475.

Major, Vincent, Alisa Surkis, and Yindalon Aphinyanaphongs (2017). "Utility of general and specific word embeddings for classifying translational stages of research". In: *CoRR* abs/1705.06262. arXiv: 1705.06262. URL: http://arxiv.org/abs/1705.06262.

Mesnil, Grégoire et al. (2013). "Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding." In: *INTERSPEECH*, pp. 3771–3775.

Mhaskar, Hrushikesh, Qianli Liao, and Tomaso A. Poggio (2016). "Learning Real and Boolean Functions: When Is Deep Better Than Shallow". In: *CoRR* abs/1603.00988.

Mikolov, Tomas, Kai Chen, et al. (2013). "Efficient Estimation of Word Representations in Vector Space". In: *CoRR* abs/1301.3781. arXiv: 1301.3781. URL: http://arxiv.org/abs/1301.3781.

Mikolov, Tomas, Quoc V. Le, and Ilya Sutskever (2013). "Exploiting Similarities among Languages for Machine Translation". In: *CoRR* abs/1309.4168. arXiv: 1309.4168. URL: http://arxiv.org/abs/1309.4168.

Mikolov, Tomas, Ilya Sutskever, et al. (2013). "Distributed Representations of Words and Phrases and Their Compositionality". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., pp. 3111–3119. URL: http://dl.acm.org/citation.cfm?id=2999792.2999959.

Minsky, Marvin L. and Seymour A. Papert (1988). *Perceptrons: Expanded Edition*. Cambridge, MA, USA: MIT Press. ISBN: 0-262-63111-3.

Morin, Frederic and Yoshua Bengio (2005). "Hierarchical Probabilistic Neural Network Language Model". In: *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*. Ed. by Robert G. Cowell and Zoubin Ghahramani. Society for Artificial Intelligence and Statistics, pp. 246–252. URL: http://www.iro.umontreal.ca/~lisa/pointeurs/hierarchical-nnlm-aistats05.pdf.

Niazi, Salman et al. (2016). "HopsFS: Scaling Hierarchical File System Metadata Using NewSQL Databases". In: *CoRR* abs/1606.01588.

Ordentlich, Erik et al. (2016). "Network-Efficient Distributed Word2Vec Training System for Large Vocabularies". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. CIKM '16. Indianapolis, Indiana, USA: ACM, pp. 1139–1148. ISBN: 978-1-4503-4073-1. DOI: 10.1145/2983323.2983361. URL: http://doi.acm.org/10.1145/2983323.2983361.

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2014). "Glove: Global Vectors for Word Representation". In: *EMNLP*. ACL, pp. 1532–1543.

Ratner, Alexander J et al. (2016). "Data Programming: Creating Large Training Sets, Quickly". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., pp. 3567–3575. URL: http://papers.nips.cc/paper/6523-data-programming-creating-large-training-sets-quickly.pdf.

Ratner, Alexander et al. (2017). "Snorkel: Rapid Training Data Creation with Weak Supervision". In: *CoRR* abs/1711.10160. arXiv: 1711.10160. URL: http://arxiv.org/abs/1711.10160.

Reis, D. C. et al. (2004). "Automatic Web News Extraction Using Tree Edit Distance". In: *Proceedings of the 13th International Conference on World Wide Web*. WWW '04. New York, NY, USA: ACM, pp. 502–511. ISBN: 1-58113-844-X. DOI: 10.1145/988672.988740. URL: http://doi.acm.org/10.1145/988672.988740.

Riloff, Ellen (1995). "Little Words Can Make a Big Difference for Text Classification". In: *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '95. Seattle, Washington, USA: ACM, pp. 130–136. ISBN: 0-89791-714-6. DOI: 10.1145/215206.215349. URL: http://doi.acm.org/10.1145/215206.215349.

Ritter, Alan, Colin Cherry, and Bill Dolan (2010). "Unsupervised Modeling of Twitter Conversations". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. HLT '10. Los Angeles, California: Association for Computational Linguistics, pp. 172–180. ISBN: 1-932432-65-5. URL: http://dl.acm.org/citation.cfm?id=1857999.1858019.

Ritter, Alan, Sam Clark, et al. (2011). "Named Entity Recognition in Tweets: An Experimental Study". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. EMNLP '11. Edinburgh, United Kingdom: Association for Computational Linguistics, pp. 1524–1534. ISBN: 978-1-937284-11-4. URL: http://dl.acm.org/citation.cfm?id=2145432.2145595.

Ritter, Alan, Mausam, et al. (2012). "Open Domain Event Extraction from Twitter". In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '12. Beijing, China: ACM, pp. 1104–1112. ISBN: 978-1-4503-1462-6. DOI: 10.1145/2339530.2339704. URL: http://doi.acm.org/10.1145/2339530.2339704.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Learning representations by back-propagating errors". In: 323, pp. 533–536. DOI: 10.1038/323533a0.

Rumelhart, David E., James L. McClelland, and CORPORATE PDP Research Group, eds. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 2: Psychological and Biological Models*. Cambridge, MA, USA: MIT Press. ISBN: 0-262-13218-4.

Russakovsky, Olga et al. (2015). "ImageNet Large Scale Visual Recognition Challenge". In: *Int. J. Comput. Vision* 115.3, pp. 211–252. ISSN: 0920-5691. DOI: 10.1007/s11263-015-0816-y. URL: http://dx.doi.org/10.1007/s11263-015-0816-y.

Seki, Yohei (2003). "Sentence Extraction by tf/idf and Position Weighting from Newspaper Articles". In:

Severyn, Aliaksei and Alessandro Moschitti (2015). "UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification." In: *SemEval@NAACL-HLT*. Ed. by Daniel M. Cer et al. The Association for Computer Linguistics, pp. 464–469. ISBN: 978-1-941643-40-2. URL: http://dblp.uni-trier.de/db/conf/semeval/semeval2015.html#SeverynM15.

Shieber, Stuart M. (1987). "Evidence Against the Context-Freeness of Natural Language". In: *The Formal Complexity of Natural Language*. Ed. by Walter J. Savitch et al. Dordrecht: Springer Netherlands, pp. 320–334. ISBN: 978-94-009-3401-6. DOI: 10.1007/978-94-009-3401-6_12. URL: https://doi.org/10.1007/978-94-009-3401-6_12.

Silva, C. and B. Ribeiro (2003). "The importance of stop word removal on recall values in text categorization". In: *Proceedings of the International Joint Conference on Neural Networks, 2003.* Vol. 3, 1661–1666 vol.3. DOI: 10.1109/IJCNN.2003.1223656.

Socher, Richard et al. (2013). "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, WA: Association for Computational Linguistics, pp. 1631–1642.

Sugathadasa, Keet et al. (2017). "Synergistic Union of Word2Vec and Lexicon for Domain Specific Semantic Similarity". In: *CoRR* abs/1706.01967. arXiv: 1706.01967. URL: http://arxiv.org/abs/1706.01967.

Sun, Chen et al. (2017). "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era". In: *CoRR* abs/1707.02968.

Tabassum, Jeniya, Alan Ritter, and Wei Xu (2016). "A Minimally Supervised Method for Recognizing and Normalizing Time Expressions in Twitter". In: *CoRR* abs/1608.02904. arXiv: 1608.02904. URL: http://arxiv.org/abs/1608.02904.

Tixier, Antoine J.-P., Michalis Vazirgiannis, and Matthew R. Hallowell (2016). "Word Embeddings for the Construction Domain". In: *CoRR* abs/1610.09333. arXiv: 1610.09333. URL: http://arxiv.org/abs/1610.09333.

Valiant, Leslie G. (1990). "A Bridging Model for Parallel Computation". In: *Commun. ACM* 33.8, pp. 103–111. ISSN: 0001-0782. DOI: 10.1145/79173.79181. URL: http://doi.acm.org/10.1145/79173.79181.

Vine, Lance De et al. (2015). "Analysis of Word Embeddings and Sequence Features for Clinical Information Extraction". In: *ALTA*. ACL, pp. 21–30.

Vora, Parth, Mansi Khara, and Kavita Kelkar (2017). "Classification of Tweets based on Emotions using Word Embedding and Random Forest Classifiers". In: *International Journal of Computer Applications* 178.3, pp. 1–7. ISSN: 0975-8887. DOI: 10.5120/ijca2017915773. URL: http://www.ijcaonline.org/archives/volume178/number3/28651-2017915773.

Wang, Sida and Christopher D. Manning (2012). "Baselines and Bigrams: Simple, Good Sentiment and Topic Classification". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*. ACL '12. Jeju Island, Korea: Association for Computational Linguistics, pp. 90–94. URL: http://dl.acm.org/citation.cfm?id=2390665.2390688.

Wimalasuriya, Daya C. and Dejing Dou (2010). "Ontology-based Information Extraction: An Introduction and a Survey of Current Approaches". In: *J. Inf. Sci.* 36.3, pp. 306–323. ISSN: 0165-5515. DOI: 10.1177/0165551509360123. URL: http://dx.doi.org/10.1177/0165551509360123.

Wu, Wentao et al. (2012). "Probase: a probabilistic taxonomy for text understanding". In: *Proc. 2012 ACM SIGMOD Int. Conf. Manag. Data*. SIGMOD '12. New York, NY, USA: ACM, pp. 481–492. ISBN: 978-1-4503-1247-9. DOI: 10.1145/2213836.2213891. URL: http://doi.acm.org/10.1145/2213836.2213891.

Yampolskiy, Roman V. (2013). "Turing Test as a Defining Feature of AI-Completeness". In: *Artificial Intelligence, Evolutionary Computing and Metaheuristics: In the Footsteps of Alan Turing*. Ed. by Xin-She Yang. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 3–17. ISBN: 978-3-642-29694-9. DOI: 10.1007/978-3-642-29694-9_1. URL: https://doi.org/10.1007/978-3-642-29694-9_1.

Zaharia, Matei et al. (2010). "Spark: Cluster Computing with Working Sets". In: *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud'10. Boston, MA: USENIX Association, pp. 10–10. URL: http://dl.acm.org/citation.cfm?id=1863103.1863113.

Zhang, Ye et al. (2016). "Neural Information Retrieval: A Literature Review". In: *CoRR* abs/1611.06792. arXiv: 1611.06792. URL: http://arxiv.org/abs/1611.06792.

Zipf, G.K. (1949). *Human behavior and the principle of least effort: an introduction to human ecology*. Addison-Wesley Press. URL: https://books.google.se/books?id=1tx9AAAAIAAJ.

Zsuzsanna Albert, Reka and Albert-Laszlo Barabasi (2001). "Statistical Mechanics Of Complex Networks". In: 74.

TRITA TRITA-EECS-EX-2018:138