Mining User-Generated Texts via Neural Classification

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

Shiwei Zhang

Bachelor of Computer Science and Technology

Hunan University, China

Master of Information Technology

The University of Melbourne, Australia

School of Computing Technologies

College of Science, Technology, Engineering and Maths

RMIT University

October 2020

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work that has been carried out since the official commencement date of the approved research program; and, any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed. I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Shiwei Zhang
School of Computing Technologies
RMIT University
18 October 2020

Acknowledgments

The past four years at RMIT University have been an unforgettable and invaluable experience for me. When I first started my research career in 2016, I had almost zero research experience and only knew a little bit about data mining and natural language processing. Throughout my PhD I have learned a lot about how we properly conduct scientific research and experiments, and now I became more confident to face new research challenges in the future and more motivated to pursue a research career. I am indebted to the help and support of many people in developing and completing this research. First and foremost, my greatest thanks go to all my supervisors, Xiuzhen (Jenny) Zhang, Jey Han Lau, Jeffrey Chan, and Cecile Paris. Jenny, who opened the door of research to me, gave me a lot of guidance and helped throughout my PhD studies. Jey Han's technical expertise in deep learning has been instrumental to the success of the research and also helped me a lot on my writing. I would also like to thank Cecile and Jeffrey for being responsive and providing critical feedback on my research progress.

Throughout my PhD I have been supported by the RMIT University and CSIRO Data61 scholarships and computing resources. I would like to thank Cecile for supporting me in using the computing resources at CSIRO and thank Jenny for building a GPU server for our group.

Lastly, I would like to thank my parents: Duo Zhang and Fengying Zhu, to support me in studying abroad, especially giving me countless financial support. The majority of expenses in my family is the cost of my education and living. I never know how to pay them back, but I hope that they are at least a little proud of me for what I have been through so far.

Contents

List of Figures

List of Tables

Abstract

The growing number of user-generated texts and the availability of powerful hardware provide researchers with an unprecedented opportunity to develop advanced deep learning-based models to analyze opinions, sentiments, attitudes and emotions; this is referred to as sentiment analysis and opinion mining. This thesis aims to develop deep learning-based text classification techniques for mining opinions from user-generated texts such as Tweets and reviews.

In the first part of this thesis, we investigate how we can develop deep learning-based models for Twitter sentiment analysis and opinion mining applications that lack sufficient training resources. In particular, we tackle the following two research challenges: 1) to develop a fully language-independent sentiment classification approach for languages lacking sufficient training resources, and 2) to leverage cross-domain resources in attention-based neural models for irony detection that lacks sufficient training examples. To address the first research challenge, we propose a character-based convolutional neural network for language-independent sentiment classifications. We experiment with Twitter sentiment datasets in six different languages. Experimental results show that our model is fully language-independent and can even work with Tweets written in mixed languages. To address the second research question, we propose a transfer learning-based methodology for irony detection on Twitter. Our proposed method transfers sentiment attentions generated from external sentiment resources to an attention-based model, successfully improving the attention-based model's performance on detecting context incongruity in ironic Tweets.

In the second part of this thesis, we focus on mining opinions from Amazon product reviews, specifically mining product reviews to answer product questions. The first challenge is a lack of ground truth, i.e., there is no direct association between product queries and reviews.

The second challenge is the language mismatch between questions and reviews. To address these two challenges, we propose a mixture-of-experts-based neural framework that leverages existing question and answer pairs as supervision signals, and a pre-trained language model as the relevance function to solve the language mismatch problem. To improve the reliability of our product question answering models, we propose a reliable product question answering framework that uses a conformal predictor as the rejection model to reject unreliable predictions. We experiment with a huge Amazon dataset. Experimental results show that our model can consistently find the most relevant reviews for product questions and generate more reliable predictions when the rejection model is adopted.

Chapter 1

Introduction

User-generated texts—those written by the public—concern a variety of topics, such as politics, movies, food and electronics, and can be collected from a range of on-line resources, from discussion forums (e.g., Reddit [1] and Quora [2]), micro-blogs (e.g., Twitter [3] and Weibo [4]), social network applications (e.g., Facebook [5] and Wechat [6]), photo and video-sharing platforms (e.g., YouTube [7] and Instagram [8]) to e-commerce platforms (e.g., Amazon [9] and Taobao [10]) and review websites (e.g., Yelp [11] and IMDB [12]). User-generated texts in research and industrial applications, especially in natural language processing (NLP) and text mining, have seen rapid growth in the past decade. One reason for this is that collecting user-generated texts is relatively inexpensive. In addition, analyzing opinions in user-generated texts is crucial to understanding human language. Many NLP and text mining tasks, such as sentiment analysis and opinion mining, involve analyzing opinions, sentiments, evaluations, attitudes, and emotions [1]. Many such tasks can be formulated as text classification tasks, such as sentiment

[1] https://www.reddit.com
[2] https://www.quora.com
[3] https://www.twitter.com
[4] https://www.weibo.com
[5] https://www.facebook.com
[6] https://www.wechat.com
[7] https://www.youtube.com
[8] https://www.instagram.com
[9] https://www.amazon.com
[10] https://world.taobao.com
[11] https://www.yelp.com
[12] https://www.imdb.com

classification [2] and emotion detection [3]. Apart from being a source of text classification, user-generated texts can also serve as sources of text summarization [4], text generation [5], lexical normalization [6], machine translation [7], and so on. User-generated texts include a large amount of unstructured information, making text mining extremely difficult because the contents of user-generated texts are easily understandable to humans but remain almost inaccessible to machines. In recent years, however, deep learning-based methods have achieved tremendous success in almost all NLP tasks, and are likely to continue to dominate in this area in the coming years. In this thesis, we mainly focus on mining opinions from user-generated texts using deep learning-based classification techniques.

Table 1.1: Customer reviews and ratings for product "AirPods."

	☆☆☆☆☆	The battery life is pretty good, and popping them back in the case for 10-15 minutes boosts them almost back up to full.
	☆☆	The battery life is extremely good, but for the money, you can buy better. Much better.
	☆☆☆☆	AirPods 配合iphone music, 音质很棒!
	☆	I was really looking forward to getting these and received them a month after ordering. I was amazed to see that these "new" AirPods were covered in dirt when I opened them.
	☆☆☆☆☆	I bought a couple pair for Christmas presents. So far no complaints. My husband uses them with an Android phone! And my daughter with her iPhone! So far GREAT!

Opinion mining is a broad field of study that analyzes opinions on a variety of topics, for example sentiment analysis, extracting opinions on products, people, events, services, and social issues [1]. Table 1.1 provides an example in the form of five customer reviews for the Apple product "AirPods" (a pair of wireless headphones). By analyzing product reviews, a company can better understand whether customers like or dislike their product, and this is one of the applications of sentiment analysis. Sentiment analysis and opinion mining can be conducted at three different levels: document level, sentence level, and aspect level [1]. At the document level, only the overall sentiment polarity of a document is needed. For example, the last review in Table 1.1 contains five sentences and the overall sentiment polarity of this review is very positive, because it has five stars, which is the highest rating in product reviews (the

more stars, the more positive). Sentiment analysis at the document level might be too coarse for some applications, since each sentence of a given document might have a different sentiment polarity. For example, the first, third, and fourth sentences in the last review in Table 1.1 are neutral in terms of sentiment, while the second and the last sentences are positive. In other words, a sentence is likely to contain a single opinion while a document might contain multiple opinions. So, sentiment analysis and opinion mining at the sentence level are necessary; some studies have investigated sentiment analysis and opinion mining at the sentence level [8, 9].

Although sentiment analysis and opinion mining provide more accurate information at the sentence level than at the document level, this level still lacks detail and is insufficient for certain applications because it does not identify opinion targets or sentiments assigned to targets, such as what aspects of products users mention in reviews or whether a user likes or dislikes a particular aspect. Moreover, a sentence's overall sentiment about a specific target does not imply that the author's opinion is positive or negative about a target's aspects [10]. For example, in the review " The battery life is extremely good, but for the money, you can buy better. Much better," the overall sentiment is negative, but the sentiment on battery life is positive. Some studies have investigated sentiment analysis at the aspect level to identify sentiment polarity or even extract opinions about a particular entity [1, 11]. By analyzing sentiment at the aspect level, we are able to obtain more detailed information from reviews, such as (in this example) information on the battery life, price, and sound quality of AirPods. For example, after analyzing the review "The battery life is extremely good, but for the money, you can buy better. Much better," we find that the author is happy with the battery life but is very dissatisfied with the price. Sentiment analysis and opinion mining at different levels have advantages and disadvantages, and which option should be chosen depends solely on the application. In this thesis, we choose to conduct our experiments at the document level (or Tweet level) for Twitter applications, since Tweets are short texts (140 characters), but we use the sentence level for reviews because displaying a relevant sentence for a query is more user-friendly than presenting a lengthy full review.

One significant challenge facing sentiment analysis is the lack of resources, especially annotated data, for languages other than English. User-generated texts are written in a range

of languages. For example, Twitter supports over 40 languages and about 50% of Tweets are non-English [13]. In some cases, user-generated texts may even contain words from multiple languages, such as the third review in Table 1.1, "AirPods 配合iphone music, 音质很棒!". The author of this review is happy with the sound quality of AirPods, especially when they are paired with iPhone Music (an official Apple application). English is an international language, so it is very common for people to blend English words with their mother tongue when writing reviews and Tweets, especially terms such as OK, bye, iPhone, COVID-19, Java, and Nike. These words are likely to be preferred over their counterparts in other languages. To fully analyze the sentiment of user-generated texts, it is therefore essential to develop a language-independent classification model for texts in multiple languages and in mixed languages.

User-generated texts often involve the widespread use of creative and figurative language, like irony and sarcasm. Since the presence of irony or sarcasm often causes polarity reversal (e.g., "I love being ignored"), this could negatively impact sentiment classification [12]. The terms irony and sarcasm are often used interchangeably, despite their subtle differences in meaning [12]. Detecting irony is not only essential for sentiment analysis; it is also important for security services in discriminating between potential threats and comments that are merely ironic [13]. Irony detection is a challenging task [14] that requires inferring the hidden, ironic intent, which cannot be achieved by literal syntactic or semantic analysis of the textual content. Some research focuses on detecting sentiment polarity contrast (or context incongruity), a commonly seen form of irony in user-generated texts [15]. For example, in "I love being ignored," "love" is positive and "ignored" is negative. Most previous studies rely on classical features-based models or rule-based approaches. For example, one previous study [16] follows the rule "a text utterance is perceived to be ironic if a positive sentiment is followed by a negative situation," relying on pre-defined or pre-learned sentiment lexicons or phrases.

The most challenging part of detecting context incongruity is that implicit context incongruity is less likely to be covered by pre-defined or pre-learned sentiment lexicons, e.g., "I like to think of myself as a broken down Justin Bieber - my philosophy professor" and "I was amazed to see that these "new" AirPods were covered in dirt when I opened them." Attention-

[13]https://latimesblogs.latimes.com/technology/2010/02/twitter-tweets-english.html

based recurrent neural networks, as one of most popular models for NLP problems [11, 17, 18] have also been adopted for irony detection [19–21] and have shown better performance than traditional machine learning models such as rule-based approaches and classical feature-based machine learning methods. However, the standard self-attention mechanism often generates attentions for only partial texts forming the context—statement contrast and thus fails to detect context incongruity. One of the reasons that attention-based models fail to detect context incongruity is that almost all previous studies use a handful of human-labeled ironic Tweets for training; however, pattern recognition for detecting irony is so complex and difficult that a considerable size of dataset is needed. As it is costly to build a large annotated dataset for training a high-performance model, transfer learning with sufficient sentiment resources seems to be a reasonable alternative.

User-generated texts often contain rich information or opinions that can be leveraged to fulfill others' information needs. Opinion searching and retrieval has therefore become a useful web application of user-generated texts [1], typically covering two types of opinion search queries, i.e., seeking public opinions about a particular entity or an aspect of an entity and seeking individual opinions on a particular entity or an aspect of an entity. Product reviews have been found to be helpful for providing insights to answer product-related queries [22]. For example, suppose that a user is wondering whether AirPods are compatible with Android phones. The last review in Table 1.1 provides the answer: "My husband uses them with an Android phone! And my daughter with her iPhone! So far GREAT!". The use of reviews as resources for product question answering (PQA) not only relieves the community from answering the growing number of product queries manually, but also shortens the time customers wait for a response and improves their online shopping experience. However, it is difficult to directly train a supervised system to find reviews relevant to product queries, as there is a lack of ground truth, i.e., there is no direct association between product queries and reviews. To tackle this, a better approach is to use existing question and answer pairs as supervision signals. Another key challenge is the language mismatch between questions and reviews. For example, words in bold in the question "Will these gloves keep hands warm while **driving in winter**?" differ from those in the review "No, these aren't warm gloves, but they do still take

the edge off **gripping an ice cold steering wheel**" but their meanings are similar. Thanks to the self-attention mechanism [23], pre-trained neural language models such as BERT [24] and GPT [25] can capture more complex word interactions and dependencies than traditional word embeddings. This makes pre-trained neural language models more effective for soft matching words in queries and documents than traditional word embeddings and other bag-of-words representations [26].

Product reviews are helpful for answering product-related questions. However, not every question can be answered by reviews: the existing set of reviews may not contain any answers relevant to the question, or a question may be poorly phrased and difficult to interpret and therefore require additional clarification. In such a case, a system should abstain from returning any reviews and forward the question to the product seller; this is related to one of the major concerns of machine learning, i.e., reliability or AI safety. Reliability (or AI safety) is an important factor in real-world decision-making systems. It remains a great concern in implementing state-of-the-art deep learning models in real-world applications [27]. High-stakes applications, such as self-driving cars, require not only point predictions but also accurate quantification of prediction uncertainty [28]. A reliable system should measure the confidence of predictions and abstain from decisions where low confidence is low. For example, self-driving cars [29] that are not confident about future predictions in certain situations should not make any moves. Instead, the self-driving system should ask the driver to assume control.

A reliable PQA framework should consider questions' answerability, but this is not largely explored in the PQA literature. Such evaluation focuses on simply ranking performance, without penalizing systems that return irrelevant reviews. That said, the question answerability issue has begun to draw some attention in machine comprehension (MC). Traditionally, MC models assume the correct answer span always exists in the context passage for a given question. As such, these systems will provide an incorrect (and often embarrassing) answer when the question is not answerable. This provides motivation for the development of better comprehension systems that can distinguish between answerable and unanswerable questions. Various MC models have been proposed to detect question-answerability in addition to predicting answer span [30–33]. The MC models are trained to first detect whether a question is answerable

or unanswerable and then find answers to answerable questions. [34] propose a risk controlling framework to increase the reliability of MC models, where risks are quantified based on the extent of incorrect predictions for both answerable and unanswerable questions. Different from that MC always having one answer, PQA is a ranking problem where there can be a number of relevant reviews/answers; PQA is more challenging than a simple binary classification for answerability, and risk models designed for MC cannot be trivially adapted for PQA. An ideal PQA risk controlling framework is expected to ensure that PQA models return more concise and accurate answers for answerable questions and nil answers for unanswerable questions.

Overall, this thesis aims to develop deep learning-based models to tackle three opinion mining problems from user-generated texts, namely sentiment analysis, irony detection and PQA. In terms of methods, we propose developing deep learning-based classification models that involve transfer learning techniques and techniques for improving the reliability of neural models. Text classification has been widely studied over the last few decades. Text classification approaches can be grouped into three categories [35]: rule-based methods [36], machine learning (data-driven) based methods [37, 38], and hybrid methods [39]. Machine learning (data-driven) based methods can be roughly classified into two categories: conventional machine learning approaches [37, 40] and deep learning-based approaches [9, 41]. Conventional classification approaches usually include text preprocessing and feature engineering, while deep learning-based models tend to process the original form of data directly and automatically learn features. In the past decade, hundreds of deep learning-based models have been proposed for various text classification problems and have become highly developed. One major challenge in machine learning research problems is that most problems require a large quantity of labeled data, but it is not feasible to build a large amount of data for training since the cost is prohibitively high. In NLP, there are widely available unlabeled text corpora containing rich semantic and syntactic information for learning language models. Moreover, the existing labeled text resources and text-related machine learning models can provide additional help for text mining tasks that lack labeled data. Therefore, there is a need to leverage the existing data or knowledge learned by previous models for learning high-performance models. This methodology is referred to as transfer learning. In recent years, pre-trained language models such as BERT [24] and

XLNet [42] have been shown to be useful in learning common language representations by utilizing a large amount of unlabeled data. By leveraging knowledge learned by these pre-trained language models, many deep learning-based models have achieved impressive results in various NLP tasks, including text classification [41], text summarization [43], and text comprehension [24]. In this thesis, transfer learning techniques are adopted for tackling scarcity issues in training data, (e.g., limited annotated resources or training examples), all of the tasks in this thesis currently face. Techniques for improving the reliability of a text classifier are also investigated for the sake of a reliable PQA framework.

1.1 Contributions

The contributions of the thesis are as follows:

- We propose two convolutional neural networks, UniCNN and a word-character CNN, to tackle language-independent Twitter sentiment classification problems. We experiment with Twitter sentiment datasets containing Tweets in six different European languages and achieve promising results. We also evaluate UniCNN with another Twitter dataset that has Tweets in more than 40 languages, including many in mixed languages. Experimental results demonstrate that UniCNN is not language-independent but can work with Tweets written in mixed languages, and word-character CNN, with two different transformations of short text as the input, gives much better performance than using only words or characters.

- We propose three attention-based neural models for irony detection on Twitter. The proposed methods incorporate sentiment attentions generated from external sentiment resources to improve attention-based models. By incorporating transferred sentiment, our models can detect both implicit and explicit context incongruity in most instances. Experiments show that our three proposed sentiment attention mechanisms result in better performance than baseline models, including several popular neural models for irony detection on Twitter.

- We propose a neural framework to discover relevant reviews for answering product questions. Our proposed neural framework successfully learned a neural relevance function for ranking reviews by leveraging existing questions and answers as the training signal. We experiment with seven product categories and demonstrate that our best model produces strong performance, outperforming state-of-the-art systems by consistently finding the most relevant reviews for product queries. With the use of BERT as the relevance function, the language mismatch problem between questions and reviews has also been successfully addressed. We conduct several user studies and find that the reviews selected by our best model are not only relevant but also use different words/phrases to those in questions when their meanings are similar.

- Extending our previous work, we propose a reliable PQA framework to tackle answer reliability. Our framework introduces a novel application of conformal predictors as a rejection model to reject unreliable predictions. With our proposed framework, the returned results are more concise and accurate in answering questions, including returning nil answers for unanswerable questions. In terms of producing reliable predictions, experiments on a widely used Amazon dataset show that PQA models integrating the rejection model consistently outperform those without the rejection model.

1.2 Thesis Outline

Chapter 2

In this chapter, we provide an overview of text classification. First, we briefly summarize text classification techniques, mainly focusing on deep learning-based text classification models. We then discuss the importance of transfer learning in text classification, mainly focusing on answering the following three questions: *when to transfer?*, *what to transfer?* and *how to transfer?*. Next, we discuss another important issue in text classification, the reliability of classification models. Lastly, we give some examples of applications where text classification techniques are mainly adopted to solve user-generated text problems.

Chapter 3

In this chapter, we propose two CNN-based classifiers for language-independent Twitter sentiment classification. We first discuss the importance of language-independent Twitter sentiment classification and then discuss the drawbacks of existing methodologies for language-independent Twitter sentiment classification. Next, we describe character-based neural models, which are shown to be useful for text classification and have the potential to be applied in language-independent Twitter sentiment classification. We introduce our proposed model, which leverages the benefits of UTF-8 character encoding and character-based CNN. We provide a detailed explanation of how our model does not require language identification and is, therefore, a fully language-independent approach. Further, we investigate whether leveraging convolutional features at both word-level and character-level helps in Twitter sentiment analysis. We run tests over Twitter datasets that contain Tweets in multiple languages and even in mixed languages.

Chapter 4

In this chapter, we study the problem of irony detection on Twitter and investigate a transfer learning-based approach. First, we discuss common patterns to infer hidden, ironic intent. We then discuss the importance for sentiment analysis of accurately detecting irony or sarcasm in social media. Next, we review previous methodologies for detecting irony, especially deep learning-based methods. We focus on why traditional attention-based models fail to fully detect context incongruity and propose our methods. In our methods, we formulate irony detection as a transfer learning task, where supervised learning on irony-labeled text is enriched with knowledge transferred from external sentiment analysis resources. Importantly, we focus on identifying hidden, implicit incongruity without relying on explicit incongruity expressions, as in "I like to think of myself as a broken down Justin Bieber - my philosophy professor." We propose three transfer learning-based approaches to use sentiment knowledge to improve the attention mechanism of recurrent neural models for capturing hidden patterns for incongruity. We experiment extensively to determine whether incorporating transfer sentiment attention improves models with regard to detecting sentiment contrast.

Chapter 5

In this chapter, we propose the retrieval of relevant product reviews for answering product-related questions. We first discuss the possibility of using product reviews to answer product queries on e-commerce platforms. Next, we discuss the key challenges facing PQA using reviews, particularly the lack of ground truth and the language mismatch between user queries and reviews. To address this, we propose two neural models that discover relevant reviews for answering product queries, using existing question and answer pairs as supervision signals, and pre-trained word embeddings or language models as the relevance function. We experiment with a large Amazon dataset that has questions collected from several different categories. We also conduct user studies for quality evaluation.

Chapter 6

This chapter proposes a reliable PQA framework, focusing on improving the reliability of text classification approaches using only a small amount of annotated data. We first discuss the unreliable predictions generated by PQA models. Next, we investigate the intuition that many questions may not be answerable with a finite set of reviews. We suggest that when a question is not answerable, a system should return nil answers rather than provide a list of irrelevant reviews, which can significantly negatively impact the user experience. Moreover, we investigate the intuition that, for answerable questions, those reviews most relevant to answering the question should be included in the results. We present our PQA framework to improve the reliability of PQA systems. It rejects unreliable answers so that the returned results are more concise and accurate at answering questions, and produces nil answers for unanswerable questions. We experiment with a widely used Amazon dataset and compare the performance of three PQA models with and without adopting our reliable PQA framework.

Chapter 7

In the final chapter, we summarize and discuss the impacts of this thesis. Lastly, we propose future work.

Chapter 2

Background

This chapter first provides background knowledge about text classification techniques. Next, it discusses prior transfer learning approaches in NLP. Finally, it discusses the reliability of modern neural networks.

2.1 Conventional Text Classification

2.1.1 Overview

Text categorization is the problem of classifying text into categories or classes. It has been widely studied in recent decades. Many machine learning approaches have been applied for tackling various text classification applications. Conventional classification approaches usually

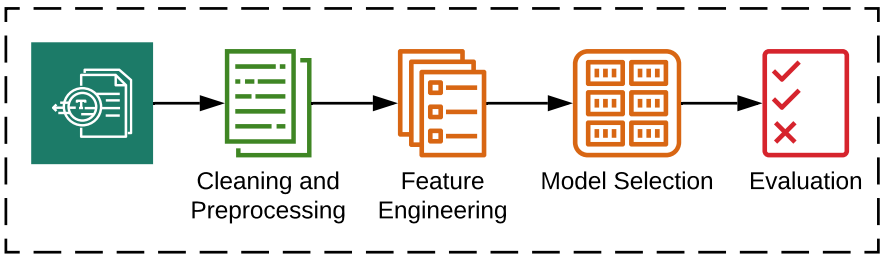


Figure 2.1: The pipeline of conventional text categorization approaches.

include the following four steps: text cleaning and preprocessing, feature engineering, model selection, and evaluation (as illustrated in Figure 2.1).

2.1.2 Cleaning and Preprocessing

In most texts, many words, such as stop words and misspellings, are not specific to an individual class or discriminatory to the different classes. When using conventional classification algorithms, especially statistical and probabilistic learning algorithms, those words could adversely affect the performance of algorithms. So, texts usually need to be cleaned and preprocessed before feature engineering. The most common techniques and methods for text cleaning include tokenization, stop words removal, stemming, capitalization, noise removal, spelling correction and lemmatization.

Text tokenization is the first step of text cleaning and preprocessing, where texts are split into tokens; there may be be words, phrases, or other meaningful tokens. For example, "They will fly on a Crew Dragon spacecraft." will be split into a list of tokens: {"They", "will", "fly", "on", "a", "Crew", "Dragon","spacecraft", "."}. Popular text tokenization tools include NLTK Tokenizer [44], spaCy [1], and Stanford Tokenizer [45]. Most texts share common words, such as "a", "the", and "after" that do not have important meanings in classification algorithms. These are called stop words often need to be removed (stop words removal). However, stop words can also be used as golden features in some tasks, for example plagiarism detection [46], where a small list of frequent stop words is used to capture syntactic similarities between suspicious and original documents. In addition, texts often contain a diversity of capitalization, forming sentences. The most common approach is to convert all letters to lower case, which helps reduce the vocabulary size but may also cause significant problems; for example "Apple" (a brand name) will be treated as "apple" (a fruit). In stemming, different forms (e.g., singular/plural noun form and different tenses) of the same word are consolidated into a single word. For example, the stem of the word "learning" is "learn" and the stem of the word "approaches" is "approach". There are three types of stemming algorithms [47]: truncating methods such as Porter [48], statistical methods such as hidden Markov model (HMM) stem-

[1] https://spacy.io

mer [49], and inflectional and derivational methods such as Krovetz stemmer (KSTEM) [50]. In lemmatization [51], the suffix of a word is replaced or completely removed in order to get back to the common base form (lemma); for example, the common base form of "am," "are," and "is" is "be". The last step of text pre-processing is text normalization [6, 52], which includes noise removal (removing unnecessary tokens or characters, such as characters, punctuation, incorrectly spelled words, and rare words), and spelling correction (replacing an incorrect word with the correct one).

2.1.3 Feature Engineering

Before choosing a classification algorithm, a fundamental step that needs to be accomplished is feature engineering including text representation and feature selection. A number of text representation techniques have been studied and used for conventional classification approaches in the past few decades, including "n-grams", "bag-of-words (BoW)", "term frequency-inverse document frequency (TF-IDF)" and word embeddings.

N-grams is a technique that slices a given string into a set of overlapping n-grams [53]. For example, the bi-grams of the sentence "They will fly on a Crew Dragon spacecraft" are {"They will", "will fly","fly on","on a", "a Crew", "Crew Dragon", "Dragon spacecraft"}.

Bag-of-words (BoW) represents a given text or document with a bag of its words, normally using word frequency as the feature. For example, a given document has two sentences: "James is a computer scientist. His research is focusing on computer vision." When we need to separately represent the two sentences in this document, we first construct the vocabulary BoW and then encode each word in the vocabulary (using word frequency) into a one-hot vector:

- Bag-of-words: {"James", "is", "a", "computer", "scientist", "His", "research", "focusing", "on", "vision"}.
- Vector for first sentence: {1111100000}
- Vector for second sentence: {0101011111}

Term frequency-inverse document frequency (TF-IDF) combines term frequency (TF) and inverse document frequency (IDF) [54].

$$W(d,t) = TF(d,t) * \log(\frac{N}{df(t)})$$

IDF is obtained by $\log(\frac{N}{df(t)})$. N is the number of documents, and $df(t)$ is the frequency of item t in documents. The TF-IDF value increases proportionally to the frequency of a word in a document but is penalized by its frequency in all documents, which helps to downgrade the importance of words that have high frequency over many documents.

2.1.4 Classification Algorithms

A wide variety of algorithms have been designed for text classification, in addition to neural models. Texts and documents tend to have sparse and high-dimensional word features after feature engineering. In terms of choosing a proper model for a type of task or text, it is critical to select classification algorithms that effectively account for the characteristics of the text. In this section, we will discuss some of the most popular classification algorithms.

Naïve Bayes classifier [40] is a classification technique developed based on Bayes theorem [55]. The Naïve Bayes text classifier makes assumptions that text data are generated by a parametric model and uses training data to estimate the parameters of the model. The class of a new example is predicted by using Bayes' rules. The classifier turns the generative model around and calculates the posterior probability that a class is more likely to generate the new example. The Bayes optimal decision rule for classification is

$$\hat{H}(d) = \underset{c \in \mathcal{C}}{\operatorname{argmax}}\, p(c|D,d) = \underset{c \in \mathcal{C}}{\operatorname{argmax}}\, p(|\hat{\theta}^c)p(c)$$

where D is the training data, d is a test instance, $c \in C$ is a class, $\hat{\theta}^c$ is the estimated parameters for class c. This classification rule simply chooses the class c for which the test document d is most likely.

Logistic regression [56] is a linear classifier that models the probability of classes. It combines features of a given instance with *sigmoid* function:

$$p(c|x_i) = \frac{\exp(w * x_i + b)}{1 + \exp(w * x_i + b)}$$

where w is the learned parameters, x_i is the feature vector for the given instance and b is the bias.

Boosting [57] aims to seek a highly accurate classifier by leveraging many weak or base hypotheses, each of which may be only moderately accurate.

K-nearest neighbor (KNN) is a non-parametric technique that is commonly used in the field of pattern recognition [58], text categorization [59, 60], and object recognition [61]. KNN is easy to implement and works well in data sets with multi-modality. The drawbacks of KNN are the extensive memory requirements and computational complexity, since it uses all the features while computing the similarity between a test example and training examples. KNN predicts a new example based on the K-nearest training samples to the new example, and the predicted label, will be the one with the largest category probability.

Conditional random field (CRF) [62] is an undirected graphical model. It is effective for prediction tasks where the neighbors' contextual information or state affect the current prediction. CRFs are mostly used for sequence labeling problems such as named entity recognition [63], part-of-speech tagging [64], and gene prediction [65]. It defines a conditional probability distribution $p(y|x)$ for output y and input x:

$$p(y|x) = \frac{1}{Z_\lambda(x)} \exp\{\lambda \cdot \sum_i f(y, x, i)\}$$

where $f(y, x, i)$ is a feature vector for input sequence x and output sequence y at position i. λ is a weighting vector, and $Z_\lambda(x)$ is a normalization factor.

Support vector machine (SVM) [37, 66] has been one of the most prominent machine learning models for high-dimensional sparse data commonly encountered in applications such as text classification. The original SVM was developed by Vapnik [67]. SVMs are linear functions of the form $f(x) = wx + b$, where $w \cdot x$ is the inner product between the weight vector w and the input vector x. The basic idea of SVM is to choose a hyper-plane that maximizes the margin between the positive and negative examples.

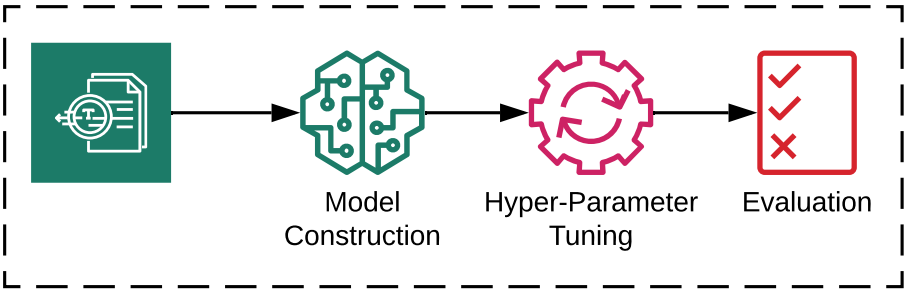


Figure 2.2: The pipeline of deep learning-based approaches.

2.2 Deep Learning-Based Classification

2.2.1 Overview

There are hundreds of deep learning-based models proposed for various text classification problems. These models can be grouped into the following categories, based on their main architectures: feed-forward neural networks [68], convolutional neural networks (CNNs) [9], recurrent neural networks (RNNs) [69], attention-based neural networks [70], graph neural networks (GNNs) [71], Transformer-based neural networks [41] and hybrid models [39]. The pipeline of using deep learning-based models for text classification is shown in Figure 2.2. It generally includes the following three steps: constructing neural networks, tuning parameters, and evaluation. In comparison with conventional approaches, most deep learning-based models tend to process texts in their original form (raw), which means most text preprocessing techniques are not used anymore. Moreover, deep learning-based models barely conduct feature engineering; instead, only distributed representations are adopted, and features are learned automatically. In addition, deep learning-based models are much more flexible, and most studies propose their own deep learning models with unique architectures or computation graphs. However, a good model usually requires a large volume of data and involves great effort for tuning hyper-parameters [72, 73], which is computationally expensive. In addition to advanced neural architectures, distributed text representations also play a significant role in tackling NLP problems. In the following sections, details of both distributed text representations and deep learning models for text classification are briefly discussed.

2.2.2 Distributed Representations

Distributed representations play a significant role in improving the performance of text classification models. There are six common word representations from two different families—word embeddings: word2vec [74], GloVe [75] and fastText [76] and second, contextualized embeddings or language models: ELMo [77], GPT [25] and BERT [24]. Although most use embeddings at the word level, sometimes the distributed representations of text can also be learned at character level [78], sub-word level [79, 80] (which is used by most Transformer models), sentence level [81], and paragraph level [82]. Importantly, even though distributed representations are treated as features or input layers to the deep learning model, sometimes they can be directly used for text classification without adding more trainable layers on top, except for a classification layer. For example, in fastText [83], an average pooling layer follows the embedding layer, which then generates a vector as the final feature for the prediction layer (a fully connected layer plus softmax). In addition, some Transformer-based models that produce contextualized representations (e.g., BERT and GPT) can also be used as text classifiers.

2.2.2.1 Word Embeddings

Word embedding is an NLP technique that maps each word of vocabulary to a fixed n-dimensional vector of real numbers. The vector of words supposedly captures the semantic meaning of the words at some level. In recent years, a variety of word embedding methods have been proposed to learn meaningful word vectors. Most of those learning methods are based on the distributional hypothesis, according to which semantically similar words tend to appear in the same contexts [84].

Word2vec [74] can be trained with one of two objectives, namely continuous bag-of-words (CBOW) and skip-gram. The CBOW model tries to predict the target word based on previous words, while the skip-gram model tries to predict the context words given the target word. When using the word vector space of word2vec to evaluate the distances of words, there are various dimensions of differences. For example, the analogy "king is to queen as man is to woman" should be encoded in the word embeddings space by the vector equation: "king" − "queen" = "man" − "woman".

GloVe [75]: is a global log-bilinear regression model for the unsupervised learning of word vectors that have been learned by the objective of estimating the log-probability of a word pair co-occurrence. GloVe outperforms other word representation models on word analogy, word similarity, and named entity recognition tasks [75].

FastText [76]: is based on the skip-gram model, but each word is represented as a bag of character n-grams. This is especially helpful in morphologically rich languages and for handling rare and misspelled words in English.

2.2.2.2 Contextual Representations

Contextual representations are language modeling functions that compute dynamic word embeddings for words, given their context. In traditional word embeddings, such as word2vec and GloVe, the word vector of a word is fixed even though many words have different meanings in different contexts. For example, the word "apple" in the sentence " I like to have an apple after a meal" and the sentence "I like Apple's products" has two totally different meanings. The contextual embeddings, which generate word embeddings based on a word's context on the fly, are different from traditional word embeddings. Contextual embeddings are pre-trained as general purpose language models using large-scale unannotated data. They can later be fine-tuned in downstream tasks such as sentiment analysis, question answering, machine translation, and summarization.

ELMo (embeddings from language models) [77] vectors are derived from a bidirectional long short-term memory (bi-LSTM) model trained with a paired language model (LM) objective on a large text corpus. ELMo trains an LM bidirectionally, with a left context (where the LM starts from the left goes to the end of the sentence) and a right context (where the LM starts from the right and moves to the beginning of the sentence).

GPT (generative pre-training) [25] has a similar training objective, but uses the Transformer decoder [23] as the LM instead of bi-LSTM. The learning process of GPT has two stages: pre-training and discriminative fine-tuning. By unsupervised pre-training on a large number of unlabeled text corpora with long stretches of contiguous text and discriminative fine-tuning on downstream tasks, GPT is able to transfer knowledge learned from unlabeled

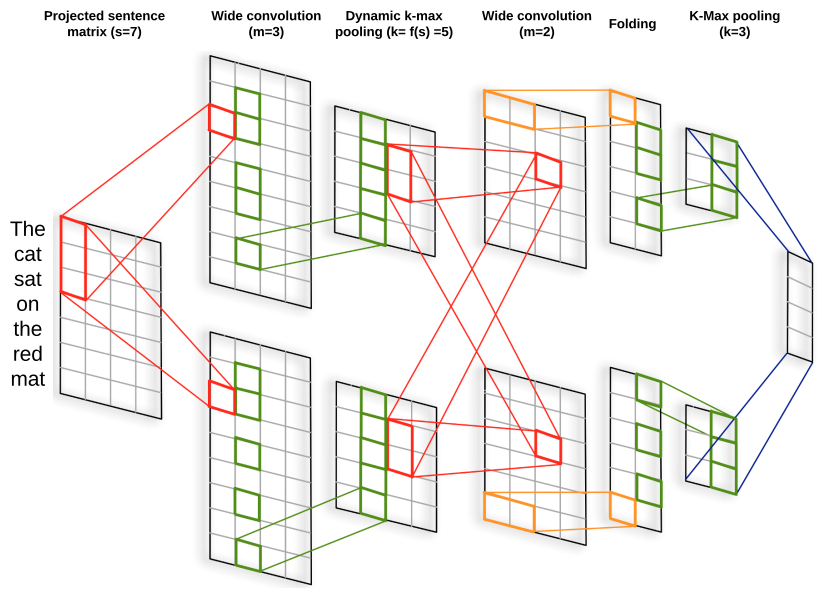


Figure 2.3: The architecture of the DCNN model (Source: [85]).

text to downstream tasks such as textual entailment, question answering, and text classification. GPT-2 [83] is a modified version of GPT, which achieves promising results on several NLP tasks even without fine-tuning on downstream tasks (zero-shot transfer learning). It highlights language models' ability to perform well across many tasks if large language models are pre-trained on sufficiently large and diverse datasets.

BERT (bidirectional encoder representations from transformers) [24] is also based on the Transformer [23], but it uses the Transformer as an encoder. In addition, BERT is bidirectional instead of left-to-right (as in GPT). BERT also introduces two novel pre-training objectives: masked words prediction and next sentence prediction. In masked words prediction, it randomly masks some words/tokens, and the objective is to restore them from the context. In next sentence prediction, given a sentence pair (A, B), the model predicts whether sentence B follows sentence A.

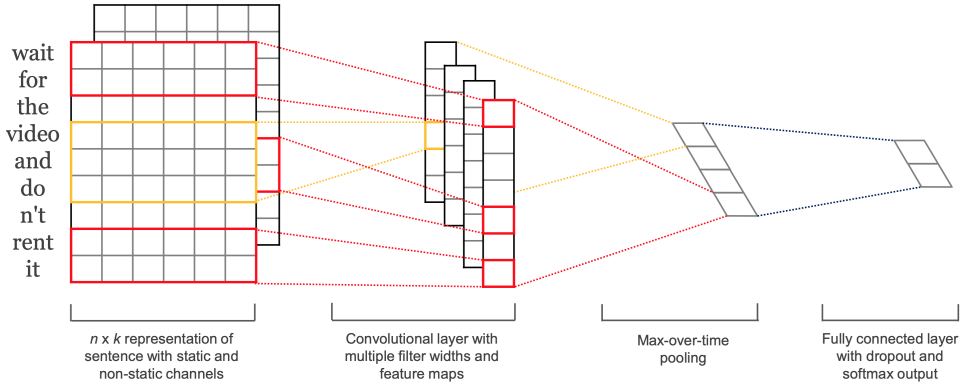


Figure 2.4: The architecture of the CNN_Kim model (Source: [9]).

2.2.3 Deep Learning-Based Text Classification

2.2.3.1 CNN-Based Text Classification

The convolutional neural network (CNN) [86] was originally invented for computer vision and achieved amazing performance on image classification [87]. It has subsequently been shown to be effective for NLP tasks including text classification [9, 85, 88]. One of the early CNN-based text classifiers is the dynamic convolutional neural network (DCNN) proposed by Kalchbrenner et al. [85]. The architecture of DCNN is shown in Figure 2.3. It consists of convolutional layers, a dynamic k-max pooling layer and a fully connected layer. The convolutional filters of DCNN are one-dimensional, convolving across each row of features in the sentence matrix. The convolutional filters with n-grams at every position in the sentence allow features to be extracted independently of their position in the sentence.

Another popular CNN-based text classification model is CNN_Kim, proposed by Kim [9], which has since been adopted by many works. CNN_Kim, shown in Figure 2.4, is simpler than DCNN. It has only one convolutional layer and one max pooling layer. The convolutional layer in CNN_Kim has filters with different windows sizes {2, 3, 4}. The dimensions of the filter in CNN_Kim are equal to the dimensions of the word vectors, which is different from DCNN. Another novel aspect of CNN_Kim is that it uses pre-trained word vectors, word2vec, and then fine-tunes them on classification tasks.

The above two CNN-based classifiers process text at the word-level. The main drawback of

word-level CNN is the fixed vocabulary or out-of-vocabulary words (OOV). Words not in the vocabulary and misspellings are replaced by a unique token: "UNK". Working on characters has the advantage that abnormal character combinations such as misspellings and other OOV words or characters can be learned. A character-level CNN is therefore proposed by Zhang et al. [88] for text classification. There are also studies investigating the impact of CNN architectures on model performance, such as the number of deep layers. Conneau et al. [72] present a very deep CNN (VDCNN) model for text classification. This consists of up to 29 convolutional layers and operates at the character level, using only small convolutions and pooling layers. Another deep CNN is the deep pyramid CNN (DPCNN), proposed b Johnson et al. [89], which is a word-level CNN with 15 weight layers. To enable the training of deep CNNs, both VDCNN and DPCNN adopt the residual connections developed in ResNet [90].

2.2.3.2 RNN-Based Text Classification

A recurrent neural network (RNN) is an extension of a conventional feed-forward neural network. An RNN is able to handle a variable-length input sequence, in contrast to CNN. RNN-based models view the text as a sequence of words or tokens. They are able to capture word dependencies and text structures for text classification. However, the basic RNN is likely to fail to capture long-term dependencies because the gradients tend to either vanish or explode. Long short-term memory (LSTM) [91] and gated recurrent unit (GRU) [92] are two of the most popular RNN variants, and have a more sophisticated activation function to solve the gradient vanishing problem. LSTM introduces a memory cell to remember values over arbitrary time steps, and three gates (input gate, output gate, and forget gate) to regulate the flow of information into and out of the cell. GRU also introduces gating units to modulate the flow of information inside the unit, but without having a separate memory cell. Tang et al. [93] develop two target dependent LSTMs for target-dependent sentiment classification, where target information is automatically taken into account. Tai et al. [69] present a tree-LSTM for sentiment analysis and to evaluate semantic relatedness. This is a generalization of LSTM to tree-structured network typologies. Attention mechanisms [94], originally invented for machine translation, have been adopted in many RNN-based models for many text classification

tasks, such as attention-based bi-LSTM for relation classification [70], attention-based LSTM for target-dependent sentiment classification [95], and hierarchical attention networks for document classification [96]. The attention mechanism used in classification models is normally called "soft attention". It is usually placed on top of an LSTM/GRU model. Suppose that H is a matrix consisting of output vectors produced by an LSTM layer and t is the sentence length. M is the outputted matrix of an LSTM/GRU and w^t is the transpose of weights of a fully-connected layer. The output of the attention layer is the weighted sum of these output vectors:

$$M = tanh(H)$$

$$\alpha = softmax(w^t M)$$

$$r = H\alpha^t$$

2.2.3.3 Transformer-based Text Classification

CNNs are likely to fail to capture relationships between words in long sentences, unlike RNNs. However, RNNs suffer from excessive computational cost since they have to process the text word by word. Transformers [23] overcome these limitations by applying self-attention mechanisms to compute the attention score between every word in a sentence. The computed attention scores reflect the influence each word has on another. Transformer-based models normally first pre-train models on large-scale unlabeled text corpora via an unsupervised training, then fine-tune the models on downstream tasks including text classification. The most popular transformer-based models include XLNet [42], BERT [24], and GPT [25]. Both XLNet and GPT are autoregressive language models, while BERT is an autoencoding language model. Transformer-based text classification models have recently achieved impressive results on many NLP tasks. For example, XLNet is the leading model on several text classification tasks, such as the AG News corpus [88], the DBpedia ontology dataset [88], and the Amazon reviews sentiment analysis dataset [88]. Also, BERT and variants of BERT have achieved excellent performance on several text classification tasks, for example, ALBERT [97] on benchmark GLUE [98] and RoBERTa [99] on benchmark SuperGLUE [100].

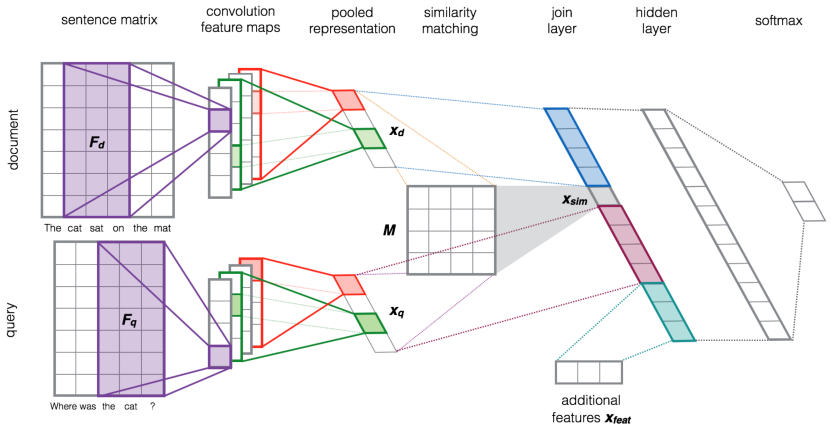


Figure 2.5: The architecture of a CNN-based S2Nets (Source: [101])

2.2.3.4 Siamese Neural Networks (S2Nets)

S2Nets are neural networks mainly designed for text matching-related tasks, such as answer selection-based question answering [102] and document ranking for search queries [101]. These tasks can be viewed as special cases of text classification. For example, in answer selection-based question answering tasks, we want to classify a candidate answer as an answer or a non-answer to a given question. In this case, the produced classification possibility of a classifier shows how possible a candidate is as the answer. All candidate selections are scored and ranked, and the top ones are considered as the answer to a given question. The process of learning S2Nets is also considered learning similarity functions, scoring the similarity between pairs of objects. For example, Severyn et al. [101] present a CNN-based S2Net (shown in Figure 2.5) for re-ranking pairs of short texts. Their model learns not only the distributed representation of text pairs, but also a similarity function to relate them in a supervised way from the available training data. In Figure 2.5, this CNN-based S2Net has two convolutional channels, which can take query and document embeddings as the input respectively and the generated feature vectors from both channels are concatenated together with other features before the prediction layer. There are recent works that use language models to construct a S2Nets, such as Sentence-BERT (Siamese BERT-networks) [103] and bi-encoders [104]. Transformer-based language models have the ability to capture relationships between words in long sentences, so an alternative way to process pairs of objects is to concatenate two sequences as one, with a

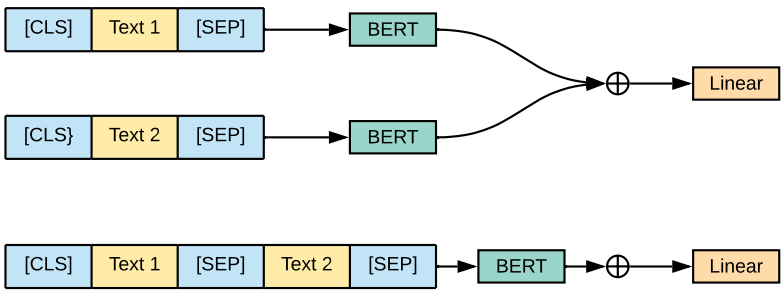


Figure 2.6: The top model (bi-encoder) is an example of using BERT in an S2Net and the bottom model (cross-encoder) is an example of treating two input texts as a single input for a BERT.

special token inserted between the two sequences (as shown in Figure 2.6). This is also known as the cross-encoder [104]. By jointly encoding the context and the candidate in a single Transformer, a cross-encoder has the advantage of learning richer interactions between input sequences by allowing every word in one sequence to attend to every word in another sequence. It is, however, more computationally expensive than a bi-encoder [104], where interactions between input sequences are learned at a higher and more abstract level, such as sentence level (using abstract representations of sentences).

2.2.3.5 Other Text Classifiers

So far, we have discussed popular text classifiers, such as CNN-based models, RNN-based models, S2Nets and transformer-based models. Transformer-based models are the state-of-the-art models for most text classification tasks. Apart from the above mentioned models, there are other types of deep learning-based text classifiers, such as graph neural networks (GNNs) and hybrid models. GNNs for text classifications model the internal relationships of text, e.g., converting text to graph-of-words and using graph convolutional operations to convolve the graph-of-words [71]. Hybrid classification models are models composed in different neural layers. For example, a CNN-LSTM presented by Wang et al. [39] for sentiment analysis has LSTM layers following convolutional layers and max pooling layers. CapsNets, presented by Sabour et al. [105] address the problems of pooling layers in CNNs, which are likely to lose some information during pooling operations. Capsules of CapsNets are able to maintain all

the information of inputs from lower layers to upper layers. This gives more information to the final prediction layer in comparison with the pooling layers in CNN. Zhao et al. [106] explore CapsNets with dynamic routing for text classification.

2.3 Text Classification via Transfer Learning

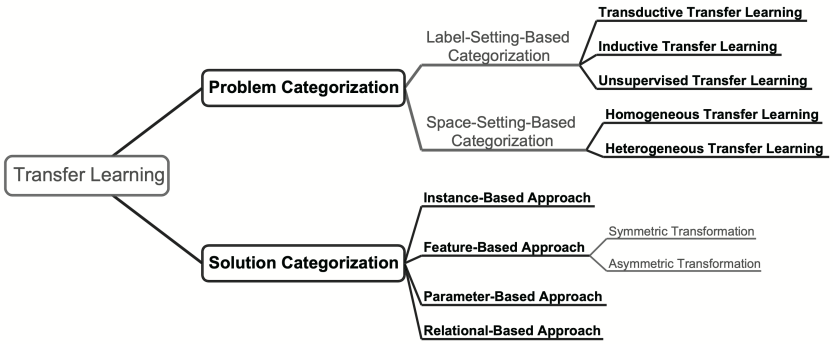


Figure 2.7: Categorizations of transfer learning [107].

2.3.1 Overview

Supervised learning is the technique adopted by the majority of deep learning-based text classification models. With supervised learning we use labeled data, i.e., a data set that has already been annotated or classified, to infer a learning algorithm. Supervised classification methods have achieved significant success in research and have been applied in many text classification problems. In recent years particularly, benefiting from the increasing volume of labeled data and the fast development of deep learning techniques, they have made much progress. However, the required large quantity of labeled data is not always available in many machine learning research problems and applications, and it is not feasible to build a large amount of data for training since the cost is prohibitively expensive. In NLP, there are widely available unlabeled text corpora containing rich semantic and syntactic information for learning language models. Moreover, the existing labeled text resources and text-related machine learning models can provide additional help for text mining tasks that lack labeled data. Therefore, there is a need to leverage the existing data or knowledge learned by previous

models for learning high-performance models. This methodology is referred to as transfer learning.

There are several categorization criteria for transfer learning, shown in Figure 2.7. For example, transfer learning can be classified into **homogeneous transfer learning** and **heterogeneous transfer learning**, based on whether the feature spaces and label spaces in the source and target domains are the same [108]. **Domain adaptation** is a case of heterogeneous transfer learning, where the input feature spaces differ between source and target domains [108]. Other categorization criteria [109] of transfer learning include **inductive transfer learning** (where labeled instances are available in the target domain), **transductive transfer learning** (where labeled instances are available in the source domain but not in the target domain), and **unsupervised transfer learning** (where there is no labeled data in either source or target domain). Pan et al. [109] summarize transfer learning approaches in four classes: instance-based transfer learning, feature-based transfer learning, parameter-based transfer learning, and relation-based transfer learning. **Instance-based transfer learning** approaches try to reuse some instances of the data in the source domain by applying a re-weighting strategy for learning in the target domain. **Feature-based transfer learning** approaches transfer the feature representation of the knowledge learned in source domains to the target domain so as to improve the performance of the target task. **Parameter-based transfer learning** approaches transfer the knowledge at model/parameter level. **Relational-based transfer learning** approaches assume that some relationships among the data in the source and target domains are similar. Those relationships can therefore be transferred from the source domain to the target domain, for example logical relationships or rules learned in the source domain. In terms of the number of labeled instances in the target domain, there are three extreme forms of transfer learning: **zero-shot learning**, **one-shot learning**, and **few-shot learning**. In one-shot learning (or few-shot learning) , only one (or a few) labeled examples of the target task are given, while no labeled examples at all are given for the zero-shot learning task.

All of these categorization criteria of transfer learning are actually addressing the following three main research issues [109]: 1. *When to transfer?*, 2. *What to transfer?*, 3. *How to transfer?*. In the following sub-sections, I will mainly discuss how the combination of deep

learning and transfer learning in recent years solves text classification problems regarding the above three main research issues.

2.3.2 When to Transfer?

When to transfer? asks in which situations do we need to transfer knowledge from source domains to target domains? In conventional text classification approaches, it is necessary to take time investigate whether knowledge from source domains should not be transferred to the target domain, because it might have negative impacts on the performance of learning in the target domain, also referred as negative transfer [109]. For example, suppose we have annotated sentiment data on reviews of electronics and we lack labeled data on reviews of automobiles. We want to transfer sentiment knowledge learned from electronics to automobiles. The risk here is that sentiment words in the source domain could have the opposite sentiment meaning in the target domain. For example, the word "small" can have a positive meaning when describing an electronic device but a negative meaning when describing a car or its internal space. Such knowledge transfer can have a negative impact on target domains. However, in recent years, transfer learning has became an indispensable part of deep learning-based text classification approaches. In 2014, Kim [9] proposed a CNN-based sentence classification approach, comparing CNNs with randomly initialized word embeddings and with pre-trained word embeddings (word2vec), and finding that CNNs with pre-trained word embeddings consistently outperform those with randomly initialized word embeddings. Pre-trained word embeddings are learned by neural language models using a large unlabeled data corpus, and are believed to capture linguistic regularities in language. Using pre-trained word embeddings in NLP tasks is an approach of transfer learning and it has achieved great success in many NLP problems. This leads to a very important strategy for using deep learning models: **pre-training**. From the perspective of training, pre-training helps to provide a better model initialization, which leads to a better generalization performance and a speedier optimization on the target task [110]. From the perspective of transfer learning, pre-training on a huge text corpus can learn better language representations than random initialization, which eventually helps with the target tasks [107]. So, transfer learning is a technique that is now implicitly or explicitly adopted in

almost every NLP task, including text classification. The issue for transfer learning now is not when to transfer, but what to transfer and how to transfer.

2.3.3 What to Transfer?

Embeddings: The most popular transferred feature is word embeddings learned via unsupervised learning, such as word2vec, GloVe and fastText. These word embeddings capture the semantic meanings of words and can be used to replace randomly initialized word embeddings in deep learning-based models, they have vastly improved the performance of text classification approaches. All of these word embeddings are learned via unsupervised training, while some studies learn certain task-specific embeddings and then use those embeddings for other tasks. For example, Tang et al. [2] present a method that learns sentiment-specific word embedding for Twitter sentiment classification. Unsupervised word embeddings learn only the semantic meanings of words and ignore their sentiments; for example, the words "good" and "bad" tend to be very close in a vector space while their sentiment meanings are actually opposites. The sentiment-specific word embedding [2], are learned from massive distantly supervised Tweets collected by positive and negative emoticons. Then, sentiment-specific word embeddings are used for Twitter sentiment classification. This achieved the best performance in their experiments. In Liu et al. [111], topical word embeddings are learned by employing latent topic models to assign topics for each word in the text corpus. Then, topical word embeddings are evaluated on contextual word similarity and text classification.

Pre-trained model: The typical method of transfer learning adopted in modern deep learning approaches includes two steps: pre-training (i.e., a language model is pre-trained on a large corpus) and fine-tuning (i.e., the pre-trained language model, with an additional task-specific layer on top of it, is fine-tuned on downstream tasks). In recent years, an increasing number of works have directly adopted all layers of a pre-trained deep learning model, especially neural language models, for target text classification tasks. By pre-training on a large corpus, a neural language model is expected to store vast amounts of linguistic knowledge that can then be transferred to downstream tasks. For example, in BERT [24], the model is firstly pre-trained on two unsupervised tasks, masked language modeling and next sentence prediction,

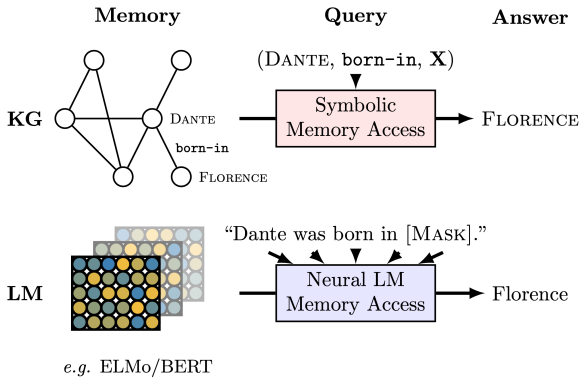


Figure 2.8: Querying knowledge bases (KB) and language models (LM) for factual knowledge (Source: [112]). To answer the question,"Where was Dante born?", a conventional model needs a pre-built KG, while a pre-trained BERT is able to directly predict the answer because it has stored the relational knowledge in pre-training.

then fine-tuned on downstream tasks, such as sentiment classification and natural language inference. In XLNet [42], the model adopts a generalized autoregressive pre-training method and is pre-trained using a larger unlabeled text corpora including Wikipedia, BooksCorpus, Giga5, ClueWeb, and Common Crawl, before being fine-tuned on downstream tasks including several text classification tasks. Apart from linguistic knowledge, Petroni et al. [112] find that BERT is also able to store relational knowledge in the training data comparable to that of knowledge bases, which in practice usually need to be extracted from text and require complex NLP pipelines. For example, as shown in Figure 2.8, to answer the query "(Dante, born-in, X)", a conventional model needs to search for the answer in a gold-standard relational data while BERT is able to predict the answer without schema engineering and human annotations. Taking advantage of the relational knowledge learned by BERT for open-domain question answering (open-domain QA) demonstrates very surprising performance even without fine-tuning. In other cases, a pre-trained model can be used as a subnet of another deep learning model. For example, Ostendorff et al. [113] present a document classifier that uses the concatenated features from a pre-trained BERT, meta data of books and author embeddings. Wu et al. [114] present a hybrid model that is a combination of two pre-trained models, namely a pre-trained BERT and a pre-trained CNN, for sentiment classification.

Shared layers: Another popular method is to transfer selected layers from a model trained

on a source domain to a target domain, or to train those selected layers simultaneously on both source and target domains (multi-task learning). In natural language inference, the available training data is mostly in a single language (e.g., English or Chinese), and other languages face a typical problem: the lack of supervised data. Conneau et al. [115] present a cross-lingual transfer learning-based approach tackling this problem, using the pre-trained encoder in English to train encoders in other languages using parallel data. Their proposed methods has shown very encouraging performance. In particular, it is much more efficient than machine translation-based baselines. Eriguchi et al. [116] present another cross-lingual transfer learning-based approach for sentiment classification and natural language inference. Their approach reuses the encoder of a pre-trained multilingual neural machine translation model to initialize the encoder for other NLP tasks. It leverages pre-trained multilingual representations for downstream text classification, which provides significant improvements. Liu et al. [117] present a multi-task learner (MT-DNN) that learns text representations across multiple natural language understanding tasks. The lower layers of the MT-DNN model are shared across all tasks, while the top layers are trained separately on each task. In terms of the training process, MT-DNN has two steps: pre-training and multi-task learning. Li et al. [118] present another interesting multi-task learner, the hierarchical attention transfer network (HATN), which aims to transfer attentions for emotions across domains; the framework is shown in Figure 2.9.

2.3.4 How to Transfer?

When we have decided what features will be transferred, the next step is to determine how we will transfer those features. Currently, the dominant method of transferring knowledge in deep learning-based approaches can be summarized in a two-stage process: pre-training and fine-tuning. In Goodfellow et al. [110], **pre-training** involves training models on simple tasks before confronting the challenge of training the desired model to perform the desired task. In the **Fine-tuning** stage, a joint optimization algorithm is involved to seek an optimal solution to the full problem. This two-stage approach has become the main adaption method for using neural language models in many NLP tasks, including text classification and has achieved impressive performance, as in [24] and [42]. An alternative to fine-tuning in using pre-trained

models is to freeze the pre-trained parameters; this is known as **Feature Extraction**. In addition, feature extraction also requires a task-specific architecture, while fine-tuning usually requires the addition of a task-specific output layer on top of the pre-trained model [119]. Peters et al. [119] investigate how to best adapt the pre-trained model to a given target task. From the results of experiments across diverse NLP tasks, they find that the relative performance of fine-tuning and feature extraction depends on the similarity of the pre-training and target tasks. Specifically, they find the largest differences for sentence pair tasks, where feature extraction consistently outperforms fine-tuning when using a pre-trained ELMo. For BERT, fine-tuning significantly outperforms feature extraction on most sentence pair tasks, such as paraphrase detection (PD) and semantic textual similarity (STS). At the same time, there are no big differences for the other tasks, such as named entity recognition (NER), sentiment analysis (SA), and natural language inference (NLI).

Sun et al. [41] investigate more sophisticated fine-tuning approaches for text classification tasks. For example, they experiment with different learning rates and decay factors for different layers. They find that a lower learning rate to the lower layer is effective to fine-tune BERT. They also introduce an intermediate training stage (**further pre-training**) between pre-training and fine-tuning to further pre-train a model within-task, for example, further pre-training a pre-trained BERT on the target domain using unsupervised LM objectives. They find that further pre-training within-task is able to improve the performance of BERT for target tasks. They also investigate a pre-trained BERT's performance after further pre-training it with in-domain or cross-domain data and find that both of these improve performance. For example, to work on movie review sentiment analysis (e.g., using IMDB), a BERT model is first pre-trained in the general domain corpus using unsupervised LM tasks. Then the pre-trained BERT is further pre-trained on, for example IMDB (within-task), other sentiment tasks such as Yelp reviews (in-domain), or on tasks from a different domain such as AG News (cross-domain), using unsupervised LM tasks. After completing both pre-training and further pre-training steps, the model is then fine-tuned on the target task, in this case IMDB.

Multitask learning is a learning approach to leverage training examples of multiple related tasks to improve generalization. In multitask learning, part of the model is shared and

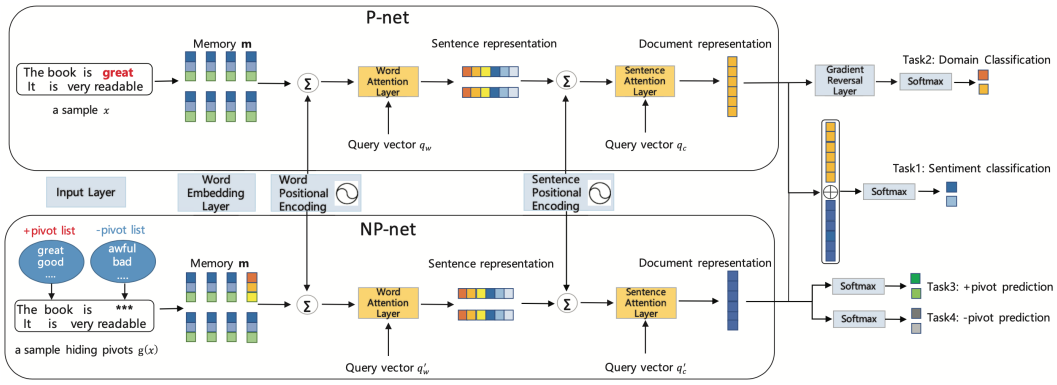


Figure 2.9: The architecture of HATN (Source: [118]). Word-level and sentence-level attentions are transferred from P-net to NP-net through positional encodings.

updated across tasks, which results in better generalization [110]. The setting of multitask learning is similar to that of transfer learning, but there are differences. Multitask learning treats all the tasks equally, while in transfer learning the target task is prioritized above other tasks [120]. Multitask learning is frequently adopted in a deep learning framework to perform knowledge transfer across different tasks, also known as **multitask transfer learning**. Liu et al. [117] present a multi-task deep neural network (MT-DNN) for learning representations across multiple NLP tasks. In MT-DNN, they incorporate a pre-trained BERT as the shared text encoding layers, while the top layers are task-specific. At the multi-task learning stage, the parameters of both shared layers and task-specific layers are updated according to the sum of all multi-task objectives. At the experiment stage, MT-DNN was evaluated on three NLP benchmarks: GLUE, SNLI and SciTail; it was that MT-DNN consistently outperformed $BERT_{LARGE}$ on all benchmarks. Li et al. [118] present a hierarchical attention transfer network (HATN) for cross-domain sentiment classification. HATN has a different way of utilizing multitask transfer learning. As shown in Figure 2.9, HATN has two subnets, P-net, for capturing pivots (domain-shared sentiment words, e.g., "good" and"excellent") and NP-net, for capturing non-pivots (domain-specific sentiment words, e.g., "readable" and "thoughtful" in the domain "Books"). The training process consists of two stages: first the P-net is trained for cross-domain sentiment classification, and second, the P-net and NP-net are jointly trained for cross-domain sentiment classification, where parameters for the sentiment classifier, domain

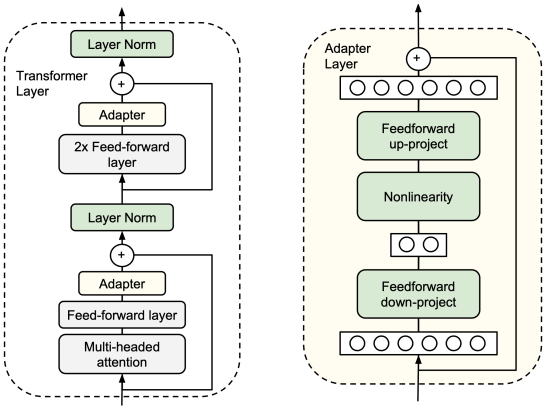


Figure 2.10: The architecture of Transformer with added adapters (Source: [122]). The diagram on the left shows how adapters are added into a Transformer cell, and the diagram on the right shows the detail of an adapter. During fine-tuning, only the adapter, the layer normalization parameters, and the final classification layer are trained, and the remaining layers are kept frozen.

classifier, and +(or positive)/-(or negative) pivot predictors are optimized jointly. The word positional encoding and the sentence positional encoding are shared and jointly learned during the training process by P-net and NP-net. Because positional encoding is consistent with attentions, attentions learned by P-net can be transferred to NP-net.

Transfer with adapter modules is another transfer learning approach based on adapter modules [121]. Adapters are new neural layers that can be added between layers of a pre-trained network. When using a pre-trained model for downstream tasks, feature extraction does not require re-training the pre-trained model's parameters while fine-tuning does. For **adapter tuning**, only parameters of the adapter need to be updated and parameters of the pre-trained model remain frozen. Houlsby et al. [122] propose a novel transfer learning method for NLP, which has adapters added into the pre-trained model. In Figure 2.10, the diagram on the left shows two adapters added into a Transformer. During the training processing on downstream tasks, only the adapter, the layer normalization parameters, and the final classification layer are trained, while the remain layers are kept frozen. The diagram on the right shows the detail of an adapter. The adapters first project the original high-dimensional features into a smaller dimension and then apply a nonlinearity. After that, they project features back to high dimensions. The total number of the adapters' parameters is approximately 0.5% to 8%

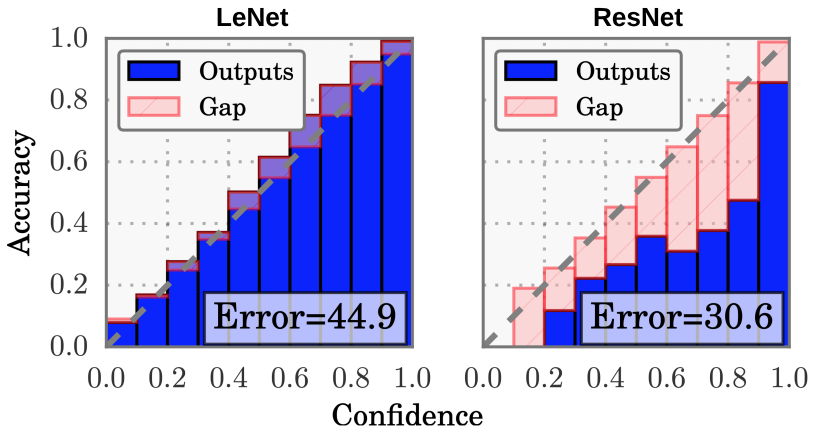


Figure 2.11: LeNet is well-calibrated, as confidence closely approximates the expected accuracy, while ResNet is not, as confidence is much higher than the expected accuracy (Source: [123]).

of the original model's parameters, which is relatively small. The performance of the proposed method is close to full fine-tuning. In terms of the total number of parameters, full fine-tuning on nine GLUE benchmark tasks has nine times the parameters of the original model, while adapter tuning only has around 1.2 times the parameters of the original model. Moreover, unlike multitask learning, adapter tuning does not require simultaneous access to all tasks during training.

2.4 Reliability of Text Classifiers

2.4.1 Should You Trust Your Text Classifier or Not?

Deep learning has achieved great success in predictive accuracy for many text classification tasks, but can we trust its predictions? Guo et al. [123] discover that modern neural networks have dramatically improved neural network accuracy, but are poorly calibrated. For example, in Figure 2.11, Guo et al. [123] compare two neural networks, a 5-layer LeNet [86] (an early age neural network) and a 110-layer ResNet [90] (a modern neural network), on the CIFAR-100 dataset. They find that ResNet is better than LeNet in terms of accuracy, but that LeNet is well-calibrated, as confidence closely approximates the expected accuracy while ResNet does not. A neural network should provide a calibrated confidence measure in addition to its

prediction. In other words, the predicted class probability should also reflect its ground truth correctness likelihood. For example, given a test example in a binary classification, where the predicted probability of being positive is 0.9, we expect that model should be 90% confident to assign a positive label to this test example. According to Figure 2.11, when ResNet's confidence is around 0.8, its accuracy is just above 0.4, which is much less than 0.8. With confidence at 0.8, ResNet should be 80% confident about its prediction; however, its accuracy is very dissatisfying, demonstrating that ResNet is poorly calibrated and that the generated probabilities do not reflect truth correctness likelihood. In other words, ResNet is overconfident about its predictions.

With the fast development of machine learning approaches for NLP applications, particularly for applications that interact with human users such as chatbots and automatic question answering (QA), AI safety is paid insufficient attention, and a real-world application can have serious consequences. For example, if the emotion system of an AI chatbot fails to detect a user's emotions correctly, it can hurt the user and create a bad user experience. QA systems may produce overconfident incorrect answers without controlling prediction reliability and may significantly damage users' search experiences [34]. Where QA systems are concerned with searching for answers in medical or legal domains, unreliable predictions can be even more harmful. If those systems mistakenly classify the non-spam or non-scam emails in spam or scam email detection systems, this could cause user to miss important emails. However, when a system can generate accurate confidence scores or well-calibrated probabilities that reflect truth correctness likelihood, users can perform risk management of AI.

2.4.2 Approaches to Reliable Classification

The rejection model [124] or selective classification [125] is a technique proposed to reduce the misclassification rate for risk-sensitive classification applications. The risk-sensitive prediction framework consists of two models: a classifier that outputs class probabilities given an example, and a rejection model that measures the confidence of its prediction and rejects unconfident predictions. Fumera et al. [126] incorporate the reject option in text categorization by allowing the classifier to withhold assigning categories to documents when the decision is

considered insufficiently reliable. Geifman et al. [125] claim that their work is the first work to adopt the rejection option in the context of deep neural networks.

Linear classifier probes, proposed by Alain et al. [127], can be inserted into the intermediate layers of a deep learning model to measure the performance of those layers. Some recent work utilizes linear classifier probes as a meta-model to measure the confidences of predictions from the base model, for example Chen et al. [128], where confidence scores are generated by liner probes to indicate the level of confidence about the predictions of the image classifier. Su et al. [34] introduce a risk control framework with PROBE-CNN model, a CNN-based probing method, to manage the uncertainty of deep learning models in Web QA. PROBE-CNN is a decision model that aims to learn a rejection region over the confidence score for selective output. The uncertainty of deep learning models in Web QA is controlled by rejecting a QA model's unreliable predictions.

Conformal predictor (CP) [129, 130] is a non-parametric machine learning model that is popularly adopted to obtain reliable predictions. Given an error probability ϵ, CP is guaranteed to produce a prediction region with probability $1 - \epsilon$ of containing the true label y, thereby offering a means to control the error rate. CP is typically applied to the output of other machine learning models, and can be used in both classification and regression tasks. There are two forms of CP in terms of the training scheme: inductive CP (ICP) and transductive CP (TCP). In addition, there is a special CP, the Mondrian conformal predictor which is proposed to handle imbalanced data. Conformal predictors have been applied to many different areas, from drug discovery [131–133] to image and text classification [134].

Calibration of deep neural networks focuses on the problem of predicted probabilities or confidence scores that do not reflect ground truth correctness likelihood. Methods for the calibration of deep neural networks commonly involve post-hoc processing that transforms uncalibrated predictions into well-calibrated predictions. Most require a hold-out validation set, which can be the same validation set used for hyper-parameters tuning. There are various post-hoc calibration methods, such as histogram binning [135], isotonic regression [136], Platt scaling [137], and temperature scaling [123]. Both isotonic regression and histogram binning are non-parametric calibration methods and work on uncalibrated probabilities, while both

Platt scaling and temperature scaling are parametric methods and work on the logit vector before the classification layer. Temperature scaling is a variant of Platt scaling for multi-class classification and is the most effective of calibration methods [123]. For simplicity, we take Platt scaling as an example to show how well-calibrated probabilities are generated in binary classification problems. Given the logit vector z_i (the vector generated by the layer just before the classification layer), the calibrated probability is generated by the following function:

$$\hat{q}_i = \sigma(az_i + b)$$

where a and b are parameters that can be optimized by minimizing negative log likelihood on the validation set. Because this is post-hoc processing, the parameters of the neural networks will not be changed.

Ensembles of neural networks are methods using ensembles of several networks for enhanced calibration. The intuition is that having more training data for training models helps reduce their bias and that using an ensemble of neural networks helps reduce the variance [138]. In practice, the ensemble similarly trains multiple neural networks on training data and uses the averaged probability as the final prediction. Lakshminarayanan et al. [138] find that increasing the number of neural networks in the ensemble significantly improves both classification accuracy and negative log likelihood (which is a popular metric for evaluating predictive uncertainty). Gal et al. [139] propose Monte Carlo (MC) dropout using dropout [140] at test time in neural networks to estimate predictive uncertainty. Dropout is usually used at training time to avoid over-fitting. When dropout is adopted at test time, the predicted probabilities will vary at different prediction times even for the same inputs, since random neurons will be dropped. The outputs of MC dropout are the averaging probability of an ensemble of the same model (the predicted probabilities of each model are different because of dropout).

2.5 Mining User-Generated Texts with Classification Models

Sentiment analysis is the task of recognize or detect the sentiment in user-generated content. Sentiments in user-generated texts express views or opinions towards a vast range of topics, from goods, movies, and celebrity news and gossip, to politics, social issues and economic and

cultural affairs. Sentiment analysis enables companies to better understand customers' views on their products and helps customers make shopping decisions by leveraging opinions from other customers [1]. Sentiment analysis is also important in political elections [141], providing the public and the media with new and timely perspectives on the dynamics of public opinion, and allowing election candidates to analyze public sentiments and target individuals with personalized political advertisements [142].

In the past decade, a considerable amount of academic research has been carried out in academia and some sentiment analysis-related data are released for developing more advanced sentiment analysis tools, for example **Yelp** reviews sentiment analysis [2], **IMDB** movie reviews sentiment analysis [3], the Stanford Sentiment Treebank (**SST**) [143], and Amazon reviews sentiment analysis [4]. The majority of research in sentiment analysis has formulated problems as classification tasks and presented various types of classification models, ranging from feature engineering-based machine learning models to deep learning-based models. For example, a feature engineering-based work Xia et al. [144] investigate two types of features, part-of-speech feature sets and word-relation feature sets, together with their ensemble framework for sentiment classification.

In recent years, the dominant methods for sentiment classification have been deep learning-based approaches. The state-of-the-art models in sentiment analysis [145, 146] barely include any feature engineering and usually choose to fine-tune a pre-trained language model, such as GPT [25] or BERT [24], on a sentiment data set. Before the widespread use of pre-trained language models, the most popular approaches for sentiment classification were: CNNs, such as CNN_Kim [9] and DPCNN [89], and attention-based RNNs, such as attention-based LSTM [95, 147].

Most of the work in sentiment analysis focuses on analyzing text in English. However, there is also some work tackling sentiment analysis in other languages, such as Chinese [148], Arabic [149], and Spanish [150], and multi languages [151]. Traditional **multilingual sentiment classification** approaches require a parallel corpus for each language in the classification task

[2]https://www.kaggle.com/yelp-dataset/yelp-dataset.
[3]https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews.
[4]https://www.kaggle.com/datafiniti/consumer-reviews-of-amazon-products.

via translation into English [152]. Classification models are then trained on the translated English corpus for each language. The effectiveness of this translation-based approach may be limited due to the fact that the classification model misses language-specific linguistic features and that it depends on the accuracy of translation. Recent deep learning-based approaches rely on cross-lingual word embeddings to transfer sentiment information from the source language to the target language, where cross-lingual word embeddings can be obtained without any cross-lingual resources in an unsupervised manner [153, 154].

A document/sentence may have multiple targets, and its author may have different opinions towards different aspects, for example "The new iphone is very fast, but it is also quite expensive!". **Aspect-based sentiment analysis** [155] is a sub-task of sentiment analysis, which classifies text opinions at the aspect-level. In recent years, classification models with pre-trained language models, have been widely adopted for aspect-based sentiment analysis [146, 156] and achieved excellent performance.

Affective computing is often studied along with sentiment analysis, focusing on endowing machines with the ability to detect and understand affective states and other social signals of humans [157]. Affective computing and sentiment analysis can be helpful for improving customer relationship management and recommendation systems [158]. For example, features that customers like or dislike can be analyzed through affective computing and sentiment analysis; then, the recommended items can be adjusted so that those with negative feedback are removed. **Sentic computing** [159] is a multi-disciplinary approach to sentiment analysis that leverages both computer and social sciences for sentiment analysis and opinion mining. More specifically, sentic computing leverages commonsense knowledge bases, such as Open Mind Common Sense [160] and WordNet-Affect [161], to interpret the cognitive and affective information associated with natural language [162].

Sarcasm and **irony detection** in social media are closely related to sentiment analysis, where the intended meaning of a given utterance is the opposite of what it literally expresses [15], for example "I love being ignored." Several datasets are publicly available, for example, Reyes2013 [163], Barbieri2014 [164], Ptacek2014 [165], Riloff2013 [16], Moh2015 [166], and SemEval2018 [167]. Classification approaches to irony detection can be roughly split into

three classes: rule-based approaches, classical feature-based machine learning methods, and deep neural network models. Rule-based approaches generally rely on linguistic features such as sentiment lexicons or hashtags to detect irony on Twitter [14, 168, 169]. Twitter uses hashtags to invert literal sentiment in Tweets [168]. The most popular hashtags for indicating irony include #irony, #sarcasm and #not [169]. The use of hashtags (e.g., #sarcasm) is believed to be a replacement for linguistic markers such as exclamations and intensifiers [170]. Classical feature-based machine learning approaches use hand-crafted features [171] for irony detection, such as sentiment lexicons, subjectivity lexicon, emotional category features, emotional dimension features and structural features. In recent years, deep learning-based approaches have been applied to irony detection, where (deep) features are automatically derived from texts using neural network models. Poria et al. [172] propose a CNN-based model for irony detection, which leverages extracted sentiment, emotion, and personality features. There are also several studies that use CNN-LSTMs [173, 174] for sarcasm detection. Another interesting work also focuses on detecting rhetorical questions and sarcasm using CNN-LSTM, but with an additional fully connected layer used for the purpose of leveraging linguistic inquiry and word count (LIWC) features [175]. The attention-based model is a popular approach that has been adopted for irony detection; for example one previous work propose a neural network with intra-attention for irony detection in social media, focusing on intricate similarities between each word pair in a sentence [176].

Hate speech detection [177] and **abusive speech detection** [178] are also important because both hate and abusive speech may have profound negative influences on the civility of a community or on a user's emotions. Hate speech and abusive speech in social media often use offensive and threatening words/phrases that target particular groups of people based on their religion, ethnicity, nationality, color, or gender, which may result in a divided society. Accurate hate and abusive speech detection will help social media platforms put an early stop to the spread of hate and abuse, ultimately supporting social solidarity. Both hate speech detection and abusive speech detection are often formulated as classification problems, and deep learning-based classification models have achieved promising results [179].

Product reviews are user-generated texts containing users' positive/negative opinions on

products, that may help other customers when shopping. However, some reviews may be designed to provide false information (e.g., false positive or malicious negative opinions), in order to affect customers decision-making; this is called opinion spamming [180]. **Opinion spam detection** is an important research area studying how to detect spam reviews, which is often treated as a classification task; again, deep learning-based models are the dominant approaches. For example, Ren et al. [181] propose a neural network model for detecting deceptive opinion spam, where it first learns sentence representation with CNN and then sentence representations are merged by a gated RNN as features to identify deceptive opinion spam.

Aspect-based opinion summarization is different from traditional text summarization as an aspect-based opinion summary is a summary of aspect-based opinions from multiple reviews [1, 182], which essentially gives a quantitative sentiment analysis on aspects of products, such as the number of reviews that are positive about a particular product's quality and the number of reviews that contain positive opinions about the battery life. The model for aspect-based opinion summarization is usually a classification model that classifies documents regarding each aspect; the classification results are then grouped and aggregated by aspects as the aspect-based opinion summary.

Opinion search and retrieval: have two typical types of opinion search queries on the Internet [1]: those seeking the opinions of the public on a particular entity (i.e., a smart phone), or an aspect of an entity (i.e., the battery life of a smart phone), and those seeking individual opinions on a particular entity or aspect of an entity. A typical opinion search and retrieval system has two stages: the first classifies whether a document is relevant or not to a given search query, while the second classifies the relevant documents as opinionated or not-opinionated. Besides, the opinionated documents can then be classified into positive, negative, or mixed (i.e., containing both positive and negative opinions), if sentiment analysis is required. Text REtrieval Conference (TREC) [5] is a well-known task for opinion search and retrieval. **Retrieval-based QA** is closely related to opinion search and retrieval; however, retrieval-based QA tends to include queries seeking not only personal opinions but objective

[5] http://trec.nist.gov/

information, such as product specifications. Retrieval-based QA is also different from open-domain QA and text comprehension which look for an answer (usually a text span) from a document or a paragraph. Retrieval-based QA usually focuses on searching for the most relevant sentence/phrase from a list of candidates as the answer. For instance, a classifier is usually adopted as a scoring or ranking function to score or rank candidate documents. There are a few retrieval-based QA applications, such as community QA, Ubuntu dialogue QA and Amazon product QA. **Community QA** [183] is a collection of questions and comments from the Qatar Living forum, and the task is to rank these comments according to the relevance with respect to a given question. **Ubuntu dialogue QA** [184] is a collection of almost one million multi-turn dialogues from the Ubuntu-related chat room for solving problems in concerning the use of the Ubuntu system, which is formulated as a best response selection task. **Amazon product QA** [22] is a collection of product questions and reviews from Amazon, and the task is to discover relevant reviews to answer questions.

Chapter 3

Language-Independent Twitter Classification via Convolutional Neural Networks

3.1 Introduction

As an internationally popular microblogging service, Twitter supports more than 40 languages, and about 50% of Tweets are non-English.[1] However, most research on Twitter classification including sentiment classification is only for English. To fully analyze Twitter content, it is therefore important to develop language-independent classification models for Tweets of multiple languages and of mixed languages.

Traditional multilingual text (document) classification approaches require a parallel corpus for each language in the classification task via translation into English [152]. Classification models are then trained on the translated English corpus for each language. The effectiveness of this translation-based approach may be limited as the classification model misses language-specific linguistic features and depends on the accuracy of translation. Language-independent Twitter sentiment classification is much more challenging as most resources such as lexicons, pre-trained word embeddings and annotated corpora are in English. Traditional

[1] http://latimesblogs.latimes.com/technology/2010/02/twitter-tweets-english.html

approaches to achieve language independence include n-gram based methods for Romance languages [185], and translation-based methods that require language identification [186]. Character-level CNNs [78, 88] have been proposed for language-independent text classification, and it has been demonstrated that when characters are used as input, the performance of CNNs is improved when the network is deeper [187]. However, most language-independent sentiment analysis models including CNN-based models are designed for European languages of overlapping alphabets and require language identification to build the alphabet to encode and quantize characters. In this chapter, we aim to develop deep learning-based models for sentiment analysis that does not require language identification.

In this chapter, we propose two CNN-based models, UniCNN and hybrid word-character CNN. Our first proposed model, UniCNN (Unicode UTF-8 encoded character CNN), is a fully language-independent character-based CNN for classifying Tweets of multiple languages and of mixed languages. To achieve complete language-independence, we propose to encode characters of all languages using the UTF-8 code. The advantages of using UTF-8 encoding instead of characters are:

- Language identification is no longer required. The UTF-8 code for characters is a universal alphabet for *any* languages.
- It allows neural networks to read Tweets (documents) of mixed languages. Since we are using the whole character set of UTF-8 as the alphabet, UniCNN can encode Tweets containing characters of different languages, for example Tweets in Chinese but containing some English phrases.

We experimented with UniCNN for Twitter sentiment classification and Twitter informativeness classification. The sentiment classification task includes six Twitter datasets of six different European languages. The informativeness classification task included a Twitter dataset of over 40 languages, with many Tweets in mixed languages. We benchmarked UniCNN against several baseline models. The Wilcoxon signed-rank test showed that UniCNN significantly outperformed character-based neural models and classical translation-based models using word-level feature unigrams and bigrams. UniCNN performed on a par with the translation-based

word-level neural model, a word-level neural model that uses external data sources for training word embeddings.

Additionally, we propose a hybrid word-character CNN model for language-independent Twitter sentiment classification, which leverages the benefits of CNN at both the word and character levels. Our intuition is that word-level and character-level deep features carry complimentary information and thus can improve the classification performance. To the best our knowledge, our proposed word-character CNN is the first CNN variant to tackle Twitter sentiment classification. Our proposed hybrid word-character CNN consists of two lookup tables, two embedding layers and two convolutional layers for short texts at word-level and character-level, separately. Hidden features that affect the sentiment polarity are extracted by two convolutional channels, which are merged before the final fully connected layer. We experimented with our hybrid word-character CNN for Twitter sentiment classification in six different European languages. According to our experiments, using two different short text transformations as the input of CNN for Twitter sentiment classification gives much better performance than using only words or characters.

3.2 Related Work

Character-based CNNs are the most related to our research [188, 189]. Character-based CNN models need an alphabet to encode and quantize characters to build the neural networks. All existing character-based CNN models are designed for the alphabet of European languages like English, German, Polish, and Spanish. To apply these models to multilingual text classification requires first identifying language, whether or not automatically, to build the alphabet to encode and quantize characters. In mixed-language documents any foreign characters outside the alphabet are ignored, but the foreign language characters may contain important signals for classification. Although not designed for text classification, a character-level CNN with highway network and bidirectional GRU for machine translation was proposed by Lee et al. [188]; this model accepts a sequence of characters in the source language and outputs a sequence of characters in the target language. It is shown that this character-based translation model can be better than subword-level translation model in the multilingual many-to-one translation

task (German, Czech, Finnish, and Russian to English). It is notable that Lee et al.'s model still needs to first perform language identification on the fly.

Another character-based CNN was proposed to focus on cross-language Twitter sentiment classification [189]. However, their character-based model is limited by the semantic and syntactic structural differences among languages and requires prior knowledge of the language to build the alphabet for developing the one-hot vector representation.

In the wider context of NLP research, since multiple languages are commonly seen in natural language applications, building cross-language models is highly desirable for several NLP tasks, including multilingual part-of-speech (POS) tagging [190], language-independent sentiment classification [185], and named entity recognition (NER) [190, 191]. Word-based neural models are popularly applied for these tasks. To achieve multilingual and cross-lingual analysis, tokenization and translation are generally needed. The model presented by Yang et al. [191] is based on GRUs and conditional random fields; it generates representation of words based on characters and is designed for leveraging morphological similarities between languages, such as English and Spanish. An attention-based LSTM neural model is proposed by Zhou et al. [70]; it is able to learn distributed document representations in Chinese and English. Gillick et al. [190] described an LSTM-based model that can read text as bytes rather than as characters or words. Their model is much more compact than word-based models, and can be used in POS tagging and NER in a multilingual environment without using a string tokenizer.

With traditional classification models based on feature engineering, to build multilingual or cross-lingual models, a typical method requires translation of texts from low-resource languages into a resource-rich language, usually English [152, 186], or the building of a parallel corpus that provides a bridge between languages. Other methods focus on creating language-independent features or models. The method proposed by Narr et al. [185] uses Naïve Bayes as the classifier and n-gram as features, and is mainly designed for languages that use spaces as word separators.

Overall, approaches for dealing with sentiment analysis-related tasks in languages other than English can be roughly classified into two categories. The first category of approaches usually requires language translation-based resources, while our proposed model does not need.

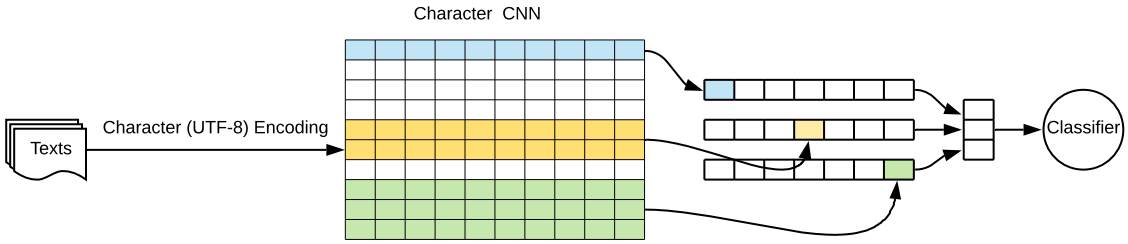


Figure 3.1: The architecture of our UniCNN. Each character of a tweet is first transformed into a sequence of UTF-8 integers and then mapped into a one-hot vector, followed by a matrix multiplication with the embedding matrix. Embeddings of each character are concatenated as a matrix passed into convolutional layers. Convolutional layers with different kernel size will generate a vector, followed by the global max pooling layer. The last layer is the prediction layer that generates probabilities. Based on probabilities, our model assigns a positive/negative label to a given tweet.

The second category of approaches directly process original text but at the character-level or using n-grams, which usually requires language identification. Our proposed method also processes original text at the character-level but using UTF-8 encoding, which does not need language identification. This eventually makes our model language-independent.

3.3 Methodology

3.3.1 UniCNN

Figure 3.1 depicts our proposed UniCNN model, which comprises two parts: the first part is the scheme for encoding characters with UTF-8 codes, and then the character CNN.

Character embedding with UTF-8 codes: The Unicode Standard is a character coding system designed to encode texts in different languages. It currently has an alphabet of 136,755 characters catering for 139 written languages on the earth. Unicode currently has three major forms: UTF-8, UTF-16, and UTF-32 (UTF-8 stands for Unicode Transformation Format: 8-bit). In our work, we use UTF-8, since Twitter only accepts UTF-8 encoded text and all other encodings must be converted to UTF-8.[2] Additionally, since 2010, UTF-8

[2]https://dev.twitter.com/basics/counting-characters

Table 3.1: Examples of characters and their numerical UTF-8 codes

!	*	+	2	9	<	?	@	A	H	Q	a	h
33	42	43	50	57	60	63	64	65	72	81	97	104

has become the dominant unicode format on the web.[3] UTF-8 uses one to four bytes for encoding characters. English uses one byte and actually the first 128 characters of UTF-8 have a one-to-one correspondence with ASCII, which contains not only English characters but also punctuations and numbers. Other languages such as Arabic, Chinese and Japanese, use two or three bytes. In our model, we use the corresponding decimal value of each character as the unit of input representation instead of the characters themselves. Table 3.1 shows some examples of characters with their corresponding numerical values in UTF-8.

In our work, each character is represented by its numerical value in UTF-8, which transforms a sentence or Tweet to a vector of n numerical values (n is the number of characters in a sentence or Tweet). The character cloud shown in Figure 3.2 is generated on dataset CrisisLexT26 [192], which will be discussed in more detail in the "Experiment" part of this chapter. As can be seen, the character diversity in the corpus of Tweets is beyond imagination. The traditional character-based CNNs will filter out most characters when there is a pre-defined alphabet. However, with the assistance of UTF-8 encoding, unexpected characters will be easily handled. Another advantage of this input transformation is that it does not require a step of building an alphabet, since it uses the whole set of UTF-8 characters as the alphabet. By applying UTF-8 encoding, each character is replaced by its UTF-8 integer value, then a sentence can be simply transformed to a sequence of numerical values regardless of what the written languages are and how they separate words.

Before approaching the embedding layer, a lookup table is built based on the whole dataset. After a Tweet has been transformed into a sequence of numbers using UTF-8 encoding, this sequence is mapped into one-hot vector through the lookup table and then passed to the embedding layer. The embedding layer learns the distributed character representation. In this layer, each one-hot encoding performs matrix multiplication with an embedding matrix

[3] http://www.utf-8.com

Figure 3.2: Character Cloud generated on experiment dataset CrisisLexT26. (The number of characters is massive, and the types of character are so diverse. However, each character in this figure can be mapped to an integer value by UTF-8 encoding.)

$W \in \mathbb{R}^{v \times d}$ (where v is the number of unique characters in the dataset, and d is the embedding dimensionality), and the results are concatenated as a matrix (see Figure 3.1). The embedding weights are randomly initialized at the beginning and trained in the neural network with the respect to the loss function. Additionally, a dropout layer is adopted after the embedding layer to reduce the over-fitting of character embeddings [193].

The architecture of character-based CNN: In our work, we use the traditional architecture of CNN which is convolution-and-pooling, with slight modifications (see Figure 3.1). It is commonly believed that the convolution-and-pooling architecture can capture local aspects that can be used for prediction tasks [194]. In terms of the pooling layer, we choose global max pooling, which returns the largest value in a convolutional feature map. For a given input $x \in \mathbb{R}^{d_{in}}$, the output of the convolutional layer is given by $p_i = g(x_i W + b)$, where $b \in \mathbb{R}^{d_{out}}$ is the bias and $W \in \mathbb{R}^{d_{in} \times d_{out}}$ is a matrix of weights. After the convolution-and-pooling layers, there are two fully connected layers. For the non-linearity function g we use a rectified linear unit (ReLU), and the regularization method in our model uses the most popular one, dropout. The results of many experiments show that the combination of ReLU and dropout performs very well [194]. The final fully connected layer produces output transformed by the softmax function. Finally, the loss function of our model uses cross-entropy loss $\sum_i y_i log(\hat{y}_i)$ (where y_i is the true distribution, and $\hat{y}_i$ is the predicted distribution) and the training optimizer we use Adam [195].

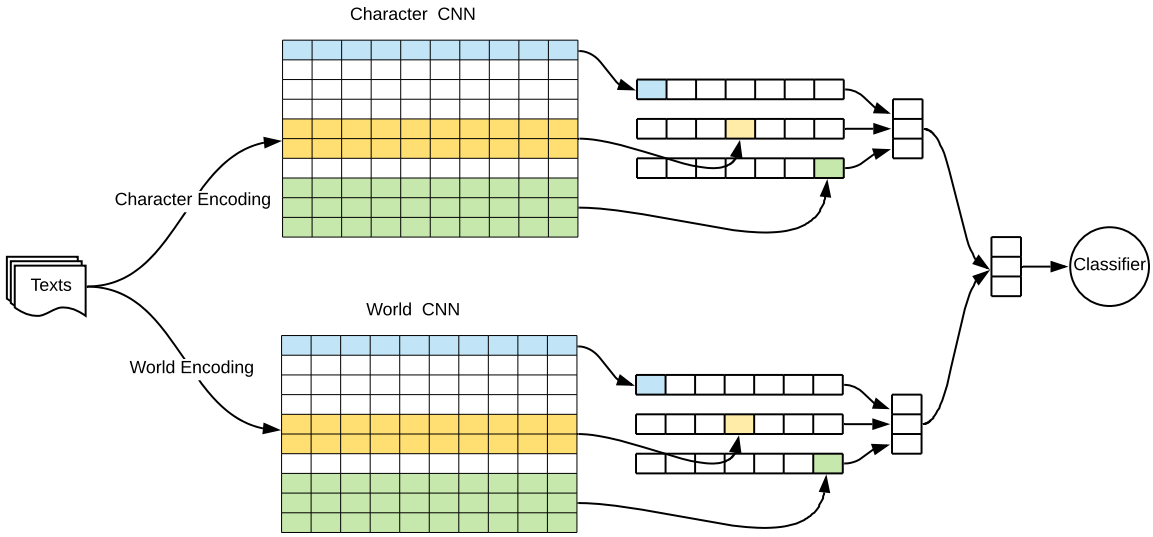


Figure 3.3: The architecture of our word-character CNN. Each piece of tweet is simultaneously processed by both word-based and character-based convolutional channels. The deep features of the two convolutional channels are concatenated in the merge layer. Then the output of merge layer is passed into the prediction layer. Word-character CNN assigns a positive/negative label to a given tweet based on probabilities generated by the prediction layer.

Character-based neural networks are believed to capture the similarities of sub-words (e.g., "cats" and "cat") better than word-based models [196]. The idea of the character-based model is to obtain word representations or character representations from character composition [78]. So, we take the advantages of character-based CNNs, and add Unicode encoding so as to bypass the burden of language translation and achieve real language-independence.

3.3.2 Word-Character CNN

Our word-character CNN (showed in Figure 3.3) is a hybrid CNN that takes advantage of both word-based CNN and our proposed character-based CNN (UniCNN). The intuition is that word-level deep features and character-level deep features are complementary in nature. In order to achieve language-independence, all models in our work (including baselines) do not need language identification, which means that short texts are directly processed by three models without stemming or other language-related feature engineering and without using

word embeddings (word2vec or GloVe) obtained from external sources. In all word-based CNNs, words are tokens that are separated by spaces in a sentence, while in character-level CNNs characters are regulated UTF-8 characters. In terms of the distributed representation of words and characters, we randomly initialize them with a given mean and standard deviation within a normal distribution.

Our word-character CNN has two convolutional channels: word-based convolution and character-based convolution. For an input Tweet t, the output of the embedding layer is a matrix that is passed to the corresponding CNN channel. The output of the convolutional layer is given by $p_i = g(x_i W + b)$ (where b is the bias, and W is a matrix of weights), which are sequences of size $m - h + 1$ at the word level and $n - h + 1$ at the character level (where h is the size of the filters). Then the convolutional feature maps are the input of the global max pooling layer, and the output is a sequence of largest values from each convolutional feature map, which is of size h. After the convolutional layer and the fully connected layer, we include a merge layer, which can perform concatenation or pair-wise operations on both word-level and character-level deep features (e.g., sum, multiply, or average). In our work, we use vector concatenation, and the output of the merge layer is a concatenated sequence of outputs from two CNN channels. Importantly, a dropout layer is included after the embedding layer, convolutional layer and the fully connected layer to reduce over-fitting. The final fully connected layer transforms the output of the merge layer by the softmax function, which results in one value for each class. Since we are focusing on sentiment classification, the loss function we apply is cross-entropy $\sum_i y_i log(\hat{y}_i)$ (where y_i is the true distribution and $\hat{y}_i$ is the predicted distribution).

3.4 Experiment

3.4.1 Dataset

We evaluate our character CNN on two Twitter classification tasks: Twitter sentiment classification and informativeness classification, where Twitter sentiment classification is used for language-independent evaluation and informativeness classification is used for evaluating mod-

Table 3.2: Language-independent dataset statistics.

Classification	Dataset	# Tweets	Positive	Negative
Sentiment	English	31,151	16,582	14,569
	German	29,570	17,269	12,301
	Polish	89,389	53,324	36,065
	Slovak	35,987	24,367	11,620
	Slovenian	42,264	20,325	21,939
	Swedish	21,739	9,313	12,426
Informativeness	CrisisLexT26 (40+languages)	27,968	16,841	11,127

els on handling mixed languages. UniCNN is evaluated on both tasks while Word-Character CNN is evaluated only on Twitter sentiment classification. The following section will introduce the dataset used for each task, data preprocessing, and evaluation. For Twitter sentiment classification, we use human-annotated Twitter corpora from Mozetivc et al. [151]. This dataset has 1.6 million annotated Tweets in 13 European languages, which might be the largest Twitter sentiment collection in current existence. In our work, we pick Tweets in six languages from these corpora, namely English, Polish, German, Slovak, Slovenian, and Swedish, since these subsets have acceptable self-agreement and inter-agreement rates [151] (self-agreement is the agreement for multiple annotations by the same annotator, inter-agreement is the agreement for multiple annotations by different annotators). The high self-agreement and inter-agreement rates imply a high quality for the annotated work. The basic statistics of the used sentiment dataset are given in Table 3.2. At the step of data pre-processing, we removed hashtags, URLs, and user mentions, since these elements are noisy characters when applying character-based CNNs.

3.4.2 Experiment Setup

Each dataset is randomly divided into two parts, with 80% used for training and 20% for testing. We evaluate the accuracy of classification models.

The parameter settings for CNNs are as follows. For the hyper-parameters settings of CNNs and embedding layer, we performed a grid search by investigating the following values: number of filters $\in$ {256,512,1024,2048}, dropout rate $\in$ {0.2,0.3,0.4,0.5}, and character/word

embedding dimensions ∈ {50, 100,150}. Based on experimental results on the English sentiment dataset, we decided to use, 1024 filters for the first convolutional layer, 2048 filters for the second convolutional layer, and 0.4 and 0.2 as the dropout rates for the dropout layer after the embedding layer and the dropout layer after two convolutional layers, respectively. The dimensions of both character and word embedding have been set to 100. For LSTMs, both word and character embedding dimension settings are the same as for the CNNs, while the number of state dimensions is set to 25. All neural models are programmed using a Python deep learning library (Keras) with Tensorflow as the backend, and each model is trained on a single GTX 1080TI with 11GB RAM.

We compare UniCNN against the following baselines:

- SVM_Unigram and SVM_Bigram. In our work, we apply word-based n-grams. Specifically, we choose unigrams (one-word) and bigrams (two-words). For the classifier we use support vector machine (SVM) with a linear kernel. The data representation is TF-IDF, which is a well-known weighting scheme in NLP. We name the two separate approaches *SVM_Unigram* and *SVM_Bigram*.

- Translation-based neural models. It appears necessary to translate all Tweets from source languages into a target language, which is English in this case. The translation work is completed by using the Google cloud translation API. In our work, there are two translation-based models, namely TransCNN(word) and TransCNN(char) for both experiments. *TransCNN(word):* uses our standard CNN as the classifier, but with Glove2Vec [75] as the input representation, which is pre-trained word vectors. Pre-trained Glove2Vec has several different versions, such as vectors trained on Wikipedia data or vectors trained on Tweets. In our experiment, we use word vectors trained on 2 billion Tweets, giving us 27 billion tokens (or words). Additionally, when training the neural model, word vectors are left to be trainable. *TransCNN(char):* also uses our standard CNN, but the input representation uses the one-hot vector of each character, which is different from former model. The alphabet for one-hot encoding contains 94 characters, the list is given below.

```
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJK
```

```
LMNOPQRSTUVWXYZ<=>?@:;!\"#$%&'()*+,-./ []^_`\\{|}~
```

Additionally, it also uses our standard embedding layer, which uses the randomly initialized weights that will be trained with respect to the loss function.

Table 3.3: Accuracy of UniCNN versus the other approaches.

	UniCNN	Tran-sCNN(char)	Tran-sCNN(word)	SVM_Unigram	SVM_Bigram
Translation	No	Yes	Yes	No	No
Tokenization	No	No	Yes	Yes	Yes
Sentiment Classification					
English	0.7794	0.7825	**0.7957**	0.7623	0.6049
German	**0.7577**	0.7274	0.7367	0.7291	0.6031
Polish	**0.7908**	0.7682	0.7324	0.7220	0.6251
Slovak	0.7545	0.7309	0.7260	**0.7628**	0.7035
Slovenian	**0.7659**	0.7379	0.7623	0.7438	0.5752
Swedish	**0.7752**	0.7432	0.7218	0.7403	0.6178
Informativeness Classification					
CrisisLexT26	0.8418	0.8345	**0.8503**	0.8412	0.7937
p-value	-	0.0313	0.1563	0.0469	0.0156

3.4.3 Results for Character CNN

In this section, we report and analyze the results of all models listed in Table 3.3. Not only are the results for our UniCNN are compared with those for the other four models, but comparison is also made between translation-based models and between n-gram based models and neural models. We also applied the Wilcoxon signed-rank test [197]. The Wilcoxon signed-rank test p-value of our UniCNN with the other four approaches are also shown in Table 3.3.

UniCNN vs. other models: Table 3.3 shows that, in terms of classification accuracy, UniCNN is better than TransCNN(char), the translation-based character CNN, and SVM_Unigram and SVM_Bigram, the traditional SVM models using unigram and bigram features. UniCNN has comparable accuracy with Trans_CNN(word), the translation-based word-embedding CNN model. The superior performance of UniCNN over TransCNN(char) can be attributed to the limitations enforced by translation, and that over the SVM models shows

the power of deep features extracted from the CNN. UniCNN's comparable performance with TransCNN(word) can be explained by the semantic structure representation brought about by the word vectors.

On the other hand, it should be noted that UniCNN is a fully language-independent model that does not require translation or tokenization. In contrast, all other models require some form of translation or tokenization, which incurs extra computation.

N-grams vs. neural models: In general, all translation-based neural models and our UniCNN performs better than n-gram based models in most experiments, but SVM_Unigram can still achieve acceptable results. Especially in the informativeness task, SVM_Unigram is fairly comparable with the state-of-the-art model, a CNN with pre-trained word vectors, and our UniCNN. However, it must be noted that the CrisisLexT26 has been translated into English before the application of n-gram models. Unlike sentiment datasets, which use space as the word separator, CrisisLexT26 is a mixed-language dataset whose languages (e.g., English and Chinese) are different in structure. In sentiment tasks, the results for SVM_Unigram are not much worse than those for neural models, even achieved the highest accuracy in the Slovak sentiment task. However, the results for SVM_Bigram are much lower than those for other models in almost all tasks. In addition, using GPU and Tensorflow effectively facilitates the training procedure of the neural network. The whole training process takes several minutes each neural model, while n-gram based models take more than an hour or even several hours, depending on the data size.

TransCNN(word) vs. UniCNN: Using pre-trained word vectors as the input and re-training them in neural models is a method that has been widely adopted in many NLP tasks, and is the state-of-the-art technique for NLP classification tasks. In our experiment, the results are consistent with the current trend. In the English sentiment task, TransCNN(word) achieved the highest accuracy (0.7957), while our UniCNN achieved 0.7794. However, when other languages are translated into English, the results for TransCNN(word) are much lower than results for our UniCNN. For example, in the Polish sentiment task, the accuracy of our UniCNN is 0.7908, which is 0.05 higher than the accuracy of TransCNN(word) (0.7324). In fact, our UniCNN outperformed all baselines in all sentiment tasks apart from the English sentiment

task, which indicates that when there are not pre-trained word vectors available in the source language, using UniCNN is more likely to achieve better results than using translated texts with the state-of-the-art model. On the other hand, since pre-trained word vectors are trained on a large dataset in an unsupervised manner for learning the semantic meaning of words, using pre-trained word vectors can be an effective way to improve the accuracy of various NLP classification tasks.

TransCNN(char) vs. UniCNN: Both are character-based neural models, but with differences in input and alphabet. The first difference is that the input text is the translated Tweets for TransCNN(char), while for UniCNN the input are the Tweets in the source language. The second difference is that TransCNN(char) has a pre-defined alphabet of characters, while UniCNN does not. Generally, our UniCNN outperforms TransCNN(char) across all tasks except the English sentiment task. It is mainly because the English sentiment dataset is not a translated dataset. Therefore, using UniCNN is likely to achieve better results than TransCNN(char), when translation is needed.

TransCNN(word) vs. TransCNN(char): Both are translation-based models, but TransCNN(word) is a word-based neural model, while TransCNN(char) is a character-based neural model. The results of these two translation-based neural models are not constant. In English, German, and Slovenian sentiment tasks, TransCNN(word) slightly outperformed TransCNN(char), while TransCNN(char) performed better in the other three sentiment tasks. Why the results for word-based and character-based models are not constant is uncertain; however we can conclude that with translated text as the training and testing datasets, there are not many differences between the word-based CNN and the character-based CNN.

For the informativeness task, the experiment results for all these models except SVM_Bigram are reasonably close. In terms of resource consumption, our UniCNN is clearly the winner, since it is translation-free. Translation-based models require much higher consumption in terms of time and money. The cost of the Google Translation API is $20 per million characters and the translation speed is roughly two Tweets per second. In contrast, our UniCNN is much faster and incurs zero cost.

Table 3.4: The accuracy of our word-character CNN versus word-based CNN, character-based CNN, word-based LSTM, and character-based LSTM. The results for word-based CNN, character-based CNN, and word-based LSTM reported in previous work [189] are in brackets.

	word-char CNN	word-based CNN	char-based CNN	word-based LSTM	char-based LSTM
English	**0.7936**	0.7817 (0.766)	0.7704 (0.762)	0.7737 (0.770)	0.7246
German	**0.7895**	0.7464 (0.733)	0.7541 (0.755)	0.7361 (0.734)	0.6792
Polish	**0.8119**	0.7494	0.7948	0.7513	0.7624
Slovak	**0.7713**	0.7478	0.7592	0.7490	0.7220
Slovenian	**0.7807**	0.7718	0.7669	0.7630	0.6966
Swedish	**0.7873**	0.7587	0.7686	0.7438	0.7102

3.4.4 Results for Word-Character CNN

We compare our word-character CNN with the following baselines: word-based CNN [9] (but without using pre-trained embeddings, for language-independent purposes), character-CNN (our UniCNN and another character-CNN [189]), word-based LSTM, and character-based LSTM. We report the experiment results in two parts: (a) Experiment results for word-character CNN and our baselines and (b) Experiment results for word-based CNN and character-based CNN.

Word-character CNN *vs* baselines: In Table 3.4, it is clear that our word-character CNN significantly outperforms both word-based CNN and character-based CNN across all six datasets (the p-values of Wilcoxon signed-rank tests for our word-character CNN against all baselines are less than 0.05). Specifically, the average improvement of word-character CNN over word-level CNN is 0.03, and the average improvement of word-character CNN over character-based CNN is 0.015.

Additionally, we report the results for word-based CNN, character-based CNN, and word-based LSTM in a previous work [189] in brackets. According to Table 3.4, we find that our word-character CNN outperforms both character-based CNN and word-based CNN, as reported in previous work [189]. However,it should be noted that the results for our word-based and character-based CNNs and word-based LSTM are nearly identical to the results of theirs [189].

Word-based *vs* character-based CNNs: In Table 3.4 we also have a clear comparison between the word-based CNN with the character-based CNN. First of all, the performance

of the word-based CNN and the character-based CNN are not always consistent across six datasets. On the English and Slovenian datasets, the word-based CNN slightly outperforms the character-based CNN, while the character-based CNN performs better than the word-based CNN for all other datasets. Most interestingly, on the Polish dataset, the accuracy of both word-character and character-based CNNs is significantly better than that of the word-based CNN. This may be due to the grammatical differences between Polish and languages like English. Character-based CNNs have the advantage over word-based CNNs of capturing the different forms of a word. Therefore, the performance of word-based and character-based CNNs is very likely to be affected by language's specific features. However, it is clear that word-based and character-based CNNs capture different convolutional features for sentiment polarity, since they treat the same Tweets as different inputs for the convolutional layer.

The proposed word-character CNN treats a short text in two different forms, words and characters, and two convolutional channels are involved to detect the hidden features that affect the sentiment polarity. Most importantly, the experiments' results prove that taking advantage of deep features at both word-level and character-level benefits the CNN structure for Twitter sentiment analysis.

3.5 Summary

In this chapter, we have proposed two CNN-based models for language-independent Twitter classification. The first proposed model is a character-based CNN with Unicode UTF-8 encoding. Our proposal of UTF-8 character encoding does not require language identification and therefore is a fully language independent approach. Furthermore, the character-based CNN on learning distributed character representations leads to promising classification accuracy. We have tested our first method on language-independent Twitter sentiment classification and informativeness classification, which has mixed languages in Tweets. Experiments showed that our method is fully language-independent. Most importantly, our UniCNN model is the first neural model that can work with Tweets written in mixed languages. Lastly, our UniCNN does not need language knowledge to pre-define an alphabet.

Our second proposed model, word-character CNN, is proposed for language-independent

Twitter sentiment classification, which leverages the deep convolutional features of short texts at both the word and character levels. Based on our experiments, it is clear that using two different transformations of short texts as the input of CNN for Twitter sentiment classification gives much better performance than using only words or characters.

Chapter 4

Irony Detection via Sentiment-based Transfer Learning

4.1 Introduction

Compared to most text classification tasks, irony detection is a more challenging task [14] that requires inferring the hidden, ironic intent, which cannot be achieved by literal syntactic or semantic analysis of textual contents. Indeed, the challenge of irony detection is clearly shown in the sentiment polarity classification task of the Evaluation Campaign of Natural Language Processing and Speech Tools for Italian (Evalia 2016) [198]. Three independent sub-tasks are included, namely subjectivity classification, polarity classification, and irony detection. The performance on both subjective classification and polarity classification is over 10% higher than that on irony detection in terms of the F measure.

According to linguistics research, irony is the incongruity expressed between the context and statement conveyed in a piece of text [15, 199, 200]. Sentiment polarity contrast is a commonly seen form of irony on Twitter [15, 16]. For example, in the Tweet "I love when I wake up grumpy," "love" expresses positive polarity whereas the phrase "wake up grumpy" expresses negative polarity. The hidden sentiment polarity contrast signifies the ironic intent of the Tweet. Moreover, the extent of irony perception depends on the strength of the context ("wake up grumpy") and the strength of the statement ("love"). Explicit incongruity refers

to a contrast in explicit sentiment words, as in "I love being ignored," where "love" is positive and "ignored" is negative. Implicit incongruity refers to contrast within phrases expressing implicit sentiment polarity but not using explicit sentiment words, as in "I love this paper so much that I put it in my drawer"; the phrase "put it in my drawer" implies a negative polarity and forms a contrast with the positive sentiment of "love."

The task of irony detection is to classify a piece of text as ironic or non-ironic. Existing studies mostly formulate irony detection as a standard supervised learning text categorization problem. Approaches to irony detection on Twitter can be roughly classified into three classes, namely rule-based approaches, classical feature-based machine learning methods and deep neural network models. In the literature, rule-based and classical feature-based machine learning models are proposed for irony detection (See [201] and [202] for surveys). Over the past few years, deep learning models have been applied for irony detection [19, 21, 172, 174, 175, 203] and have shown better performance than classical feature-based machine learning models. Of all the neural network-based models, attention-based models are the most effective. Apart from standard attention models, one previous work [176] propose an intra-attention mechanism for sarcasm detection, with a layer looking into intricate similarities between each word pair.

Most previous studies do not study context incongruity for irony detection. A few studies focus on identifying context incongruity for irony detection, but with limitations. One previous study [16] makes use of the pattern "positive sentiment followed by negative situation" to detect irony on Twitter. This approach can miss many forms of context incongruity that do not follow this pattern. Another previous study [15] manually engineers explicit and implicit context incongruity features for irony detection and is still able to capture limited context incongruity.

Although ironic intent is mainly expressed by incongruity of sentiments between a context and a statement, the limited availability of annotated resources is a barrier to a model to fully detect that pattern given the extremely varied sentiment patterns present in human language. On the other hand, sentiment resources, which can be leveraged for irony detection, are widely and readily available. In this chapter we therefore formulate irony detection as a transfer learning task, where supervised learning on irony labels is enriched with knowledge transferred

from external sentiment analysis resources. Moreover, we focus on the key issue for irony detection—identifying hidden, implicit incongruity without explicit incongruity expressions, as well as explicit incongruity. Our key idea is to transfer external sentiment knowledge from sentiment resources to train the deep neural model for irony detection. Resources for sentiment analysis are readily available, including sentiment lexicons [204] and sentiment corpora [205].

We propose three sentiment-based transfer learning models to improve the attentive recurrent neural model for identifying explicit and implicit context incongruity for irony detection on Twitter. The three models are designed to transfer different types of sentiment knowledge. The first two methods are focused on transferring hard sentiment attention generated from a pre-defined sentiment corpus, but the hard attention in the first model is treated as an external feature, while in the second model it is treated as an extra supervision signal. The last model is focused on transferring deep features from the sentiment analysis on Twitter for the irony detection task, where features from both tasks are mapped into a common latent feature space. By comparing these different approaches, the most effective way of using sentiment-based transfer learning for irony detection can be found.

The main contributions made in this chapter are:

- We propose three novel methods for attention-based models to incorporate sentiment features, instead of feature-vector concatenation. Experiments show that the proposed methods are effective to improve the accuracy of attention models for irony detection.

- Learning deep features on sentiment Tweets corpora and transferring them into the attention-based neural model is the most effective way to detect both explicit and implicit context incongruity.

- We contrast human-labeled and hashtag-labeled datasets for the evaluation of irony detection models. To the best of our knowledge, this has not been done before. We find that the human-labeled dataset is much more challenging than the hashtag-labeled dataset and gives a more accurate estimation of the performance of irony detection models in real applications. We also discuss several possible reasons why irony detection is easier for the hashtag-labeled datasets.

4.2 Related Work

Approaches to irony detection on Twitter can be roughly divided into three classes, namely rule-based approaches, classical feature-based machine learning methods, and deep neural network models. Rule-based approaches generally rely on linguistic features such as sentiment lexicons or hashtags to detect irony on Twitter [14, 168, 169]. Twitter uses hashtags to invert literal sentiment in Tweets [168]. The most popular hashtags for indicating irony include #irony, #sarcasm and #not [169]. The use of hashtags (e.g., "#sarcasm") is believed to be a replacement for linguistic markers such as exclamations and intensifiers [170]. Classical feature-based machine learning approaches use hand-crafted features [171] for irony detection, such as sentiment lexicons, subjectivity lexicons, emotional category features, emotional dimension features, and structural features.

In recent years, deep learning-based approaches have been applied to irony detection, where (deep) features are automatically derived from texts using neural network models. Using the similarity score between word embeddings as features has shown improved performance for irony detection [203]. Nozza et al. [206] present an unsupervised framework for domain-independent irony detection, which takes the advantages of probabilistic topic models for discovering topic-irony and meanwhile uses word embeddings to improve generalization abilities. A convolutional neural network (CNN) was proposed in [172] for irony detection, which uses a pre-trained CNN for extracting sentiment, emotion, and personality features for irony detection. There are also several studies that use CNN-LSTM structures [173, 174] for sarcasm detection. Another interesting work focuses on detecting rhetorical questions and sarcasm also using CNN-LSTM, but with an additional fully connected layer for the purpose of taking LIWC features [175]. These existing studies use the convolutional network to automatically derive deep features from texts for irony detection. The results of these deep learning approaches are generally better than those of classical feature engineering-based approaches.

Attention-based RNNs have been proposed for irony detection [19–21] and other NLP tasks [11, 17, 18], with a great success. The self-attention mechanism is not directly targeted to identify context incongruity. Some previous work [20] studied emotion, sentiment, and sarcasm prediction, where the attention mechanism is not used specifically to detect context

incongruity. Some other previous work [19] studied irony detection for replies in social media conversations. The sentence-level attention mechanism is used to identify more informative sentences in conversations that trigger sarcasm replies. In addition, one previous study [21] focuses on irony detection in Tweets and employs the standard attention mechanism. However, the standard self-attention mechanism often generates attentions for only partial texts forming the context-statement contrast and thus fails to detect context incongruity (discussed in more detail in Section 3). Tay et al. [176] propose a neural network with intra-attention for sarcasm detection on social media, which focuses on intricate similarities between each word pair in a sentence. Almost all previous studies use a handful of human-labeled ironic Tweets for training; however, pattern recognition for detecting irony is so complex and difficult that a considerable size of dataset is needed. As it is costly to build a large annotated dataset for training a high-performance model, transfer learning with sufficient sentiment resources seems to be a practical alternative.

Irony detection via identifying context incongruity has been reported in the literature, but the proposed solutions very much rely on manually engineered patterns and features. In one previous work [16], sarcasm is identified via a pattern of "positive sentiment followed by negative situation", and a bootstrapping algorithm (originated from the word "love") to automatically learn phrases corresponding to the positive sentiment and negative situation, respectively. In [15], four types of manually engineered features, including lexical features, pragmatic features, implicit incongruity features, and explicit incongruity features, are used to train a model for irony detection. It must be noted that although irony detection needs to detect sentiment incongruity, it is different from detecting sentiment shift [207], where words and phrases change the sentiment orientation of texts, as in "I don't like this movie."

Existing studies [172] that use sentiment analysis resources for irony detection lack a principled approach to transferring the sentiment analysis knowledge. In Poria et al. [172], a comprehensive set of features, including sentiment, emotion, and personality features, are extracted from sentiment analysis resources for irony detection. The model combines all features before the prediction layer in the neural network, which makes it unclear whether the sentiment features benefit detecting context incongruity or irony detection. In contrast, our work

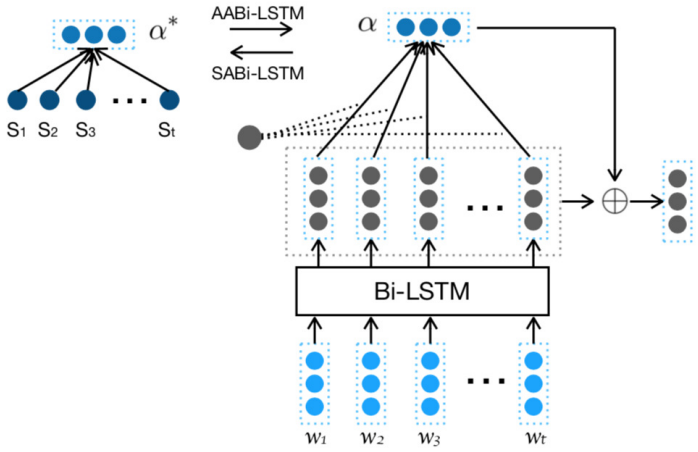


Figure 4.1: Sentiment attention Bi-LSTM models. AABi-LSTM: the model combines the hard sentiment attention with the learned soft attention. SABi-LSTM: the model treats the hard sentiment attention as a supervised signal.

not only use sentiment knowledge, but also considers the reasoning of how and why we incorporate sentiment knowledge in a neural network. Specifically, we devoted to using sentiment knowledge and resources with reasoning and visualization-focused interpretation to show how our models are detecting context incongruity.

In comparison with existing irony detection models, our model has two strengths. First, it is the first deep learning-based model that explicitly addresses context incongruity. Most deep learning-based approaches simply feed text into models and deep features are learned automatically. This usually works when a large volume of labeled datasets is available. Irony detection tasks often have limited labeled instances which makes it hard for models to detect context incongruity. Our model leverages transferred knowledge to help models detect context incongruity in an explicit way. In addition, our model works on attention values which can be used to interpret model's performance, while most deep learning-based models tend to focus on reporting accuracy only.

4.3 Attention-Based Bidirectional Long Short-Term Memory (Bi-LSTM)

In this section, we introduce attention-based bi-LSTM for the sake of understanding our proposed models. RNNs are designed to process sequences. LSTM is a commonly used RNN unit proposed by Hochreiter et al. [91] to overcome the gradient vanishing problem. In terms of network architecture, bidirectional LSTM [17], which has two layers of LSTM reading sequences forward and backward respectively, is widely used. The output of Bi-LSTM is a concatenation of forward and backward returned sequences:

$$h_i = [\overrightarrow{h}_i \parallel \overleftarrow{h}_i] \tag{4.1}$$

In the attention-based Bi-LSTM, H=$[h_1, h_2, ..., h_i]$ is a matrix consisting of output vectors produced by the Bi-LSTM, where i is the time step. The representation r of a Tweet is formed by a weighted sum of these output vectors:

$$M = \tanh(H) \tag{4.2}$$

$$\alpha = softmax(\omega^T M) \tag{4.3}$$

$$r = H\alpha^T \tag{4.4}$$

$$h^* = \tanh(r) \tag{4.5}$$

At prediction, we use $softmax$ to predict $\hat{y}$ for a Tweet. The goal of training is to minimize the cross-entropy error between the true label y_i and the predicted label $\hat{y}_i$:

$$\hat{y} = softmax(Wh^* + b) \tag{4.6}$$

$$loss^1 = -\sum_i \sum_j y_i^j \log \hat{y}_i^j \tag{4.7}$$

In equation 4.3, the vector α is the attention vector.

Bi-LSTM often fails to capture words and phrases crucial for building the ironic intent. This may be due to the inherent difficulty of the task and limited annotated training instances.

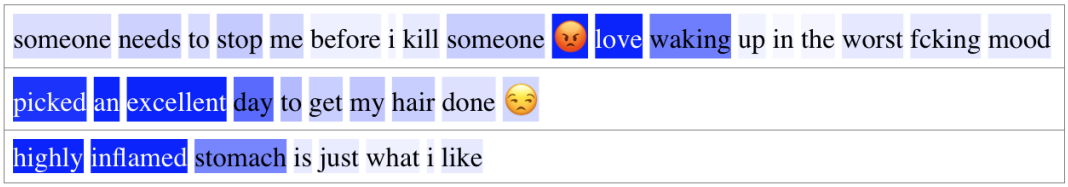


Figure 4.2: Examples of attention generated by the standard attention-based Bi-LSTM; the intensity of blue represents the attention value of each word.

As shown in Figure 4.2, with the first Tweet "someone needs stop me before i kill someone love waking up in the worst fcking mood," Bi-LSTM only placed strong attention on "loving waking," which indicates positive sentiment, and as a result failed to detect the negative sentiment expressed by "the worst fcking mood."

The failure of the standard attention mechanism to detect context incongruity is possibly caused by the inherent difficulty of the task and only relying on irony labels, which are limited. Moreover, even with attention, LSTM can only learn the long dependencies of the context. To detect context incongruity, we need to use external sentiment resources. In the next section, we describe our approach for enhancing the attention mechanism with sentiment knowledge transferred from the readily available resources for sentiment analysis.

4.4 Sentiment-Based Transfer Learning for Irony Detection

Transfer learning is an important machine learning technique that takes advantage of knowledge from solving one problem to solve other related problems, thereby potentially overcoming the burden of limited human-labeled resources. Learning deep features or abstract representation of input is the advantage of deep learning used with transfer learning [208]. Transfer learning based models are particularly useful for cross-domain tasks. Especially when a target domain has very limited data, there is a need to train a high-performance model using data in a source domain where data can be easily obtained [108]. Feature transformation can be completed by re-weighting a layer in the source domain to more closely match the target domain [209], or by mapping features from both source domain and target domain into a common latent feature space [210].

In our scenario, since detecting irony on Twitter is based on incongruity between the sentiments of a statement and a context, knowledge learned from the resources used for sentiment analysis will be incorporated into irony detection. Sentiment analysis resources are widely available, including sentiment words corpora [211, 212] and sentiment Twitter corpora [213, 214]. To improve the attention mechanism on detecting context incongruity, our proposed methods involve transferring sentiment knowledge from external resources, such as sentiment words corpora and sentiment Twitter corpora [204, 211, 212], as additional resources to enrich the limited body of human-annotated ironic Tweets. The challenge is how to represent and incorporate sentiment knowledge into the attention mechanism for irony detection.

In order to incorporate two different types of sentiment resources (i.e., sentiment word lexicons and sentiment Tweets corpora) into irony detection, we propose three different models to transfer different sentiment knowledge. The first two models incorporate sentiment word lexicons, where sentiment-based hard attention is generated to strengthen the attention distribution on sentiment parts; however, the two models employ different methods. The major difference between them is how the sentiment-based hard attention is incorporated. In the first model, the sentiment-based hard attention is treated as a feature, while in the second it is treated as a supervised signal. In contrast to our first two models, our third model is proposed to detect context incongruity using the transferred deep features from the model learned on a sentiment Twitter corpus instead of using sentiment-hard attention.

4.4.1 Sentiment-Augmented Attention Bi-LSTM (AABi-LSTM)

With the first model, the readily available sentiment words corpora [204, 211, 212] are used as additional resources to generate a sentiment distribution, which is then treated as hard attention and transferred into the soft attention mechanism in order to push the attention-based model to focus on context incongruity. In particular, our model incorporates both sentiment polarity and sentiment strength into the attention mechanism to capture the strength of incongruity in Tweets, based on the linguistic principle of "the extent of irony perception depends on the strength of context and statement" [199, 200].

In our model AABi-LSTM (shown in Figure 4.1), we first construct a sentiment hard

attention based on the sentiment of each word. The sentiment scores of each word are generated by using pre-defined sentiment corpora. For a given Tweet, the sentiment distribution $[\alpha_1^*, \alpha_2^*, \alpha_3^*, ..., \alpha_i^*]$ is generated by applying *softmax* on absolute sentiment scores of each word $[S_1, S_2, S_3, ..., S_i]$. Then, we transfer this sentiment hard attention into the attention-based model to enhance the attention to the sentiment part of a Tweet. In our first proposed mechanism, the sentiment attention vector is weighted and then added to the learned attention vector in the network, which results in directly strengthening the attention of the network on the sentiment part:

$$\alpha^* = softmax(|S|) \tag{4.8}$$

$$r = H(\alpha \oplus W\alpha^*)^T \tag{4.9}$$

4.4.2 Sentiment-Supervised Attention Bi-LSTM (SABi-LSTM)

In order to detect the complete contextual incongruity, we further propose to take advantage of the widely available sentiment Twitter corpora [213, 214] to improve the attention mechanism in a supervised manner so as to capture the complete context for incongruity. With our second model sentiment-supervised attention Bi-LSTM (SABi-LSTM), we learn the abstract representation of polarity embedded in expressions without sentiment words and transfer these learned features into irony detection model for learning context incongruity.

As shown in Figure 4.1, SABi-LSTM includes a sentiment attention mechanism that uses the sentiment hard attention in a supervised manner, which provides an alternative supervised signal to let the model learn features and attentions with reinforced attentions on sentiment parts. Technically, the attention value is used as an output of the model, and used in supervised training with sentiment hard attention as the true label. In order to let the network's attention be close to sentiment distribution, another loss function is defined to minimize the *cosine distance* or $(1 - Cosine\_Similarity)$ between attention distribution and sentiment distribution, as follows:

$$loss^2 = 1 - \frac{\sum_{i=1}^{T} \alpha_i \alpha_i^*}{\sqrt{\sum_{i=1}^{T} \alpha_i^2} \sqrt{\sum_{i=1}^{T} (\alpha^*)_i^2}} \tag{4.10}$$

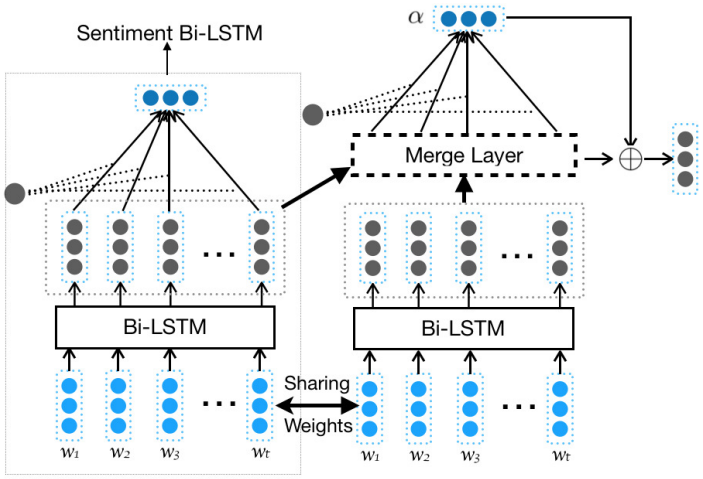


Figure 4.3: Sentiment transferred model (STBi-LSTM). The model has two training steps. 1. The sentiment Bi-LSTM is firstly trained on a sentiment corpus. 2. Two Bi-LSTMs are trained together on an irony corpus, but with the weight of sentiment Bi-LSTM frozen.

$$loss = loss^1 + \lambda * loss^2 \tag{4.11}$$

where λ is a hyper-parameter to adjust $loss^2$ when updating neural network.

4.4.3 Sentiment Transferred Bi-LSTM (STBi-LSTM)

The previous two proposed models transfer sentiment hard attention. Our third proposed method (illustrated in Figure 4.3) is designed to transfer deep features from sentiment analysis into irony detection for learning both explicit and implicit context incongruity. Our model consists of two Bi-LSTMs. One of these Bi-LSTM acts as the sentiment feature extractor, while the other is the irony detector. The training process comprises two parts. First, the sentiment Bi-LSTM is trained on a readily available Twitter sentiment corpus, and then the weights of the Bi-LSTM are kept frozen. In the second part of the training, a Tweet is given to both Bi-LSTMs. The sentiment Bi-LSTM (or the sentiment feature extractor) outputs deep features that are about words, with the implicit and explicit sentiments. The second model first learns semantic features for the context; both features are then mapped into a common latent feature space at the merge layer, and features on incongruous context are learned by the attention layer and the fully connected layer of the second Bi-LSTM. In terms of the

Table 4.1: Datasets of irony/sarcasm detection.

	Ironic vs. Non-ironic	Annotation
Reyes13	10,000 vs. 10,000	#irony vs. #education, #humor, or #politics
Barbieri14	10,000 vs. 10,000	#irony, #sarcasm vs. #eduation, #humor, etc.
Ptacek2014	18889 vs. 48890	#sarcasm for sarcastic Tweets
SemEval2018_Hashtag	2826 vs. 1792	#sarcasm, #irony, #not
Riloff2013	474 vs. 1608	manual
Moh2015	532 vs. 1397	manual
SemEval2018	2,222 vs. 2,396	manual

mathematical operation of incorporating transferred deep features at the merge layer, deep features from the sentiment Bi-LSTM and the second Bi-LSTM are concatenated before the attention mechanism:

$$H_{merged} = [H_{semantic} \parallel H_{sentiment}] \tag{4.12}$$

4.5 Experiments

We next discuss the experiment setup, including baselines and datasets, and then report results. We also report the results of error analysis for our models.

4.5.1 Baselines and Datasets

We compared our models against deep learning-based irony detection models as well as representative conventional feature-based models.

- Bi-LSTM: Attention-based Bi-LSTM structure has been employed for irony detection in conversations and for learning representations for irony detection in the literature [19, 20]. We implemented the network for our task and our implementation is based on the popular structure in [17].

- CNN-LSTM: Our implementation closely followed the architecture in [173]. It has three different neural network layers, a convolutional layer followed by two LSTM layers, and a fully connected layer with the same hyper-parameter settings.

- LSTM: The model proposed in [21] is an attention-based LSTM for irony detection on Twitter.
- CNN: The convolutional network [9] is widely used for classification problems.
- [16], [15] and [171] propose classical feature-based irony detection models. Those in [16] and [15] are representative models focused on context incongruity.

Several datasets are widely used in the irony detection literature. There are two approaches to annotating sarcasm. Some datasets are automatically annotated using the sarcasm hashtags #irony, #sarcasm, and #not. Other datasets are manually annotated (i.e., by humans).

- Reyes2013 [163], Barbieri2014 [164], and Ptacek2014 [165] are datasets automatically annotated by hashtags, as shown in Table 4.1. Each pair of sarcasm and non-sarcasm class of Tweets forms a dataset for evaluating irony detection.
- Riloff2013 [16], Moh2015 [166], and SemEval2018 [167] are manually annotated Twitter datasets. SemEval2018 is the official dataset used for SemEval 2018 Task 3 (irony detection in English Tweets). Statistics of the datasets are shown in Table 4.1.
- For an interesting comparison between hashtag-labeled datasets and manually labeled datasets, we used hashtags (i.e., #sarcasm and #irony), to automatically label SemEval2018, which result in an additional manually labeled dataset; this allows us to compare our models on a dataset with different annotation strategies. The details of the dataset are shown in Table 4.1.

We randomly split each dataset into 80% for training and 20% for testing, except SemEval2018, for which we used official training and testing splits. The parameters were tuned on a random 10% portion of the training data. For a fair comparison, following the literature, macro average F_1 was used as the evaluation metric, except for SemEval2018 where the binary F_1 was adopted.

Word-level tokenization uses the tokenizer from spaCy[1]. The word embeddings for all models were initialized with the pre-trained fastText [215] word vectors with 300 dimensions.

[1] https://spacy.io

The word-level sentiment scores were generated by NLTK with the help of a sentiment analysis tool VADER [212], which is designed for sentiment analysis of social media data, especially Twitter. Another great advantage of VADER is that it not only provides the polarity of words, but also gives the sentiment strength of words. Also, we adopted a sentiment emoji corpus [205]. The sentiment corpus for transfer learning used in our STBi-LSTM is based on two sentiment corpora used in SemEval 2017 Task 4 [213] and SemEval2015 Task 11 [214]. The hyper-parameters are selected using a grid search. The best dimension of hidden states for all variants of Bi-LSTMs in our grid search is 200.

4.5.2 Evaluation

Table 4.2: Results (F_1) for irony detection on hastag-annotated datasets.

	Reyes2013			Barbieri2014			
	edu	*hum*	*pol*	*edu*	*hum*	*pol*	*news*
Bi-LSTM	94.01	95.54	96.32	94.12	94.86	98.62	96.24
CNN-LSTM	92.04	92.73	93.33	93.15	94.78	97.48	96.10
LSTM	92.30	89.50	89.00	94.03	94.23	97.56	96.11
CNN	93.35	93.44	94.66	94.12	95.53	98.31	96.28
AABi-LSTM	94.23	95.56	96.42	94.92	95.73	98.18	96.91
SABi-LSTM	94.65	**95.82**	96.15	94.21	95.16	98.34	96.41
STBi-LSTM	**94.69**	95.69	**96.55**	**94.95**	**96.14**	**98.62**	**96.92**
Farias et al. [171]	90.00	90.00	92.00	90.00	92.00	94.00	96.00

Table 4.3: Results (F_1) for irony detection on manually annotated datasets.

	Riloff2013	Moh2015	SemEval2018	SemEval2018 Hashtag-Labeled
Bi-LSTM	73.57	58.31	64.15	69.08 / 72.18
CNN-LSTM	70.56	59.22	61.16	67.21 / 70.52
LSTM	72.17	57.16	63.66	67.04 / 72.47
CNN	74.75	57.71	62.03	70.49 / 72.80
AABi-LSTM	75.39	61.54	67.86	71.85 / 73.28
SABi-LSTM	74.63	63.37	65.33	70.25 / 73.05
STBi-LSTM	**77.85**	**66.80**	69.00	**72.11** / **73.56**
Reported Best Result	73.00 [171]	66.00 [171]	**70.54** [167]	-

In order to clearly show how models perform on datasets using different annotation strategies, we report experiment results in two separate tables. Table 4.2 reports the experimental results on datasets labeled using the hashtags #irony and #sarcasm. Table 4.3 reports the experiment results on datasets annotated by humans on crowdsourcing platforms. Results for the hashtag-labeled SemEval2018 dataset are also included in Table 4.3 for easy comparison with results for the manually labeled SemEval2018. Additionally, we report the results of our models on class-unbalanced datasets in Table 4.4.

From Table 4.2, it is obvious that irony detection on hashtag-annotated datasets is not as difficult as detection on manually labeled datasets. Most models have achieved very promising results almost across all hashtag-labeled datasets. Moreover, our results are consistent with a previous study [15], where experiments show that models in their study have better performance on hashtag-labeled datasets than on manually annotated datasets. Additionally, neural models perform better than traditional machine learning models, such as feature engineering with SVM [171].

Table 4.3 presents experiment results on human annotated datasets. In general, attention-based models work better than other models, including other neural models and conventional machine learning methods. On the dataset Riloff2013, the proposed three models all achieved better results than the baselines; in particular, our third proposed model achieved the best result, with approximately 4% improvement over Bi-LSTM.

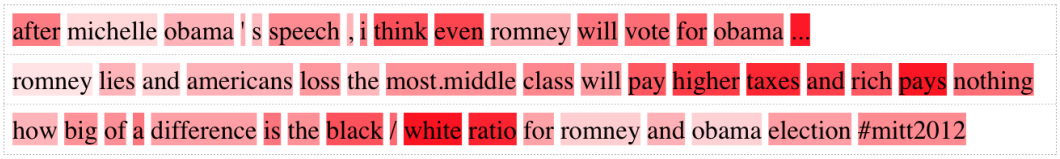


Figure 4.4: Examples of attention distribution on ironic Tweets from Moh2015. Tweets in red are incorrectly classified. Attention is generated by our model STBi-LSTM.

In our experiments, the dataset Moh2015 is the most difficult, not only because of the relatively small size of the dataset (1,397 non-ironic Tweets and 532 ironic Tweets), but also because of the type of irony expressed in this dataset. This dataset was collected using hashtags related to the "2012 US presidential election," therefore most of the ironic Tweets are either

situational irony or ironic utterances related to named entities or external knowledge.

For example, in order to detect the irony expressed by the Tweets in Figure 4.4, a model has to have knowledge of the named entities, for examples, "Obama" and "Romney," or background knowledge of the tasked event "2012 US presidential election." However, our model (STBi-LSTM) is still able to capture most of the words or phrases with sentiment meaning. The dataset used by SemEval 2018 Task 3 "irony detection in English Tweets" is another difficult task. Our proposed models achieve better results than other baseline neural models, and our results are also comparable to the best in the official rank [167].

In terms of statistical significance, Wilcoxon signed-rank tests [197] indicate significantly higher accuracy in our model STBi-LSTM (Mdn = 95.31) than in LSTM (Mdn = 91.77), $Z = 3.67$, $p < 0.001$, $r = 0.98$; CNN (Mdn = 93.78), $Z = 2.83$, $p < 0.002$, $r = 0.76$; CNN-LSTM (Mdn = 92.94), $Z = 3.67$, $p < 0.001$, $r = 0.98$; Bi-LSTM (Mdn = 94.49), $Z = 2.94$, $p < 0.002$, $r = 0.78$.

As discussed above, results show that all of our models achieved better performance on hashtag-labeled datasets than on manually labeled datasets. There are several reasons for this. First of all, the size of the dataset can affect the performance of neural models. The hashtag-labeled datasets have a greater number of Tweets than manually labeled datasets: for example, Reyes2013 (Irony vs. Education) has about 20,000 Tweets which is ten times the number of Tweets in Moh2015. Deep learning models give better results on hashtag-based datasets than on manually-tagged ones because they are bigger, and deep learning models succeed in generalizing if data is big enough. The labeling approach used for manually annotated datasets is another factor that influences the performance of deep learning models. For example, in Reyes2013 datasets, the ironic datasets were annotated by the appearance of #irony while the non-ironic datasets were labeled by the appearance of #education, #humor, or #politics. In contrast, non-ironic Tweets in manually annotated datasets are most likely to be general Tweets (SemEval2018, Riloff2013) or Tweets related to a particular event-related (Moh2015). Lastly but most importantly, we find that all attention-based models tend to put high attention on a list of common words. For example, in Figure 4.5, words like "irony" and "sarcasm" in Reyes2013 are highly attended by attention-based models. It must be noted that ironic Tweets

you said , ; maybe being honest about their feelings ? ; i laugh at the irony .
and yes , i 'm totally aware of the irony considering a recent tweet ...
parking lot of the liscense office.oh sweet irony .
parenting so how many of you folks out there read other people 's tweets ? anyone ? sarcasm

Figure 4.5: Examples of hashtag-labeled ironic Tweets in Reyes2013.

Table 4.4: Results (F_1) for irony detection on class-unbalanced datasets.

	Reyes_Unbalanced (10,000 vs. 30,000)	Barbieri_Unbalanced (10,000 vs 40,000)	Ptacek2014 (18,889 vs. 48,890)
Bi-LSTM	92.15	95.22	98.12
CNN-LSTM	91.64	95.06	98.00
LSTM	91.23	94.59	98.21
CNN	92.06	95.14	98.62
AABi-LSTM	91.70	95.30	98.83
SABi-LSTM	92.15	95.21	98.52
STBi-LSTM	**92.25**	**95.66**	**99.05**

were automatically labeled by the hashtag #irony, which was removed from ironic Tweets. In these Tweets, the word "irony" acts like a topic word or an important part of the context. Removing the word "irony" is more likely to make a context nonsensical or incomplete. We performed further analysis on ironic Tweets in Reyes2013 and found that the frequency of the word "irony" is about 3,531 in 10,000 ironic Tweets. Moreover, for non-ironic Tweets, there are some common topic words, for example, "education", "technology", "science" and "edchat" in Reyes2013 (Irony vs. Education). All of the deep learning models achieved very good results on hashtag-labeled datasets by learning patterns based on common topic words appearing in context, such as "irony" and "technology." However, all attention-based models failed to detect context incongruity since all put attentions mainly on common topic words. The performance improvement of our models over Bi-LSTM on hashtag-labeled datasets is not as pronounced as for the manually-labeled datasets; this may be explained by the fact that attention-based models are strongly affected by the common topic words in the datasets instead of by context incongruity.

For the last column in Table 4.3 ("SemEval2018 Hashtag-labeled"), there are two results for each model. Results on the left side are produced from experiments using the hashtag-labeled

oh dear, some people don't get 😳
day ninth december to have a pint at the boars head| really still in bed 😊
take a look at those calves!!! 😎😎😂😂 ladies love the calves! #about #the #arms #its #about...
i hate waking up in the mornings 😏#basicbrianna #an #early #bird
i have a very annoying bluetick for sale. 8 yrs old. needs babied and a bath.

Figure 4.6: Examples of hashtag-labeled ironic Tweets, which were annotated as non-ironic by humans. Labelling hashtags #sarcasm, #irony, and #not are not included.

training data but the manually annotated testing data, while results on the right side are produced by experiments using both the hashtag-labeled training and testing datasets. Comparing the columns "SemEval2018" and "SemEval2018 Hashtag-labeled," all models have better performance using hashtag-labeled training datasets than using manually labeled datasets. Most interestingly, using hashtag-labeled SemEval2018 training data improves the performance of all neural models on manually labeled SemEval2018 testing data, which demonstrates that the hashtag labels contain supervision signals for irony detection. On the other hand, all models have the best performance for hashtag-labeled training and test data, which demonstrates that hashtags define some aspects of sarcasm that can be more easily identified. Indeed, we analyzed some randomly selected Tweets labeled as ironic by hashtags but labeled as non-ironic by humans and found that most of them have obvious sentiment patterns. Examples of such Tweets are shown in Figure 4.6.

Table 4.4 shows the performance of our models on class-unbalanced datasets. Generally, the proposed model STBi-LSTM works the best of all of the models. However, the performance difference among the models is very small. In terms of performance difference between the class-balanced dataset and the class-unbalanced dataset, the results for all models are worse on unbalanced datasets than on balanced datasets by around 2.21 (Reyes2013) and 0.83 (Barbieri2014).

4.5.3 Discussion

We first discuss how our models improve the attention mechanism for detecting contexts for sentiment contrast. We further discuss how our third proposed model (STBi-LSTM) learns the contexts for explicit and implicit incongruity.

Figure 4.7: Differences of attention distribution among attention-based models.

Figure 4.7 presents an ironic Tweet for which the learned attention differs between our models and Bi-LSTM. Generally, Bi-LSTM is able to detect some of the words with sentiment meaning, but it often fails to detect the full context for incongruity. For example, in Figure 4.7 it spreads very high attention across the phrase "love waking up," expressing the positive sentiment, but places very small attention on the phrase "worst fcking mood," which is the negative situation of this ironic Tweet. In contrast, all our proposed models have more attention on "worst fcking mood." Most importantly, without using an explicit lexicon, our third proposed model, STBi-LSTM, is still able to detect both parts of context incongruity and spread balanced attention on both.

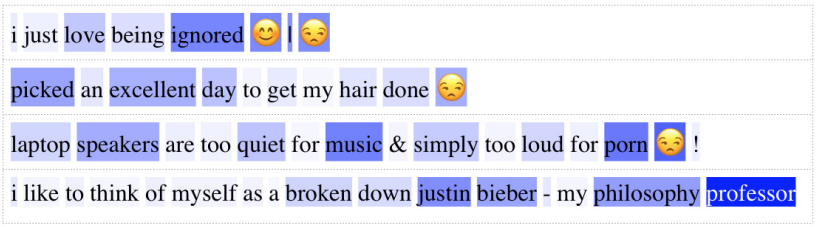


Figure 4.8: Examples of attention distribution learned by STBi-LSTM.

With the transferred deep features from the sentiment model, the STBi-LSTM performs very well, especially on detecting sentiment-based context incongruity. Figure 4.8 shows several selected examples of attention learned by STBi-LSTM. In this figure, the first two Tweets are examples of explicit context incongruity: "love" versus "ignored" is the key sentiment contrast in the first example, while "excellent day" versus sad face emoji is the sentiment contrast in the second example. In order to show the ability of our models on detecting implicit context

incongruity, we picked two example Tweets from the dataset. In Figure 4.8, the third Tweet is ironic about the sound of laptop speaker, and the irony is expressed by contrasting two situations, which are "quiet for music" and "loud for porn". Each of these two phrases does not have the explicit sentiment until it is in a contrasting context, and our model has successfully identified most key patterns for building the implicit context incongruity. The last example has two named entities as context incongruity; neither has real sentiment meaning until our model passes the learned sentiment knowledge to them. Both "Justin Bieber" and "philosophy professor" appear a few times in our sentiment training corpus, and Tweets containing "Justin Bieber" are more likely to be negative while Tweets containing "philosophy professor" are more likely to be positive. Even though neither named entity has sentiment meaning in general, the supervised sentiment training can embed an implicit sentiment via deep features. With the implicit sentiment features learned in sentiment training, our STBi-LSTM successfully detects context incongruity at the second stage of learning.

literally functioning on 4 hours of sleep and i feel great 😄 ! !	Literally functioning on 4 hours of sleep and I feel great 😄 !! #Not
i wonder what triggered the anxiety ?	I wonder what triggered the anxiety? #sarcasm
thanks i thought it was tomorrow	thanks I thought it was tomorrow #not #iknow #notthepoint
how can u miss something u never had ? ! ! #miss	How can u miss something u never had?‖#randomthoughts #miss #irony http://t.co/G5jLy9IKqn

Figure 4.9: Examples of mistakenly classified Tweets (ironic Tweets have been classified as non-ironic): Tweets in the left column are chosen from the SemEval2018 test data, Tweets in the right column are their original version where all hashtags are kept. (The intensity of red represents the attention value of each word paid by our STBi-LSTM model).

4.5.4 Error Analysis

Our models can only detect irony based on the self-contained contents of Tweets. In order to understand what our models miss in irony detection, we provide several mistakenly classified examples (with their original text content) in Figure 4.9. In ironic Tweets, hashtags such as #irony, #sarcasm, and #not are often used to indicate the irony intention. When these hashtags are removed for learning a more general model, it is hard to imagine that even humans would be able to classify such Tweets as ironic. For example, neither the second nor the third

examples carries the irony sense when the hashtags #sarcasm and #not are removed.

Some irony can only be inferred from the conversational context. As a result, when the complete conversational context is not available, it is hard even for a human to find the irony [16, 19]. In Figure 4.9, the second and third Tweets probably originate from conversational contexts in which their authors wrote them to express ironic intent. In the first example, our model successfully detected the positive sentiment "feel great," but failed to detect the negative situation "4 hours of sleep."

4.6 Summary

In this chapter, we have studied the problem of irony detection on Twitter. Context incongruity is a commonly seen form of irony on Twitter, where the contrast between a positive statement and a negative context is the common form of context incongruity. We have proposed employing transfer learning and attention-based neural networks to identify context incongruity for detecting irony. The most challenging part for training a good automatic irony detection model is the limited availability of human labeled dataset. In contrast to irony detection, sentiment analysis has sufficient resources, including pre-defined sentiment lexicons and human-annotated corpora. We have proposed our models to take advantage of these widely and readily available sentiment resources to improve the ability of attention-based model to detect context incongruity. By incorporating transferred sentiment, our models are able to detect both implicit and explicit context incongruity in most cases. Experiments show that our three proposed sentiment attention mechanisms result in better performance than the baselines, including several popular neural models for irony detection on Twitter.

Chapter 5

Discovering Relevant Reviews to Answer Product Questions

5.1 Introduction

Online customers, like offline shoppers, often have questions prior to a purchase, and it is infeasible for sellers to provide answers to every customer query in real time due to the high volume of online traffic. As product queries tend to be very product-specific, finding similar queries from other products and retrieving their answers is unlikely to be useful. An alternative solution might be to rely on reviews, as they can provide insights to the query.

To illustrate how reviews might be useful, two product queries, gold answers and top-ranked (by our model) review sentences are shown in Table 5.1. We can see that the review sentences are relevant and helpful; in the second case they provide more elaboration than the "Yes" gold answer.

There are several studies that leverage reviews to tackle product question answering (PQA). A mixture-of-experts (MoE) model [22, 216] is trained to distinguish real answers from non-answers based on reviews. In a similar vein, generative models [217, 218] are proposed to extract useful information from reviews to generate answers. The studies show some promising results, suggesting reviews are a useful resource for answering product queries. Success in this task has the implication of relieving the community from manually answering the growing

Table 5.1: Answer and top-ranked review for two queries on Amazon.

Q1: If you take this as directed twice a day how long does this size bottle last?	
	**Answer:** About a month. I skip some doses but my doctor retested and my iron levels increased after 3 month.
	Review: Following the directions of taking it twice a day, it may last a whole month.
Q2: Does this have a volume control?	
	**Answer:** Yes
	Review: The remote on the cord is really easy to use, top button is volume up, bottom is volume down, middle is pause/play/answer.

number of product queries. It also shortens the length of time customers wait for a response and improves the online shopping experience.

With that said, it is difficult to directly train a supervised system to find relevant reviews for product queries, as there is a lack of ground truth, i.e., there is no direct association between product queries and reviews. Creating an annotated dataset (by marking the relevance of reviews for each query) would inevitably be a colossal and expensive effort, given the large number of products and reviews.

To tackle this, a better approach is to use existing question and answer pairs as supervision signals. By framing answer prediction as the objective, we can then decompose the problem to two sub-tasks, by learning how to match: (1) a query with a review; and (2) a review with an answer. Post-training, the learned model for the first sub-task can then be used to extract potentially useful reviews for a query. This idea is first proposed by Mcauley et al. [22].

A key challenge in our task is the language mismatch between questions and reviews. McAuley and Yang [22] proposed a number of features and a word-based relevance function to align questions with reviews and reviews with answers. In contrast, we propose neural models that require little feature engineering, and as demonstrated in our experiments, tackle the language mismatch between questions and reviews better. To facilitate replication and future research, we release the source code of our models.[1]

To summarise, our main contributions in this chapter are:

[1] https://github.com/zswvivi/icdm_pqa

- We propose two neural models to predict answers for product queries using reviews, with the end goal of using the model to find relevant reviews given a query.
- Our end-to-end neural models use raw texts as input, and as such do not require feature engineering.
- We demonstrate that our best model outperforms current benchmarks in: (1) predicting answers for a query; and (2) finding the reviews most relevant to a query.

5.2 Related Work

Existing studies on PQA using reviews can be broadly divided into extractive approaches ([22, 216, 219, 220]) and abstractive approaches ([217, 218]). Extractive approaches extract snippets from relevant reviews to answer questions while abstractive methods generate answers by drawing from the vocabulary. With both approaches, a crucial first step is to find reviews relevant to a question.

The closest work to ours is the system Mixtures of Opinions for Question Answering (MOQA), proposed in Mcauley et al. [22]. MOQA is trained on a large corpus of previously answered questions to learn a relevance function that can surface relevant reviews, where 'relevance' is measured by how well a review helps identify the correct answer. MOQA relies on using traditional word-based relevance function and manual feature engineering to rank reviews. The lexical mismatch problem between the QA domain and reviews brings significant challenges to MOQA and can lead to low recall for answering questions. Previously proposed neural models for PQA employ Autoencoder [219], RNN [217], and convolutional network [218], but they fail to address the severe language mismatch issue between the QA domain and the review domain at the lexical level and beyond.

Other related studies focus on different aspects of the task. Wan et al. [216] study the opinion divergence problem where a question has multiple answers that are opinionated and divergent. Yu et al. [220] propose a QA system that targets yes-no binary questions. Xu et al. [221] address product compatibility related questions.

Related research in the domain of NLP includes open-domain QA [222, 223], reading comprehension [224, 225] and community QA [183, 226]. Research in these areas focuses on finding answers in documents within one domain rather than from across a different domain. Open-domain QA [222, 223] normally uses existing question and answer pairs but with crawled context as the source of answers. Reading comprehension [224, 225, 227] is similar to open-domain QA, but question-answer pairs are constructed based on a given text source. Community question answering (cQA) [183, 228, 229] is different from open-domain QA and reading comprehension; in cQA given a question and a list of texts, the task is to rank the texts according to their relevance to that question. PQA using reviews is close to cQA in terms of finding the most relevant texts from a pool of texts. However, cQA normally has ground truth, where the most relevant text has been labeled, which provides a strong supervised signal. For PQA using reviews, the most relevant review to a given question is not identified; this is the most challenging part of this task.

In terms of learning a relevance function, all PQA models including ours are quite similar, where we use existing QA pairs as the weak supervision. The main difference between our proposed model and existing models is the relevance function. We use a transformer-based language model as the relevance function that is good at learning word-level dependencies. This is especially helpful in PQA because reviews and questions often have very different vocabularies. Word-level dependencies can help address the language mismatch problem.

5.3 Methodology

Given a product query, our goal is to surface relevant reviews that can potentially provide answers. Due to the lack of annotated query and review pairs—i.e., we do not have data where relevant and irrelevant reviews for a query are marked—we cannot directly train a classification/ranking model to label/rank reviews for a given query.

We do, however, have an abundance of labeled answers for queries (e.g., via mining existing queries that have been answered by users). In order to learn the relevance between review and query, we leverage the supervision signal of community query and answer pairs by building a model whose objective is to compute the probability of an answer given a query *via reviews*.

Formally:

$$P(a|q) = \sum^{r} P(r|q)P(a|r,q)$$

where a, q and r denote answer, query and review respectively.

In other words, we are decomposing the relevance between answer and query ($P(a|q)$) into two functions, by computing the relevance between: (1) review and query ($P(r|q)$); and (2) answer and review ($P(a|r,q)$). This formulation implicitly assumes that product reviews are useful when predicting the most relevant answer given a query; we contend that this is a reasonable assumption, and find empirically that the learnt relevance function for review and query extracts useful reviews based on user studies.

In practice, instead of building ranking models that compute the full probability distribution for a set of answers, we treat this as a pairwise classification problem whereby the goal of the model is to score an answer for a given query, and the model is optimized by learning to score a true answer higher than a randomly selected non-answer.

This framework of using reviews to score answers is originally proposed in Mcauley et al. [22], it is inspired by the MoE classifier [230], which uses a number of weak classifiers or *experts*, each weighted by its confidence, to make a prediction. By seeing each review r as an *expert*, the term $P(r|q)$ can be interpreted as an expert's confidence, and the term $P(a|r,q)$ as its prediction. Mcauley et al. [22] call their model MOQA (Mixtures of Opinions for Question Answering), and it serves as our baseline system for comparison.

MOQA parameterizes the relevance functions using simple pairwise similarity metrics (such as BM25+, ROUGE-L) and bilinear models and trains using classical learning models. MOQA originally proposes two models with different objectives for tackling binary yes/no questions and open-ended questions. In this chapter, we focus only on the latter, as they are arguably the more interesting questions. For these open-ended questions, MOQA is optimized by training on the logistic loss of the probability of scoring the true answer higher than a randomly selected non-answer. We propose to implement a neural extension of their framework, by exploring by both simple networks and modern Transformer-based architectures [23]. Our neural architecture affords greater flexibility in terms of how we want to model the relationship between query/review/answer; it can also be easily extended to incorporate additional metadata

on questions or reviews as a future direction.

5.3.1 Neural Extension

In our neural models, we parameterize the relevance functions $P(r|q)$ and $P(a|r,q)$ with neural networks. We explore both simple networks and Transformer-based networks [23] for these relevance functions, with each encoding different assumptions about how it models the relationship.

As with MOQA, rather than computing the probability distribution over an answer set ($P(a|q)$), we compute a bounded $(0-1)$ score for an answer ($S(a|q)$), and optimize based on a margin loss:

$$\min(0, \delta - S(a|q) + S(a'|q))$$

where a is a real answer, a' is a randomly selected non-answer, and δ is the margin hyper-parameter.

Intuitively, the model is trained to score a real answer differently from a non-answer with at least a difference of δ (or a penalty will be incurred).

5.3.2 NNQA

NNQA uses fastText [231] as a sentence encoder to create a vector representation h_q, h_a and h_r for q, a and r, respectively, by taking an average of the pre-trained word embeddings in the sentence. Given the sentence encodings, we model the relevance functions with a bilinear function:

$$S(r|q) = \sigma(h_q \mathbf{W_1} h_r^\intercal)$$
$$S(a|r) = \sigma(h_r \mathbf{W_2} h_a^\intercal)$$

where $\mathbf{W}_*$ are model parameters.

We score an answer by combining them:

$$S(a|q) = \sum_{r \in \mathcal{R}} S(a|r) S(r|q)$$

where $\mathcal{R}$ is a set of reviews for q.

Note that in NNQA, we relax the MoE assumption where the accumulated confidence sums to 1 (i.e., $\sum_{r \in \mathcal{R}} S(r|q) = 1.0$). In preliminary experiments we did incorporate this constraint by applying a softmax operation over the confidence scores, but found that this formulation of applying sigmoid to bound confidence scores works better empirically.

5.3.3 BERTQA

Our second neural model uses BERT [24], a state-of-the-art Transformer-based model that has shown to perform competitively over a range of NLP tasks, from question-answering to paraphrase detection to natural language inference. BERT is pre-trained over a large corpus, and it can be further fine-tuned to subsequent tasks of interest by adding additional task-specific layers.

Our BERT model (henceforth BERTQA) shares the same framework as NNQA; the core difference is that the relevance functions ($S(r|q)$ and $S(a|r)$) are computed directly with the BERT encoder:[2]

$$S(r|q) = \text{softmax}(W_1^\intercal \text{BERT}(r, q))$$
$$S(a|r) = \sigma(W_2^\intercal \text{BERT}(a, r))$$

An important advantage of BERTQA is that it allows for a more fine-grained analysis between two pairs of texts, as: (1) its self-attention mechanism provides a means to assess token-level similarity for all tokens between the pairs; and (2) it uses at sub-word units (i.e., word pieces). For NNQA, as we first generate a compressed representation of the texts (h_r, h_q and h_a) before computing their relevance, we lose the token-level analysis that BERTQA has.

Also, unlike NNQA, for BERTQA we found that empirically it is beneficial to keep the MoE constraint (i.e., $\sum_{r \in \mathcal{R}} S(r|q) = 1.0$), and so we apply a softmax operation for $S(r|q)$ over all reviews in this model.

[2]Note that BERT produces a hidden state for each word in the sentences; we use only the hidden state of the CLS token, which is special token prepended at the start of the sentence pair. Also, the BERT encoder is shared for (r, q) and (a, r), and it is updated during fine-tuning.

Table 5.2: PQA data statistics.

Category	#questions	#products	#reviews	#r/p
Home and Kitchen	184,439	24,501	2,012,777	20
Sports and Outdoors	146,891	19,332	1,013,196	19
Automotive	89,923	12,536	395,872	10
Cell Phones	85,865	10,407	1,534,094	28
Health and Personal Care	80,496	10,860	1,162,587	26
Tools and Home Improv.	101,088	13,397	752,947	18
Patio Lawn and Garden	59,595	7,986	451,473	19

5.3.4 Review Filtering

In all models (our neural models and MOQA), reviews are broken into individual sentences, and a "review" (r) is a review sentence rather than the full review. For this reason, a product typically has a large number of reviews (i.e., review sentences; see statistics in Table 5.2), and incurs a significant computational overhead when we consider all reviews for each query.

To this end, we explore several methods to pre-rank the reviews for each query, with the idea that most reviews are irrelevant and so can be removed before we feed them to the neural models. We experimented with TF-IDF-based unsupervised and machine learning-based supervised models, and found that a simple BERT classifier works best.

Our BERT filter (henceforth F) works by first fine-tuning a pre-trained BERT on (query, answer) pairs to predict whether an answer is a real answer for the query.[3] When training our neural models (NNQA and BERTQA), we can optionally apply F to (query, review) pairs to pre-rank the reviews and consider only the top-N reviews. Note that there is a domain mismatch between training and test inference for F, since it is trained on (query, answer) pairs but used for (query, review); however, we find that it works sufficiently well as a preliminary filtering system.

[3]Negative training examples are generated by sampling random answers from other queries.

5.3.5 Cross-Domain Pre-training

The dataset we use has a number of distinct categories. When training the neural models, we treat each product category as an independent domain and train a number of separate models. Observing that questions across categories occasionally share similar questions, e.g., queries about compatibility or size, we explore pre-training a general model that combines data from all categories, and then fine-tuning it further for each of the domain. We hypothesize that this training procedure should help tackle both domain-invariant queries (via pre-training) and domain-specific queries (via fine-tuning). We denote models that use cross-domain pre-training with the label P, e.g., NNQA+P means the NNQA model is first pre-trained with all data and then fine-tuned for each domain.

5.4 Experiments

5.4.1 Data

We use the Amazon QA and review dataset developed by Mcauley et al. [22] for our experiments. We experiment with seven product categories; statistics for these categories are presented in Table 5.2.

Following MOQA, we split the data in the same train/development/test ratio,[4] and break reviews into sentences. That is, a *review* is in practice a *review sentence*, rather than the full review. The rationale for using review sentences is that from an application perspective, displaying a relevant sentence for a query is more user-friendly than presenting a lengthy full review.

5.4.2 Training Details

We tune our neural models based on the development partition. For MOQA, we use the open source implementation and its default optimal configuration.[5] When applying F, we keep only

[4]Note that we are unable to produce the exact same splits used in the original MOQA publication, as the splitting algorithm provided by the authors uses a random seed. For this reason we re-run MOQA on our data for proper comparison.

[5]https://cseweb.ucsd.edu/~jmcauley/

Table 5.3: Answer prediction AUC performance of all models.

	MOQA	NNQA	NNQA+P	NNQA+F+P	BERTQA+F	BERTQA+F+P
Home.	0.8946	0.8015	0.8878	0.8518	0.9086	**0.9253**
Sports.	0.8766	0.8012	0.8871	0.8358	0.9097	**0.9272**
Automotive	0.8445	0.7942	0.8715	0.8015	0.8710	**0.8923**
Cell Phones	0.8746	0.7534	0.8737	0.8254	0.8591	**0.8774**
Health.	0.8772	0.8321	0.8952	0.8338	0.9076	**0.9263**
Tools.	0.8719	0.8080	0.8874	0.8375	0.8961	**0.9153**
Patio Lawn.	0.8301	0.7743	0.8752	0.8274	0.8943	**0.9141**
Average	0.8671	0.7950	0.8826	0.8305	0.8923	**0.9111**

the top ten reviews. For the neural models (NNQA and BERTQA), we set an upper limit of 100 for the total number of reviews (and discard the rest).

5.4.3 Quantitative Evaluation: Answer Prediction

We first present a quantitative assessment of our models. Due to the lack of annotated query and review data, we evaluate the models based on their accuracy in predicting the right answer for a given query.

We follow MOQA and use AUC as the evaluation metric:

$$\frac{1}{\mathcal{Q}} \sum_{q \in \mathcal{Q}} \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} S(a > a')$$

where a is a real answer, a' is randomly-sampled non-answer, $\mathcal{A}$ is the set of all non-answers,[6] and $\mathcal{Q}$ is the set of all queries.

Intuitively, AUC calculates for each query the proportion of cases where the model assigns a higher score to the real answer. A score of 1.0 indicates perfect accuracy. We present the full results in Table 5.3.

Looking at MOQA vs. NNQA, we see that MOQA is superior to NNQA in all categories, and about 7% better on average. Applying cross-domain pre-training (NNQA+P) improves the model substantially, with the largest gain in Automotive; on average NNQA+P now outperforms MOQA by 2%. However, when we apply F to filter away irrelevant reviews

[6]For every query, we randomly sampled 100 non-answers.

and keep only the top ten reviews (NNQA+F+P), we see a consistent drop in all categories, suggesting that the F procedure has inevitably discarded potentially useful candidates. We tried increasing the number of reviews that are kept (up to the top 50), but found that the performance was still consistently worse than NNQA+P.

Next we look at BERTQA. Due to memory constraints, we were unable to run BERTQA using all reviews; we therefore present only the results of BERTQA+F variants. Encouragingly, even with the filter, BERTQA+F has a superior performance over the baseline (MOQA) and all NNQA models, with an average performance improvement of over 3% over MOQA and 10% over NNQA. When we pre-train it over multiple domains (BERTQA+F+P), we see another substantial improvement, further widening the gap between BERTQA and other models. In addition, the p-value of Wilcoxon signed-rank test for our best model (BERTQA+F+P) against the baseline (MOQA) is 0.016 that is much less than 0.05. We hypothesize that the strong performance of BERTQA is likely to be attributed to its self-attention architecture, which facilitates a more fine-grained analysis of words between sentences.

5.4.4 Qualitative Survey: Relative Review Quality

We reiterate that our end goal is to extract helpful reviews that can provide answers for a product query. Although our quantitative evaluation (Section 5.4.3) demonstrates that BERTQA is a good model, its ability to discover useful reviews has yet to be directly assessed.

To this end, we use the trained relevance function ($S(r|q)$) of the models to select the most relevant review sentence for some queries, and ask users to judge the relevance of these reviews. We use the Amazon Mechanical Turk as the crowdsourcing platform.[7]

For the survey, we test three models: MOQA, NNQA+P, and BERTQA+F+P. MOQA serves as our baseline, and it has been previously shown to outperform an unsupervised retrieval baseline (Okapi-BM25+/ROUGE) [22]. NNQA+P and BERTQA+F+P are chosen because they are the best neural models (based on their answer prediction performance).

We experiment with three product categories: “Cell Phones” (the most popular category, with an average of 28 reviews per product); “Patio Lawn and Garden” (the least popular

[7] https://www.mturk.com/.

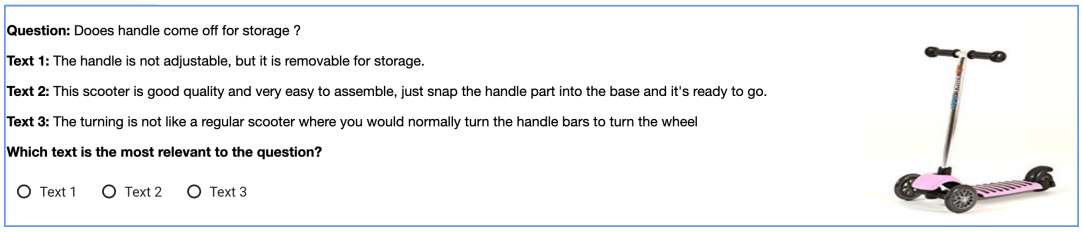


Figure 5.1: MOQA *vs.* NNQA+P *vs.* BERTQA+F+P. Extracted reviews are arranged in a random order.

category); and "Sports and Outdoors" (average in terms of popularity). For each category, we randomly select 100 questions, and for each question, we select the most relevant review as ranked by each of the three models. A crowdworker is presented with the product query, product image, and the three top-ranked reviews, and is asked to select the most relevant review based on the query. A screenshot of survey is presented in Figure 5.1.

We present three queries in an HIT, and one of the queries is a control question, where the three "reviews" consist of one real answer and two randomly selected non-answers. Each HIT is annotated by seven workers. The control question serves as a check to confirm that our workers are performing the task properly.[8]

We present the survey results in Table 5.4. For each category, we calculate the number of times each model is selected by a user. Across three categories, we see that BERTQA+F+P reviews are consistently selected by users as being most relevant. Surprisingly, MOQA performs better than NNQA+P in two out of three categories, showing that better performance in answer prediction does not necessarily translate to review extraction.

Considering that the candidate set for MOQA is the full set of reviews, while for NNQA+F and BERTQA+F+P it is up to 100 and 10 reviews, respectively, we expect that BERTQA+F+P might be handicapped. The user study, however, reveals that this is clearly not the case. In fact, our results suggest that relevant reviews can often be found among the top ten candidates, suggesting that pre-ranked reviews (by F) are good.

This then raises a natural question: if the top ten BERT filter reviews are already good, do we need re-rank them with BERTQA? To better understand this, we conduct a second survey

[8]We turn on the Masters qualification requirement for survey, and found that all workers answer the control questions correctly.

Table 5.4: Number of selected reviews for MOQA, NNQA+P and BERTQA+F+P.

	MOQA	NNQA+P	BERTQA+F+P
Sports.	24	32	**44**
Cell Phones	35	17	**48**
Patio Lawn.	31	30	**39**

Table 5.5: User study on comparison of the top one ranked review from F and BERTQA+F+P.

	F (BERT filter)	BERTQA+F+P
Sports and Outdoor	42	**58**
Cell Phones	36	**64**
Patio Lawn and Garden	48	**52**

to compare BERTQA+F+P vs. F, We follow the same procedure as before, by randomly selecting 100 questions from the three categories. For each question, we select the top ranked review by BERTQA+F+P and F, and ask workers to select the more relevant review.

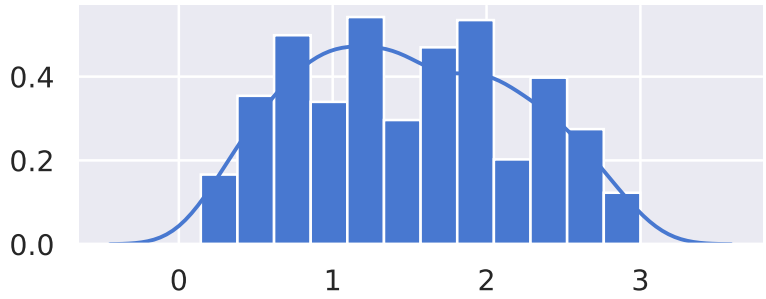
Table 5.5 shows the number of reviews selected by users for the two models. Encouragingly, BERTQA+F+P reviews are consistently selected as being more relevant, showing that the re-ranking step by BERTQA is important.

5.4.5 Qualitative Survey: Absolute Review Quality

Table 5.6: Average relevance scores for three reviews.

Q1: How many screen protector come with this package?  **Review:** Also, I was a little concerned that there was only 1 screen protector in the package. **Score:** 2.85
Q2: What're the dimensions of the largest planter?  **Review:** The planters are a nice size for plants. **Score:** 1.56
Q3: Will my att sim card work with this phone?  **Review:** This phone is CHEAP for a smartphone. **Score:** 0.29

Figure 5.2: Distribution of average relevance score. (Mean = 1.50, std. = 0.70)



In the previous section, we conducted user studies to compare a number of models on how well they find relevant reviews. Although BERTQA+F+P is shown to outperform the other models (relative performance), we are yet to measure the actual utility of its reviews (absolute performance). To this end we conduct a second user study and ask annotators to judge how helpful the reviews are in answering questions. Note that we test only the reviews selected by BERTQA+F+P here.

We randomly select 200 questions from the same three categories used previously. Annotators are asked to rate the helpfulness of a review on an ordinal scale from 0 to 3, where 0 indicates that the review does not answer the question and is not relevant, and 3 indicates that the review directly answers the question. The final relevance score for each review is the average rating given by seven annotators; we present some examples in Table 5.6.

The distribution of the relevance scores is displayed in Figure 5.2. Interestingly, the results are rather mixed; we can see that not all of the reviews are helpful, suggesting that there is still plenty of room for improvement.

5.4.6 Language Mismatch

The language used in queries tends to be very different from that used in reviews, and we hypothesize that the strength of BERTQA lies in its ability to align paraphrases without overlapping words (owing to its sub-word tokenization and deep self-attention layers). We calculate word-level Jaccard similarity between each question and the most relevant review as

Table 5.7: Top-ranked reviews by BERTQA+F+P. Boldface indicates phrases with similar meanings.

Question:	Will these gloves keep hands warm while **driving in winter**?
Review:	No, these aren't "warm" gloves, but they do still take the edge off **gripping an ice cold steering wheel**.
Question:	I want to use these pants for **surf fishing**. are they any good?
Review:	The pants fit well and keep their shape **in the water**.
Question:	How is the microphone. do you **sound clear to other callers**?
Review:	The headset is quite comfortable and is very **clear on both ends of the conversation**.

ranked by MOQA and BERTQA+F+P across all categories. The average Jaccard similarity for MOQA and BERTQA+F+P is 0.097 and 0.069 respectively. The significantly lower similarity value of BERTQA+F+P confirms our hypothesis: although BERTQA+F+P finds more relevant reviews, its reviews tend to have fewer words overlapping with questions. We present several examples of top-ranked reviews by BERTQA+F+P in Table 5.7. These examples demonstrate that the reviews selected by BERTQA+F+P uses different words/phrases to the questions, but that their meanings are similar.

5.5 Summary

Discovering helpful reviews to answer product-related queries is a challenging problem due to the language mismatch between queries and reviews, and the lack of annotated reviews. In this work, we adopt a MoE inspired framework, by leveraging reviews to predict answers for queries. We have presented two neural models, a simple model (NNQA) and a Transformer-based model (BERTQA). We have evaluated the models in terms of answer prediction and their ability to find relevant reviews, and shown that BERTQA produces the best performance in both tasks, demonstrating the importance of its architecture, which allows a more fine-grained analysis between two sources of text.

Chapter 6

Less is More: Rejecting Unreliable Reviews for Product Question Answering

6.1 Introduction

In the previous chapter, we showed that product reviews are a good source from which to extract helpful information as answers. The core idea of state-of-the-art PQA models is to take advantage of existing product reviews and find relevant reviews that answer questions automatically. In general, most PQA models implement a relevance function to rank existing reviews based on how they relate to a question. Some directly present a fixed number (typically ten) of the top-ranked reviews as answers for the question [22, 232], while others generate natural-language answers based on the relevant reviews [217, 218]. However, not every question can be answered by reviews: the existing set of reviews may not contain any relevant answers for the question, or a question may be poorly phrased and difficult to interpret and therefore requires additional clarification. Q2 in Table 6.1 is an example of an unanswerable question. The user wants to know whether the notebook comes with Dell's warranty, but none of the reviews mention anything about a warranty. In such a case, a system should abstain from

Table 6.1: Example of an answerable (Q1) and an unanswerable question (Q2). Green denotes high probability/confidence scores, and red otherwise.

Q1: What is the chain for on the side?			
Top three Ranked Reviews	Prob	Conf	Accept
 - This was driving me crazy but i see that another reviewer explained that grill has wire clip on chain to be used as extended match holder for igniting the gas if the spark mechanism fails to work or is worn out as sometimes happens with any gas grill.	0.99	0.82	✔
- PS Could not figure out the purpose of that little chain with the clip attached to the outside of the grill - even after reading entire manual.	0.95	0.54	✘
- It is to replace an old portable that I have been using for about 10 years.'	0.91	0.40	✘
Q2: Does this Dell Inspiron 14R i14RMT-7475s come with dell's warranty?			
Top three Ranked Reviews	Prob	Conf	Accept
 - I don't really recommend the PC for people who wants install heavy games programs.	0.74	0.48	✘
- The computer is nice, fast, light, ok.	0.12	0.01	✘
- I bought the computer for my daughter.	0.05	0.00	✘

returning any reviews and forward the question to the product seller. In the PQA literature, the issue of answerability is largely unexplored, and as such evaluation focuses on simple ranking performance without penalizing systems that return irrelevant reviews.

That said, the question answerability issue has begun to draw some attention in machine comprehension (MC). Traditionally, MC models assume the correct answer span always exists in the context passage for a given question. As such, these systems will give an incorrect (and often embarrassing) answer when the question is not answerable. This motivates the development of better comprehension systems that can distinguish between answerable and unanswerable questions. Since the second version of SQuAD, a popular MC dataset, was released [233], which contains approximately 50,000 unanswerable questions, various MC models have been proposed to detect question-answerability in addition to predicting answer span [30–33]. The

MC models are trained to first detect whether a question is answerable or unanswerable and then find answers to answerable questions. Sun et al. [34] propose a risk controlling framework to increase the reliability of MC models, where risks are quantified based on the extent of incorrect predictions, for both answerable and unanswerable questions. In contrast to MC, which always has one answer, PQA is a ranking problem in which there can be a number of relevant reviews/answers; PQA is more challenging than simple binary classification for answerability, and risk models designed for MC cannot be trivially adapted for PQA.

In this chapter, we focus on the problem of answer reliability for PQA. Answer reliability is the generalization of answerability. Answerability is a binary classification problem where answerable questions have only one answer (e.g., MC). In our problem setting, a product question can have a variable number of reliable answers (reviews), and questions with nil reliable answers are the unanswerable questions. The challenge is therefore how to estimate the reliability of answers. As our work shows, the naive approach of thresholding based on the predicted probability from PQA models is not effective for estimating the reliability of candidate answers.

We tackle answer reliability by introducing a novel application of conformal predictors as a rejection model. The rejection model [124] is a technique proposed to reduce the misclassification rate for risk-sensitive classification applications. The risk-sensitive prediction framework consists of two models: a classifier that outputs class probabilities, given an example, and a rejection model that measures the confidence of its prediction and rejects unconfident predictions. In our case, given a product question, the PQA model makes a prediction on the relevance for each review, and the rejection model judges the reliability for each prediction and returns only reviews that are relevant *and* reliable as answers. As an example, although the positive class probabilities given by the PQA model to the top three reviews are very high in Q1 (Table 6.1), the rejection model would reject the last two because their confidence scores are low. Similarly for Q2, even though the first review has high relevance, the rejection model will reject all reviews based on the confidence scores, and return an empty list to indicate that this is an unanswerable question.

For the PQA models, we explore both classical machine learning models [22] and BERT-

based neural models FLTR and BERTQA proposed in the previous chapter. For the rejection model, we use an inductive Mondrian conformal predictor (IMCP) [129, 130, 133, 234]. IMCP is a popular non-parametric conformal predictor used in a range of domains, from drug discovery [235] to chemical compound activity prediction [133]. The challenge of applying a rejection model to a PQA model is how we define the associated risks in PQA. Conventionally, a rejection model is adopted to reduce the misclassification rate, but for PQA we need to reduce the error of including irrelevant results. We propose to use IMCP to transform review relevance scores (or probabilities) into confidence scores and tune a threshold to minimize risk (defined by an alternative ranking metric that penalises inclusion of irrelevant results), and reject reviews whose confidence scores are lower than the threshold.

To evaluate the effectiveness of our framework in producing relevant and reliable answers, we conduct a crowdsourcing study using Mechanical Turk[1] to acquire the relevance of reviews for 200 product-specific questions on Amazon [22]. We find encouraging results, demonstrating the applicability of rejection models for PQA to handle unanswerable questions. To facilitate replication and future research, we release the source code and data used in our experiments.[2]

6.2 Related Work

In this section, we survey three topics related to our work: product question answering (PQA), question answerability and conformal predictors.

6.2.1 Product Question Answering

Existing studies on PQA using reviews can be broadly divided into extractive approaches [22] and generative approaches [217, 218]. For extractive approaches, relevant reviews or review snippets are extracted from reviews to answer questions, while for the generative approaches natural answers are further generated based on the review snippets. In both approaches, the critical step is to first identify relevant reviews that can answer a given question.

[1] https://www.mturk.com/.
[2] https://github.com/zswvivi/ecml_pqa

The key challenge in PQA is the lack of ground truth, i.e., there is limited data with annotated relevance scores between questions and reviews. Even with crowdsourcing, the annotation work is prohibitively expensive, as a product question may have a large number of reviews; this is even more of an issue when annotation is carried out at sentence level (i.e., annotating whether a sentence in a review is relevant to a query), which is typically the level of granularity with which PQA studies work [22]. For that reason, most methods adopt a distant supervision approach that uses existing question and answer pairs (QA pairs) from an external source (e.g., the community question answer platform) as supervision. The first of such studies is the Mixtures of Opinions for Question Answering (MOQA) model proposed by Mcauley et al. [22], which is inspired by the MoE classifier [230]. Using answer prediction as its objective, MOQA decomposes the task into learning two relationships: (1) relevance between a question and a review; and (2) relevance between a review and an answer, sidestepping the need for ground truth relevance between a question and a review. In the previous chapter, we extended MOQA by parameterizing the relevance scoring functions using BERT-based models [24] and found improved performance.

Another previous work [232] learns deep representations of words between existing questions and answers. To expand the keywords of a query, the query words are first mapped to their continuous representations and similar words in the latent space are included in the expanded keywords. To find the most relevant reviews, the authors use a standard keyword-based retrieval method with the expanded query, and found promising results.

6.2.2 Unanswerable Questions

There are few studies that tackle unanswerable questions in PQA. One exception is [236], where they develop a new PQA dataset with labelled unanswerable questions. That said, the author frame PQA as a classification problem, where the goal is to find an answer span in top review snippets (retrieved by a search engine), and as such the task is more closely related to machine comprehension.

In the space of MC, question answerability drew some attention when SQuAD 2.0 [233], which includes over 50,000 unanswerable questions created by crowdworkers, was released.

Several deep learning MC systems have since been proposed to tackle these unanswerable questions. [30] proposed a read-then-verify system with two auxiliary losses, where the system detects whether a question is answerable and then checks the validity of extracted answers. [31] proposed a multi-task learning model that consists of three components: answer span prediction, question answerability detection and answer verification.

More generally in QA, Web QA is an open-domain problem that leverages web resources to answer questions, e.g., TriviaQA [237] and SearchQA [222]. Sun et al. [34] introduce a risk control framework to manage the uncertainty of deep learning models in Web QA. The authors argue that there are two forms of risks, by returning: (1) wrong answers for answerable questions; and (2) any answers for unanswerable questions. In its overall idea, their work is similar to ours, although their approach uses a probing method that involves intermediate layer outputs from a neural model, which is not applicable to non-neural models such as MOQA.

6.2.3 Conformal Predictors

To measure the reliability of prediction for an unseen example, conformal predictors (CPs) compare how well the unseen example *conforms* to previously seen examples. Given an error probability ϵ, CP is guaranteed to produce a prediction region with probability $1-\epsilon$ of containing the true label y, thereby offering a means to control the error rate. CPs have been applied to many different areas, from drug discovery [131–133] to image and text classification [134]. [132] proposed a neural framework using Monte Carlo Dropout [139] and a CP to compute reliable errors in prediction to guide the selection of active molecules in retrospective virtual screen experiments. In [134], the authors replace the softmax layer with a CP to predict labels based on a weighted sum of training instances for image and text classification.

The problem of unanswerable questions is an important research field that has already attracted attention in domains other than PQA. Questions in product domain also may not have answers, and so we propose models to address unanswerable questions in PQA. In comparison with machine reading comprehension models, PQA models are closer to retrieval models for ranking, so we develop a novel framework that has a rejection model (a conformal predictor) on top of a ranking model. Our proposed framework has a totally different way of handling

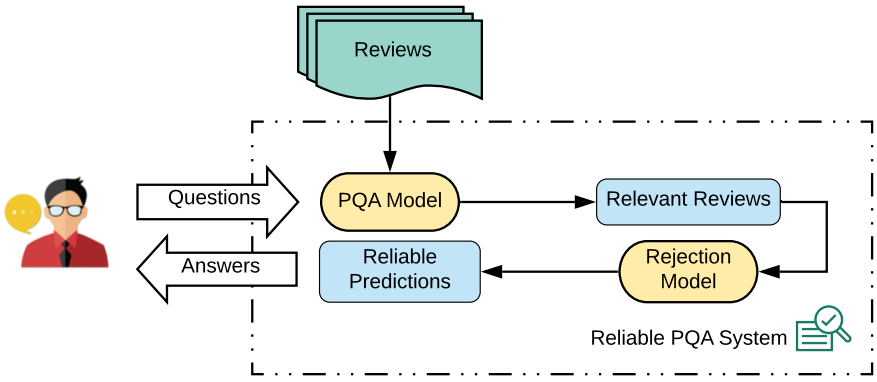


Figure 6.1: Proposed PQA with a rejection model framework.

unanswerable questions compared to how machine reading comprehension dealing with unanswerable questions.

6.3 Methodology

Our proposed framework consists of two components: a PQA model that predicts the relevance of reviews, given a product query, and a rejection model that rejects unconfident reviews and produces only confident reviews as answers. An illustration of our framework is presented in Figure 6.1.

More specifically, the PQA component (a binary classifier) models the function $\hat{y} = F(r, q)$, where r is a review, q is a product question, and $\hat{y}$ is the probability of the positive class, i.e., the review is relevant to the question.

The rejection model takes the output $\hat{y}$ from the PQA model and transforms it into a confidence score y^*. Given the confidence score, we can then set a significance level $\epsilon \in [0, 1]$ to reject/accept predictions. For example, if the confidence score of the positive class (review is relevant to the question) is 0.6 and ϵ is 0.5, then we accept the positive class prediction and return the review as relevant. On the other hand, if ϵ is 0.7, we reject the review.

6.3.1 PQA Models

We explore three state-of-the-art PQA models for our task:

6.3.1.1 MOQA

MOQA [22] is inspired by the MoE classifier. In a MoE classifier, a prediction is made by a weighted combination of a number of weak classifiers. Using answer prediction ($P(a|q)$) as its objective, MOQA decomposes the problem into: $P(a|q) = \sum_r P(a|r)P(r|q)$, where a, q, r = are answer, query and review, respectively. Each review can be interpreted as an "expert," where it makes a prediction on the answer ($P(a|r)$) and its prediction is weighted by its confidence ($P(r|q)$). The advantage of doing this decomposition is that the learning objective is now $P(a|q)$, which allows the model to make use of the abundance of existing questions and answers on e-commerce platforms. In practice, MOQA is optimized with a maximum margin objective: given a query, the goal is to score a real answer higher than a randomly selected non-answer. To model the two relevance functions ($P(a|r)$ and $P(r|q)$), MOQA uses an off-the-shelf pairwise similarity function such as BM25+ or ROUGE-L and a learned bilinear scoring function that uses bag-of-words representation as input features. After the model is trained, the function of interest is $P(r|q)$, which can be used to score a review for a given query. We use the open source implementation and its default optimal configuration.[3]

6.3.1.2 FLTR

FLTR proposed in the previous chapter is a BERT classifier for answer prediction. Using existing QA pairs on e-commerce platforms, FLTR fine-tunes a pre-trained BERT to classify answers given a question. After fine-tuning, FLTR is used to classify *reviews* for a given question. FLTR can be seen as a form of zero-shot domain transfer; where it is trained in the (answer, question) domain but at test time it is applied to the (review, question) domain.

[3] https://cseweb.ucsd.edu/$\sim$jmcauley

6.3.1.3 BERTQA

BERTQA proposed in the previous chapter is an extension of MOQA, which uses the same MoE framework, but parameterizes the relevance functions ($P(a|r)$ and $P(r|q)$) with neural networks: BERT. BERTQA addresses the vocabulary/language mismatch between different domains (e.g., answer vs. review, or review vs. query) using contextualized representations and fine-grained comparison between words via attention. In the previous chapter, we demonstrated that this neural parameterization substantially improves review discovery compared to MOQA. The downside of BERTQA is its computational cost: while MOQA can be used to compute the relevance for every review for a given query (which can number from hundreds to thousands), this is impractical with BERTQA. To ameliorate this, we proposed using FLTR to pre-filter reviews to reduce the set of reviews to be ranked by BERTQA.[4]

6.3.2 Rejection Model

In a classification task, a model may make a wrong prediction for a difficult instance, particularly when the positive class probability is around 0.5 in a binary task. For medical applications such as tumor diagnosis, misclassification can have serious consequences. In these circumstances, rejection techniques are used to reduce misclassification rates by rejecting unconfident or unreliable predictions [124].

To apply rejection techniques to PQA, we need to first understand the associated risks in PQA. The risks in PQA are: (1) including irrelevant reviews as part of the returned results for an answerable question, and (2) returning any reviews for an unanswerable question. These are similar to the risks proposed for Web QA [34], although a crucial difference is that PQA is a ranking problem (the output is a list of reviews). The implication is that when deciding the ϵ threshold to guarantee an "error rate", we are using a ranking metric as opposed to a classification metric. As standard ranking metrics such as normalized discounted cumulative gain are unable to account for unanswerable questions, we explore alternative metrics (detailed in Section 6.4.2).

[4]The original implementation uses a softmax activation function to compute $P(r|q)$ (and so the probability of all reviews sum up to one); we make a minor modification to the softmax function and use a sigmoid function instead (and so each review produces a valid probability distribution over the positive and negative classes).

Table 6.2: An example of predicted labels with different choices of significance level. p-values for the positive (1) and negative (0) labels are 0.65 and 0.45, respectively.

ϵ	**Predicted Labels**	**Outcome**
0.05	{0, 1}	Rejected
0.45	{1}	Accepted
0.75	{}	Rejected

We propose to use CPs [129, 130] as the rejection model. Intuitively, for a test example, a CP computes a nonconformity score that measures how well the new example *conforms* to the previously seen examples as a way to estimate the reliability of the prediction. CPs are typically applied to the output of other machine learning models, and can be used in both classification and regression tasks. There are two forms of CP, namely inductive CP (ICP) and transductive CP (TCP). They differ in their training schemes: in TCP, the machine learning model is updated for each new examples (and so requires re-training) to compute the nonconformity score, while in ICP the model is trained once using a subset of the training data, and the other subset—the *calibration set*—is set aside to be used to compute nonconformity scores for a new example. As such, the associated computation cost for the transductive variant is much higher due to the re-training. We use the IMCP for our rejection model, which is a modified ICP that is better at handling imbalanced data. When calculating a confidence score, IMCP additionally conditions it on the class label. In PQA, there are typically many more irrelevant reviews than relevant reviews for given a query, and so IMCP is more appropriate for our task.

Given a bag of calibration examples $\{(x_1, y_1), ...(x_n, y_n)\}$ and a binary classifier $\hat{y} = f(x)$, where n is the number of calibration examples, x is the input, y is the true label and $\hat{y}$ is the predicted probability for the positive label, we compute a nonconformity score, $\alpha(x_{n+1})$, for a new example x_{n+1} using its inverse probability:

$$\alpha(x_{n+1}) = -f(x_{n+1})$$

As IMCP conditions the nonconformity score on the label, there are two nonconformity

scores for x_{n+1}, one for the positive label, and one for the negative label:

$$\alpha(x_{n+1}, 1) = -f(x_{n+1})$$
$$\alpha(x_{n+1}, 0) = -(1 - f(x_{n+1}))$$

We then compute the confidence score (p-value) for x_{n+1} conditioned on label k as follows:

$$p(x_{n+1}, k) = \frac{\sum_{i=0}^{n} I(\alpha(x_i, k) \geq \alpha(x_{n+1}, k))}{n+1} \tag{6.1}$$

where I is the indicator function.

Intuitively, we can interpret the p-value as a measure of how confident/reliable the prediction is for the new example by comparing its predicted label probability to that of the calibration examples.

Given the p-values (for both positive and negative labels), the rejection model accepts a prediction for all labels k where $p(x_{n+1}, k) > \epsilon$, where $\epsilon \in [0, 1]$ is the significance level. We present an output of the rejection model in Table 6.2 with varying ϵ, for an example whose p-values for the positive and negative labels are 0.65 and 0.45, respectively. Depending on ϵ, the number of predicted labels ranges from zero to two. For PQA, as it would not make sense to have both positive and negative labels for an example (i.e., indicating that the review is both relevant and irrelevant for a query), we reject such an example and consider it an unreliable prediction.

6.4 Experiments

6.4.1 Data

We use the Amazon dataset developed by Mcauley et al. [22] for our experiments. The dataset contains QA pairs and reviews for each product. We train MOQA, BERTQA and FLTR on this data, noting that MOQA and BERTQA leverage both QA pairs and reviews, while FLTR uses only QA pairs. After the models are trained, we can use the $P(r|q)$ relevance function to score reviews given a product question.

To assess the quality of the reviews returned for a question by the PQA models, we ask crowdworkers on Mechanical Turk to judge how well a review answers a question. We randomly

Table 6.3: Answerable question statistics.

Relevance Threshold	2.00	2.25	2.50	2.75	3.00
#Relevant Reviews	640	351	175	71	71
#Answerable Questions	170	134	89	44	44
%Answerable Questions	85%	67%	45%	22%	22%

select 200 questions from four categories ("Tools and Home Improvement," "Patio Lawn and Garden," "Baby," and "Electronics"), and pool the top ten reviews returned by the three PQA models (MOQA, BERTQA, and FLTR),[5] resulting in approximately 20 to 30 reviews per question (total number of reviews = 4,691).[6]

In the survey, workers are presented with a question and review pair, and asked to judge how well the review answers the question on an ordinal scale: 0 (completely irrelevant), 1 (related but does not answer the question), 2 (somewhat answers the question) or 3 (directly answers the question). Each question/review pair is annotated by three workers, and the final relevance score for each review is computed by taking the mean of three scores.

Given the annotated data with relevance scores, we can set a relevance threshold to define a cut-off when a review answers a question, allowing us to control how precise we want the system to be (a higher threshold implies a more precise system). In Table 6.3 we present some statistics with different relevance thresholds. For example, when the threshold is set to 2.00, a review with a relevance score of less than 2.00 is considered irrelevant (and therefore its score is set to 0.00), and a question where all reviews have relevance scores of less than 2.00 is unanswerable. The varying relevance thresholds will produce different distributions of relevant/irrelevant reviews and answerable/unanswerable questions; at the highest threshold (3.00), only a small proportion of the reviews are relevant (but all these provide high-quality answers), and most questions are unanswerable.

Table 6.4: NDCG' examples.

Question Type	Systems	Doc List	NDCG'
Answerable	System A	111	1.000
	System B	11100	0.971
	System C	11	0.922
Unanswerable	System A	$\emptyset$	1.000
	System B	00	0.500
	System C	000	0.431

6.4.2 Evaluation Metric

As PQA is a retrieval task, it is typically evaluated using ranking metrics such as normalized discounted cumulative gain (NDCG) [238]. Note, however, that NDCG is not designed to handle unanswerable queries (i.e., queries with no relevant documents) and, as such, it is not directly applicable to our task. We explore a variant, NDCG', which is designed to work with unanswerable queries [239]. The idea of NDCG' is to "quit while ahead": the returned document list should be truncated earlier rather than later, as documents further down in the list are more likely to be irrelevant. To illustrate this, we present an answerable question in Table 6.4. Assuming it has three relevant documents (1 represents a relevant document and 0 an irrelevant document), System A receives a perfect NDCG' score while System B is penalized for including 2 irrelevant documents. System C has the lowest NDCG' score as it misses one relevant document. The second example presents an unanswerable question. The ideal result is the empty list ($\emptyset$) returned by System A, which receives a perfect score. Comparing System B to C, NDCG' penalizes C for including one more irrelevant document.

NDCG' appends a terminal document (t) to the end of the document list returned by a ranking system. For example, "111" $\rightarrow$ "111t", and "11100" $\rightarrow$ "11100t". The corresponding

[5]Following the original papers, a "review" is technically a "review sentence" rather than the full review.

[6]To control for quality, we insert a control question with a known answer (from the question and answer pair) in every three questions. Workers who consistently give low scores to these control questions are filtered out.

gain value for the terminal document t is r_t, calculated as follows:

$$r_t = \begin{cases} 1 & \textbf{if } R = 0 \\ \sum_{i=1}^{d} r_i / R & \textbf{if } R > 0 \end{cases}$$

where R is the total number of ground truth relevant documents, and r_i is the relevance of document i in the list. As an example, for the document list "11" produced by System C, $r_t = \frac{1}{3} + \frac{1}{3} = \frac{2}{3}$.

Given r_t, we compute NDCG' for a ranked list of d items as follows:

$$\text{NDCG'}_d = \frac{\text{DCG}_{d+1}\langle r_1, r_2, ..., r_d, r_t \rangle}{\text{IDCG}_{d+1}}$$

With NDCG, for an unanswerable question like the second example, both System B ("00") and System C ("000") will produce a score of zero, there failing to indicate that B is technically better. NDCG' solves this problem by introducing the terminal document score r_t.

In practice, the relevance of our reviews is not binary (unlike the toy examples). That is, the relevance of each review is a mean relevance score from three annotators, and ranges from 0–3. Note, however, that given a particular relevance threshold (e.g., 2.0), we mark all reviews under the threshold as irrelevant by setting their relevance score to 0.0.

In our experiments, we compute NDCG' up to a list of 10 reviews ($d = 10$), and separately for answerable ($\mathcal{N}_A$) and unanswerable questions ($\mathcal{N}_U$). To aggregate NDCG' over two question types ($\mathcal{N}_{A+U}$), we compute the geometric mean:

$$\mathcal{N}_{A+U} = \sqrt{\mathcal{N}_A \times \mathcal{N}_U}$$

We use geometric mean here because we want an evaluation metric that favors a balanced performance between answerable and unanswerable questions [240]. In preliminary experiments, we found that micro-average measures result in the selection of a system that always returns no results when a high relevance threshold is selected (e.g., ≥ 2.50 in Table 6.3) since a large number of questions are unanswerable. This is undesirable in a real application where choosing a high relevance threshold means we want a very precise system, and not one that never gives any answers.

6.4.3 Experimented Methods

We compare the following methods in our experiments:

Vanilla PQA Model: a baseline where we use the top ten reviews returned by a PQA model (MOQA, FLTR, or BERTQA) without any filtering/rejection.

PQA Model+thrs: a second baseline where we tune a threshold based on the review score returned by a PQA model (MOQA, FLTR, or BERTQA) to truncate the document list. We use leave-one-out cross-validation for tuning. That is, we split the 200 annotated questions into 199 validation examples and 1 test example, and find an optimal threshold for the 199 validation examples based on $\mathcal{N}_{A+U}$. Given the optimal threshold, we then compute the final $\mathcal{N}_{A+U}$ on the 1 test example. We repeat this 200 times to obtain the average $\mathcal{N}_{A+U}$ performance.

PQA Model+imcp: our proposed framework that combines PQA and IMCP as the rejection model. Given a review score by a PQA model, we first convert the score into probabilities,[7] and then compute the p-values for both positive and negative labels (Equation (6.1)). We then tune the significance level ϵ to truncate the document list, using leave-one-out cross-validation as before. As IMCP requires calibration data to compute the p-value, the process is a little more involving. We first split the 200 questions into 199 validation examples and 1 test examples as before, and within the 199 validation examples, we do another leave-one-out: we split them into 198 calibration examples and 1 validation example, and compute the p-value for the validation example based on the 198 calibration examples. We then tune ϵ and find the optimal threshold that gives the best $\mathcal{N}_{A+U}$ performance on the single validation performance, and repeat this 199 times to find the best overall ϵ. Given this ϵ, we then compute the $\mathcal{N}_{A+U}$ performance on the 1 test example. This whole process is then repeated 200 times to compute the average $\mathcal{N}_{A+U}$ test performance.

[7] This step is only needed for MOQA, as BERTQA and FLTR produce probabilities in the first place. For MOQA, we convert the review score into a probability applying a sigmoid function to the log score.

Table 6.5: Model performance; boldface indicates optimal $\mathcal{N}_{A+U}$ performance for a PQA model.

Relevance	**Model**	$\mathcal{N}_{A+U}$	$\mathcal{N}_A$	$\mathcal{N}_U$
≥ 2.00	MOQA	0.294	0.309	0.279
	MOQA+THRS	**0.319**	0.212	0.480
	MOQA+IMCP	0.318	0.212	0.477
	FLTR	0.372	0.495	0.279
	FLTR+THRS	**0.516**	0.400	0.666
	FLTR+IMCP	0.514	0.392	0.675
	BERTQA	0.360	0.464	0.279
	BERTQA+THRS	0.436	0.356	0.534
	BERTQA+IMCP	**0.447**	0.345	0.580
≥ 2.25	MOQA	0.264	0.249	0.279
	MOQA+THRS	**0.296**	0.179	0.489
	MOQA+IMCP	0.295	0.163	0.535
	FLTR	0.361	0.468	0.279
	FLTR+THRS	0.452	0.335	0.608
	FLTR+IMCP	**0.482**	0.329	0.705
	BERTQA	0.344	0.423	0.279
	BERTQA+THRS	0.373	0.293	0.477
	BERTQA+IMCP	**0.405**	0.310	0.530
≥ 2.50	MOQA	0.243	0.211	0.279
	MOQA+THRS	**0.274**	0.165	0.453
	MOQA+IMCP	0.265	0.155	0.452
	FLTR	0.359	0.462	0.279
	FLTR+THRS	0.439	0.326	0.592
	FLTR+IMCP	**0.470**	0.316	0.699
	BERTQA	0.340	0.414	0.279
	BERTQA+THRS	**0.404**	0.308	0.530
	BERTQA+IMCP	0.387	0.294	0.510
≥ 2.75	MOQA	**0.235**	0.199	0.279
	MOQA+THRS	0.229	0.129	0.407
	MOQA+IMCP	0.213	0.107	0.423
	FLTR	0.333	0.397	0.279
	FLTR+THRS	0.409	0.272	0.615
	FLTR+IMCP	**0.416**	0.299	0.577
	BERTQA	0.330	0.390	0.279
	BERTQA+THRS	0.349	0.279	0.435
	BERTQA+IMCP	**0.388**	0.296	0.509

6.4.4 Results

We present the full results in Table 6.5, reporting NDCG' performances over 4 relevance thresholds: 2.00, 2.25, 2.50, and 2.75.

We will first focus on the combined performances ($\mathcal{N}_{A+U}$). In general, all models (MOQA, FLTR, and BERTQA) see an improvement compared to their vanilla models when we tune a threshold (+THRS or +IMCP) to truncate the returned review list, implying that it is helpful to find a cut-off to discover a more concise set of reviews. Comparing the simple thresholding method (+THRS) and conformal method (+IMCP), we also see very encouraging results: for both FLTR and BERTQA, +IMCP is consistently better than +THRS for most relevance thresholds, suggesting that +IMCP is a better rejection model. For MOQA, however, +THRS is marginally better than +IMCP. We hypothesize that this may be due to MOQA producing arbitrary (non-probabilistic) scores for reviews; and as such, it is less suitable for conformal predictions. Comparing the three PQA models, FLTR consistently produces the best performance: across most relevance thresholds, FLTR+IMCP maintains an NDCG' performance close to 0.5.

Looking at the $\mathcal{N}_U$ results, we notice that all vanilla models produce the same performance (0.279). This is because there are no relevant reviews for these unanswerable questions, and therefore the top ten reviews returned by any models are always irrelevant. When we introduce +THRS or +IMCP to truncate the reviews, we see a very substantial improvement ($\mathcal{N}_U$ more than doubled in most cases) for all models over different relevance thresholds. Generally, we also find that +IMCP outperforms +THRS, demonstrating that the conformal predictors are particularly effective for the unanswerable questions.

However, when we look at the $\mathcal{N}_A$ performance, this is consistently worse when we introduce rejection (+THRS or +IMCP). This is unsurprising, as ultimately there is a trade-off between precision and recall: when we introduce a rejection model to truncate the review list, we may produce a more concise/shorter list (as seen for the unanswerable questions), but we may also inadvertently exclude potentially relevant reviews. As such, the best system is one that can maintain a good balance between pruning unreliable reviews and avoiding discarding potentially relevant reviews.

Table 6.6: Reviews produced by FLTR, FLTR+THRS and FLTR+IMCP for an answerable (Q1) and unanswerable (Q2) question.

Q1: How long the battery lasts on X1 carbon touch?		
	Ground Truth	[3, 3]
	FLTR	[0, 0, 0, 0, 3, 0, 0, 0, 0, 0]
	FLTR+THRS	[0, 0, 0, 0, 3, 0, 0, 0]
	FLTR+IMCP	[0, 0, 0, 0, 3]
Q2: What type of memory SD card should I purchase to go with this?		
	Ground Truth	[ ]
	FLTR	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
	FLTR+THRS	[0, 0]
	FLTR+IMCP	[ ]

Next, in Table 6.6 we present two real outputs of how these methods perform. The first question (Q1) is an answerable question, and the ground truth contains two relevant reviews (numbers in the list are review relevance scores). FLTR returns ten reviews, of which one is relevant. FLTR+THRS rejects the last two reviews, and FLTR+IMCP rejects three more reviews, producing a concise list of five reviews (no relevant reviews were discarded in this case). It can be seen that both FLTR+THRS and FLTR+IMCP reject predictions but do not modify the original ranking of the reviews returned by FLTR. +THRS tunes a threshold based on original relevance score, and in +IMCP the conversion of class probability to confidence score (p-value) is a monotonic transformation; and as such, the original order is preserved in both methods. This also means that if the vanilla model misses a relevant review, the review will not be recovered by the rejection model, as we see here.

The second question (Q2) is an unanswerable question (ground truth is an empty list). FLTR always returns ten reviews, and so there are ten irrelevant reviews. FLTR+THRS discards most of the reviews, but there are still two irrelevant reviews. FLTR+IMCP returns an empty list, indicating that existing reviews do not have useful information to answer Q2 and detecting correctly that Q2 is an unanswerable question.

6.5 Summary

PQA is often formulated as a retrieval problem with the goal of finding the most relevant reviews to answer a given product question. In this chapter, we have proposed incorporating conformal predictors (CPs) as a rejection model to a PQA model to reject unreliable reviews. We have tested three state-of-the-art PQA models, MOQA, FLTR and BERTQA, and found that incorporating conformal predictors as the rejection model helps filter unreliable reviews better than a baseline approach. More generally, this chapter demonstrates a novel and effective application of conformal predictors to a retrieval task.

Chapter 7

Conclusion

This chapter first summarizes the work presented in the thesis and the findings of each chapter. It then discusses possible directions of future work.

7.1 Summary of Research Findings

In Chapter 3, we addressed one of the important problems in Twitter sentiment analysis, which is the insufficient resources, especially annotated data, for languages other than English. We proposed two CNN-based models for language-independent Twitter classification. The first proposed model is a character-based CNN with Unicode UTF-8 encoding. By using UTF-8 character encoding, our model dispenses with the need for language identification and is therefore a fully language-independent approach. Furthermore, using the character-based CNN on learning distributed character representations leads to promising classification accuracy. Experiments showed that our method is fully language independent. Most importantly, our UniCNN model is the first neural model that can work with Tweets written in mixed languages. Lastly, our model does not need language knowledge to pre-define an alphabet. Our second proposed model is a word-character CNN, which leverages the deep convolutional features of short texts at both the word and character levels. Based on experiments, it is clear that using two different transformations of short text as the input of CNN for Twitter sentiment classification gives much better performance than using only words or characters. In terms

of limitations, our proposed models have not used pre-trained word/character representations since there is not pre-trained word/character representations available for all languages. In recent years, using pre-trained representations become crucial to train deep learning-based models. Training a language-independent language model is complicated, but it is worth trying. In addition, tokenization is a process that splits texts into characters, words or subwords. The most popular tokenization algorithms are WordPiece [241] and SentencePiece [242] which have been adopted by most transformer-based models. However, it is still challenging to develop a tokenization algorithm that is independent of languages. Therefore, learning a language-independent language model is an important and interesting topic to explore in the future.

In Chapter 4, we studied the problem of irony detection on Twitter. The most challenging part of training a good automatic irony detection model is the limited human-labeled dataset. In contrast to irony detection, sentiment analysis has sufficient resources, such as pre-defined sentiment lexicons and human-annotated corpora. In this chapter we therefore investigated the following two research questions:

- How can different types of sentiment knowledge be transferred for irony detection?
- How can the transferred knowledge be effectively used to detect context incongruity, especially the implicit context incongruity?

We proposed three models to take advantage of these widely and readily available sentiment resources to improve the attention-based model's ability to detect context incongruity. By incorporating transferred sentiment, our models are able to detect both implicit and explicit context incongruity at most times. Experiments show that our three proposed sentiment attention mechanisms result in better performance than the baselines, including several popular neural models, for irony detection on Twitter. However, the performance of all models is still not very satisfactory, especially on human-label datasets (accuracy under 80%). According to our analysis on misclassified instances, to improve the performance of irony detection models we need to collect more tweets for a particular topic in order to provide more background information for models. In addition, large language models play an important role in text clas-

sification systems. By pre-training language models on a large volume of unlabeled datasets, it is possible to encode some common knowledge into language models. However, the objectives of language models are mainly focusing on reconstructing original text. To improve irony detection, it is important to develop a training objective aiming to learn long-term sentence dependences because detecting irony often requires models to connect information from different positions of a context

In Chapter 5, we investigated how to develop deep learning-based text classification approaches for mining product reviews to answer product-related questions. The two key challenges are a lack of ground truth and the language mismatch between user queries and reviews. To address these two challenges, we proposed two neural models that discover relevant reviews for answering product queries, using existing question and answer pairs as training signals and pre-trained embeddings or language models as the start weights of the relevance function. We demonstrate that our best model produces strong performance, outperforming state-of-the-art systems by consistently finding the most relevant reviews for product queries. Experiments show our model can extract relevant reviews to answer questions, but from user studies we find that about half returned reviews are not fully useful. To further improve the performance of our models, two possible solutions are worthwhile trying, namely improving the reliability of models which we investigated in Chapter 6 and further pre-training language models in target domain [243]. Language models are usually learned on a large amount of web documents, such as Wikipedia. The textual resources are reviews in PQA, which have different vocabularies and knowledge from Wikipedia. Therefore, further pre-training language models on reviews could add domain specific knowledge into language models, which eventually can help PQA models select answers.

In Chapter 6, we investigated how to improve the reliability of a PQA model so that the returned list of answers contains fewer irrelevant reviews. In this chapter, we specifically tackled the following two challenges:

- Choosing an appropriate model as the rejection model.
- Identifying risks in PQA and choosing an appropriate risk function.

In our proposed rejection model, we adopted a conformal predictor as the confidence function to generate confidence scores for each prediction. Furthermore, we chose NDCG' as the risk function. By optimizing the risk function, our model is able to tune a threshold for rejecting unreliable answers so that the returned results are more concise and accurate at answering product questions, including returning nil answers for unanswerable questions. Experiments on a widely used Amazon dataset show encouraging results for our proposed framework. More broadly, our results demonstrate a novel and effective application of conformal methods to a retrieval task. Conformal methods are efficient and effective, but they also have drawbacks. In our experiments, we find that relevant reviews missed by PQA models will also not be returned by our conformal prediction-based rejection model, because conformal methods perform linear transformation on class probabilities. To further improve the performance of the rejection model, more features from PQA models should be considered when generating confidence scores and other models could also be tested as the rejection model.

7.2 Future Directions

We now discuss future work in three directions related to our current work, moving the focus to transfer learning and reliability issues.

Our first direction is to investigate how to leverage both character representations and language models for language-independent sentiment analysis and opinion mining. Chapter 3 showed that character representations have the advantage of being fully language-independent when building a model, in particular not requiring language identification. However, the current work continues to involve learning character representations via supervised training. In recent years, language models such as GPT [25] and BERT [24] have achieved high-quality results on many NLP tasks. Language models usually first split inputs into sub-words and then learn contextual representations of sub-words via unsupervised training, but they still require language identification. If we can take the advantages of both character representations and language models, this could further improve the performance of character-based models on language-independent sentiment analysis and opinion mining tasks.

Looking in a second direction, future work should investigate how best to leverage pre-

trained language models for user-generated problems, such as Twitter and reviews related classification tasks. Our current approach follows the dominant transfer learning method and is roughly a two-stages process: a language model is first pre-trained on a large text corpus and then fine-tuned on downstream tasks [24]. We are interested in incorporating more advanced fine-tuning approaches. One promising approach we are looking at is multitask learning [117]. The intuition is that learning knowledge from multiple tasks contributes to language understanding and that, conversely, a higher level of language understanding could benefit individual tasks. Another transfer learning approach is transferring with adapter modules [121], where an adapter is a neural layer that has a number of trainable parameters. The significant advantage of transferring with adapter modules is that it provides an efficient approach for fine-tuning parameters. For example, product reviews can be used in multiple tasks, such as sentiment analysis, spam detection, irony detection, and question answering. Each task has its own trained model, which is time consuming in terms of training and is also space consuming in terms of storing all trained models. When transferring with adapter modules, we only need to train a model once for all tasks and keep all parameters fixed. For each task, we only need to train adapters. Therefore, we only need to keep a single model, together with some adapters, thus saving considerable memory space. Another benefit of transferring with adapter modules is that we can treat adapters as interfaces in object-oriented programming and different pre-trained models can be treated as different classes, increasing the reusability of those pre-trained models.

The third direction for our future work leads to investigating more advanced approaches for improving the reliability of the model proposed in this thesis. In our current risk controlling framework for PQA, we adopted conformal predictors [129] to generate confidence scores for predictions, building on top of class probabilities. Class probabilities can be considered a feature for measuring the reliability of neural models. So what other features could also be useful? Linear classifier probes [127] look at intermediate layers of neural models; we can add a classifier to each layer and then use predictions of each layer as features for measuring the performance of neural models [34]. In addition, leveraging predictions of multiple models (ensembles of neural networks) has been found promising for higher accuracy and better reliability

because using ensemble of neural networks helps to reduce the variance [138].

Appendix A

Publications

- **Chapter 3:** [1] Shiwei Zhang, Xiuzhen Zhang, and Jeffrey Chan. Language-independent Twitter Classification using Character-based Convolutional Networks. In Proceedings of the International Conference on Advanced Data Mining and Applications, pages 413—425, 2017. [2] Shiwei Zhang, Xiuzhen Zhang, and Jeffrey Chan. A Word-character Convolutional Neural Network for Language-agnostic Twitter Sentiment Analysis. In Proceedings of the Australasian Document Computing Symposium, pages 1–4, 2017.

- **Chapter 4:** Shiwei Zhang, Xiuzhen Zhang, Jeffrey Chan, and Paolo Rosso. Irony Detection via Sentiment-based Transfer Learning. Information Processing & Management, 56(5):1633–1644, 2019.

- **Chapter 5:** Shiwei Zhang, Jey Han Lau, Xiuzhen Zhang, Jeffrey Chan, and Cecile Paris. Discovering Relevant Reviews for Answering Product-related Queries. In Proceedings of the International Conference on Data Mining, pages 1468–1473, 2019.

- **Chapter 6:** Shiwei Zhang, Xiuzhen Zhang, Jey Han Lau, Jeffrey Chan, and Cecile Paris. Less is More: Rejecting Unreliable Reviews for Product Question Answering. In Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, pages 1–17, 2020.

Bibliography

[1] Bing Liu. Sentiment analysis and opinion mining. *Synthesis Lectures on Human Language Technologies*, 5(1):1–167, 2012.

[2] Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 1555–1565, 2014.

[3] Muhammad Abdul-Mageed and Lyle Ungar. Emonet: Fine-grained emotion detection with gated recurrent neural networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 718–728, 2017.

[4] Michael Völske, Martin Potthast, Shahbaz Syed, and Benno Stein. Tl; dr: Mining reddit to learn automatic summarization. In *Proceedings of the Workshop on New Frontiers in Summarization*, pages 59–63, 2017.

[5] Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. The e2e dataset: New challenges for end-to-end generation. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 201–206, 2017.

[6] Bo Han and Timothy Baldwin. 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*, pages 368–378, 2011.

[7] Laura Jehl, Felix Hieber, and Stefan Riezler. Twitter translation using translation-based cross-lingual retrieval. In *Proceedings of the 17th Workshop on Statistical Machine Translation*, pages 410–421, 2012.

[8] Orestes Appel, Francisco Chiclana, Jenny Carter, and Hamido Fujita. A hybrid approach to the sentiment analysis problem at the sentence level. *Knowledge-Based Systems*, 108:110–124, 2016.

[9] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014.

[10] Fatemeh Hemmatian and Mohammad Karim Sohrabi. A survey on classification techniques for opinion mining and sentiment analysis. *Artificial Intelligence Review*, pages 1–51, 2019.

[11] Yequan Wang, Minlie Huang, Li Zhao, et al. Attention-based lstm for aspect-level sentiment classification. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 606–615, 2016.

[12] DI Hernández Farias and Paolo Rosso. Irony, sarcasm, and sentiment analysis. In *Sentiment Analysis in Social Networks*, pages 113–128. Elsevier, 2017.

[13] Paolo Rosso, Francisco Rangel, Irazu Hernández Farías, Leticia Cagnina, Wajdi Zaghouani, and Anis Charfi. A survey on author profiling, deception, and irony detection for the arabic language. *Language and Linguistics Compass*, 12(4):e12275, 2018.

[14] Roberto González-Ibáñez, Smaranda Muresan, and Nina Wacholder. Identifying sarcasm in twitter: A closer look. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 581–586, 2011.

[15] Aditya Joshi, Vinita Sharma, and Pushpak Bhattacharyya. Harnessing context incongruity for sarcasm detection. In *Proceedings of the 53rd Annual Meeting of the Associa-*

tion for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, volume 2, pages 757–762, 2015.

[16] Ellen Riloff, Ashequl Qadir, Prafulla Surve, Lalindra De Silva, Nathan Gilbert, and Ruihong Huang. Sarcasm as contrast between a positive sentiment and negative situation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 704–714, 2013.

[17] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, volume 2, pages 207–212, 2016.

[18] Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015.

[19] Debanjan Ghosh, Alexander Richard Fabbri, and Smaranda Muresan. The role of conversation context for sarcasm detection in online interactions. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 186–196, 2017.

[20] Bjarke Felbo, Alan Mislove, Anders Søgaard, Iyad Rahwan, and Sune Lehmann. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1615–1625, 2017.

[21] Yu-Hsiang Huang, Hen-Hsen Huang, and Hsin-Hsi Chen. Irony detection with attentive recurrent neural networks. In *Proceedings of European Conference on Information Retrieval*, pages 534–540. Springer, 2017.

[22] Julian McAuley and Alex Yang. Addressing complex and subjective product-related queries with customer reviews. In *Proceedings of the 25th International Conference on World Wide Web*, pages 625–635. International World Wide Web Conferences Steering Committee, 2016.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, 2019.

[25] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training (2018). *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/language-unsupervised/language_ understanding_paper. pdf*, 2018.

[26] Zhuyun Dai. *Neural Matching and Importance Learning in Information Retrieval.* PhD thesis, Tsinghua University, 2020.

[27] Joel Janai, Fatma Güney, Aseem Behl, and Andreas Geiger. Computer vision for autonomous vehicles: Problems, datasets and state of the art. *Foundations and Trends® in Computer Graphics and Vision*, 12(1–3):1–308, 2020.

[28] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 13991–14002, 2019.

[29] Simon Hecker, Dengxin Dai, and Luc Van Gool. Failure prediction for autonomous driving. In *IEEE Intelligent Vehicles Symposium*, pages 1792–1799. IEEE, 2018.

[30] Minghao Hu, Furu Wei, Yuxing Peng, Zhen Huang, Nan Yang, and Dongsheng Li. Read+ verify: Machine reading comprehension with unanswerable questions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6529–6537, 2019.

[31] Fu Sun, Linyang Li, Xipeng Qiu, and Yang Liu. U-net: Machine reading comprehension with unanswerable questions. *arXiv preprint arXiv:1810.06638*, 2018.

[32] Fréderic Godin, Anjishnu Kumar, and Arpit Mittal. Learning when not to answer: a ternary reward structure for reinforcement learning based question answering. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 122–129, 2019.

[33] Kevin Huang, Yun Tang, Jing Huang, Xiaodong He, and Bowen Zhou. Relation module for non-answerable predictions on reading comprehension. In *Proceedings of the 23rd Conference on Computational Natural Language Learning*, pages 747–756, 2019.

[34] Lixin Su, Jiafeng Guo, Yixin Fan, Yanyan Lan, and Xueqi Cheng. Controlling risk of web question answering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–124, 2019.

[35] Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. Deep learning based text classification: A comprehensive review. *arXiv preprint arXiv:2004.03705*, 2020.

[36] Yuefeng Li, Libiao Zhang, Yue Xu, Yiyu Yao, Raymond Yiu Keung Lau, and Yutong Wu. Enhancing binary classification by modeling uncertain boundary in three-way decisions. *IEEE Transactions on Knowledge and Data Engineering*, 29(7):1438–1451, 2017.

[37] Thorsten Joachims. *Learning to classify text using support vector machines*, volume 668. Springer Science & Business Media, 2002.

[38] Rie Johnson and Tong Zhang. Effective use of word order for text categorization with convolutional neural networks. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, page 103, 2015.

[39] Jin Wang, Liang-Chih Yu, K Robert Lai, and Xuejie Zhang. Dimensional sentiment analysis using a regional cnn-lstm model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 225–230, 2016.

[40] Jason D Rennie, Lawrence Shih, Jaime Teevan, and David R Karger. Tackling the poor assumptions of naive bayes text classifiers. In *Proceedings of the 20th International Conference on Machine Learning*, pages 616–623, 2003.

[41] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer, 2019.

[42] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, pages 5754–5764, 2019.

[43] Yang Liu and Mirella Lapata. Text summarization with pretrained encoders. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 3721–3731, 2019.

[44] Edward Loper and Steven Bird. 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*, pages 63–70, 2002.

[45] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[46] Efstathios Stamatatos. Plagiarism detection using stopword n-grams. *Journal of the American Society for Information Science and Technology*, 62(12):2512–2527, 2011.

[47] Anjali Ganesh Jivani et al. A comparative study of stemming algorithms. *Int. J. Comp. Tech. Appl*, 2(6):1930–1938, 2011.

[48] Peter Willett. The porter stemming algorithm: then and now. *Program: Electronic Library & Information Systems*, 40(3):219–223, 2006.

[49] Massimo Melucci and Nicola Orio. A novel method for stemmer generation based on hidden markov models. In *Proceedings of the 12th International Conference on Information and Knowledge Management*, pages 131–138, 2003.

[50] Robert Krovetz. Viewing morphology as an inference process. *Artificial Intelligence*, 118(1-2):277–294, 2000.

[51] Joël Plisson, Nada Lavrac, Dunja Mladenic, et al. A rule based approach to word lemmatization. In *Proceedings of IS*, volume 3, pages 83–86, 2004.

[52] Bo Han, Paul Cook, and Timothy Baldwin. Automatically constructing a normalisation dictionary for microblogs. In *Proceedings of the 2012 joint Conference on Empirical Methods in Matural Language Processing and Computational Natural Language Learning*, pages 421–432, 2012.

[53] William B Cavnar, John M Trenkle, et al. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, volume 161175. Citeseer, 1994.

[54] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

[55] Bruce M Hill. Posterior distribution of percentiles: Bayes' theorem for sampling from a population. *Journal of the American Statistical Association*, 63(322):677–691, 1968.

[56] Wee Sun Lee and Bing Liu. Learning with positive and unlabeled examples using weighted logistic regression. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, pages 448–455, 2003.

[57] Robert E Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2-3):135–168, 2000.

[58] Yang Song, Jian Huang, Ding Zhou, Hongyuan Zha, and C Lee Giles. Iknn: Informative k-nearest neighbor pattern classification. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 248–264. Springer, 2007.

[59] Eui-Hong Sam Han, George Karypis, and Vipin Kumar. Text categorization using weight adjusted k-nearest neighbor classification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 53–65. Springer, 2001.

[60] Zhou Yong, Li Youwen, and Xia Shixiong. An improved knn text classification algorithm based on clustering. *Journal of Computers*, 4(3):230–237, 2009.

[61] R Muralidharan and C Chandrasekar. Combining local and global feature for object recognition using svm-knn. In *International Conference on Pattern Recognition, Informatics and Medical Engineering*, pages 1–7. IEEE, 2012.

[62] John D Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, 2001.

[63] Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 188–191, 2003.

[64] Avinesh PVS and G Karthik. Part-of-speech tagging and chunking using conditional random fields and transformation based learning. *Shallow Parsing for South Asian Languages*, 21:21–24, 2007.

[65] David DeCaprio, Jade P Vinson, Matthew D Pearson, Philip Montgomery, Matthew Doherty, and James E Galagan. Conrad: gene prediction using conditional random fields. *Genome Research*, 17(9):1389–1398, 2007.

[66] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 200–209. Morgan Kaufmann Publishers Inc., 1999.

[67] Vladimir Vapnik. *The nature of statistical learning theory*. Springer Science & Business Media, 2013.

[68] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International joint Conference on Natural Language Processing*, pages 1681–1691, 2015.

[69] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1556–1566, 2015.

[70] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 207–212, 2016.

[71] Hao Peng, Jianxin Li, Yu He, Yaopeng Liu, Mengjiao Bao, Lihong Wang, Yangqiu Song, and Qiang Yang. Large-scale hierarchical text classification with recursively regularized deep graph-cnn. In *Proceedings of the 2018 World Wide Web Conference*, pages 1063–1072, 2018.

[72] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 1107–1116, 2017.

[73] Hoa T Le, Christophe Cerisara, and Alexandre Denis. Do convolutional networks need to be deep for text classification? In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[74] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[75] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, 2014.

[76] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[77] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2227–2237, 2018.

[78] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[79] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725, 2016.

[80] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[81] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. Universal

sentence encoder for english. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 169–174, 2018.

[82] Andrew M Dai, Christopher Olah, and Quoc V Le. Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998*, 2015.

[83] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[84] Magnus Sahlgren. The distributional hypothesis. *Italian Journal of Disability Studies*, 20:33–53, 2008.

[85] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 655–665, 2014.

[86] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[87] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[88] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, pages 649–657, 2015.

[89] Rie Johnson and Tong Zhang. Deep pyramid convolutional neural networks for text categorization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 562–570, 2017.

[90] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[91] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[92] Kyunghyun Cho, B van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014.

[93] Duyu Tang, Bing Qin, Xiaocheng Feng, and Ting Liu. Effective lstms for target-dependent sentiment classification. In *Proceedings of the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3298–3307, 2016.

[94] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations*, 2015.

[95] Min Yang, Wenting Tu, Jingxuan Wang, Fei Xu, and Xiaojun Chen. Attention based lstm for target dependent sentiment classification. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[96] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.

[97] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2019.

[98] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations*, 2019.

[99] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[100] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, pages 3261–3275, 2019.

[101] Aliaksei Severyn and Alessandro Moschitti. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 373–382, 2015.

[102] Shervin Minaee and Zhu Liu. Automatic question-answering using a deep similarity neural network. In *2017 IEEE Global Conference on Signal and Information Processing*, pages 923–927. IEEE, 2017.

[103] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 3973–3983, 2019.

[104] Samuel Humeau, Kurt Shuster, Marie-Anne Lachaux, and Jason Weston. Poly-encoders: Architectures and pre-training strategies for fast and accurate multi-sentence scoring. In *International Conference on Learning Representations*, 2019.

[105] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3856–3866, 2017.

[106] Min Yang, Wei Zhao, Jianbo Ye, Zeyang Lei, Zhou Zhao, and Soufei Zhang. Investigating capsule networks with dynamic routing for text classification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3110–3119, 2018.

[107] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 2020.

[108] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):9, 2016.

[109] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2009.

[110] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[111] Yang Liu, Zhiyuan Liu, Tat-Seng Chua, and Maosong Sun. Topical word embeddings. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[112] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 2463–2473, 2019.

[113] Malte Ostendorff, Peter Bourgonje, Maria Berger, Julián Moreno-Schneider, Georg Rehm, and Bela Gipp. Enriching bert with knowledge graph embeddings for document classification. *arXiv preprint arXiv:1909.08402*, 2019.

[114] Xing Wu, Tao Zhang, Liangjun Zang, Jizhong Han, and Songlin Hu. Mask and infill: applying masked language model to sentiment transfer. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5271–5277. AAAI Press, 2019.

[115] Alexis Conneau, Ruty Rinott, Guillaume Lample, Adina Williams, Samuel Bowman, Holger Schwenk, and Veselin Stoyanov. Xnli: Evaluating cross-lingual sentence representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2475–2485, 2018.

[116] Akiko Eriguchi, Melvin Johnson, Orhan Firat, Hideto Kazawa, and Wolfgang Macherey. Zero-shot cross-lingual classification using multilingual neural machine translation. *arXiv preprint arXiv:1809.04686*, 2018.

[117] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496, 2019.

[118] Zheng Li, Ying Wei, Yu Zhang, and Qiang Yang. Hierarchical attention transfer network for cross-domain sentiment classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[119] Matthew E Peters, Sebastian Ruder, and Noah A Smith. To tune or not to tune? adapting pretrained representations to diverse tasks. In *Proceedings of the 4th Workshop on Representation Learning for NLP*, pages 7–14, 2019.

[120] Yu Zhang and Qiang Yang. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114*, 2017.

[121] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, pages 506–516, 2017.

[122] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799, 2019.

[123] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1321–1330, 2017.

[124] Radu Herbei and Marten H Wegkamp. Classification with reject option. *The Canadian Journal of Statistics/La Revue Canadienne de Statistique*, pages 709–721, 2006.

[125] Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 4885–4894, 2017.

[126] Giorgio Fumera, Ignazio Pillai, and Fabio Roli. Classification with reject option in text categorisation systems. In *Proceedings of the 12th International Conference on Image Analysis and Processing*, page 582, 2003.

[127] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

[128] Tongfei Chen, Jirí Navrátil, Vijay Iyengar, and Karthikeyan Shanmugam. Confidence scoring using whitebox meta-models with linear classifier probes. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1467–1475. PMLR, 2019.

[129] Vladimir Vovk, Alex Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*. Springer Science & Business Media, 2005.

[130] Glenn Shafer and Vladimir Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(Mar):371–421, 2008.

[131] Jiangming Sun, Lars Carlsson, Ernst Ahlberg, Ulf Norinder, Ola Engkvist, and Hongming Chen. Applying mondrian cross-conformal prediction to estimate prediction confidence on large imbalanced bioactivity data sets. *Journal of Chemical Information and Modeling*, 57(7):1591–1598, 2017.

[132] Isidro Cortes-Ciriano and Andreas Bender. Reliable prediction errors for deep neural networks using test-time dropout. *Journal of Chemical Information and Modeling*, 59(7):3330–3339, 2019.

[133] Paolo Toccaceli and Alexander Gammerman. Combination of inductive mondrian conformal predictors. *Machine Learning*, 108(3):489–510, 2019.

[134] Dallas Card, Michael Zhang, and Noah A Smith. Deep weighted averaging classifiers. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 369–378, 2019.

[135] Bianca Zadrozny and Charles Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *Proceedings of the 18th International Conference on Machine Learning*, volume 1, pages 609–616. Citeseer, 2001.

[136] Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 694–699, 2002.

[137] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large Margin Classifiers*, 10(3):61–74, 1999.

[138] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6405–6416, 2017.

[139] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059, 2016.

[140] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[141] Hao Wang, Doğan Can, Abe Kazemzadeh, François Bar, and Shrikanth Narayanan. A system for real-time twitter sentiment analysis of 2012 us presidential election cycle. In *Proceedings of the ACL 2012 System Demonstrations*, pages 115–120, 2012.

[142] Carole Cadwalladr and Emma Graham-Harrison. Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach. *The Guardian*, 17:22, 2018.

[143] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, 2013.

[144] Rui Xia, Chengqing Zong, and Shoushan Li. Ensemble of feature sets and classification algorithms for sentiment classification. *Information Sciences*, 181(6):1138–1152, 2011.

[145] Manish Munikar, Sushil Shakya, and Aakash Shrestha. Fine-grained sentiment classification using bert. In *2019 Artificial Intelligence for Transforming Business and Society*, volume 1, pages 1–5. IEEE, 2019.

[146] Hu Xu, Bing Liu, Lei Shu, and Philip Yu. Bert post-training for review reading comprehension and aspect-based sentiment analysis. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 1, 2019.

[147] Xinjie Zhou, Xiaojun Wan, and Jianguo Xiao. Attention-based lstm network for cross-lingual sentiment classification. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 247–256, 2016.

[148] Lijuan Zheng, Hongwei Wang, and Song Gao. Sentimental feature selection for sentiment analysis of chinese online reviews. *International Journal of Machine Learning and Cybernetics*, 9(1):75–84, 2018.

[149] Muhammad Abdul-Mageed, Mona Diab, and Sandra Kübler. Samar: Subjectivity and sentiment analysis for arabic social media. *Computer Speech & Language*, 28(1):20–37, 2014.

[150] Julian Brooke, Milan Tofiloski, and Maite Taboada. Cross-linguistic sentiment analysis: From english to spanish. In *Proceedings of the International Conference RANLP-2009*, pages 50–54, 2009.

[151] Igor Mozetič, Miha Grčar, and Jasmina Smailović. Multilingual Twitter sentiment classification: The role of human annotators. *PloS one*, 11(5):e0155036, 2016.

[152] Kerstin Denecke. Using SentiWordNet for multilingual sentiment analysis. In *International Conference on Data Engineering Workshop*, pages 507–512. IEEE, 2008.

[153] Yanlin Feng and Xiaojun Wan. Towards a unified end-to-end approach for fully unsupervised cross-lingual sentiment analysis. In *Proceedings of the 23rd Conference on Computational Natural Language Learning*, pages 1035–1044, 2019.

[154] Nicolas Garneau, Mathieu Godbout, David Beauchemin, Audrey Durand, and Luc Lamontagne. A robust self-learning method for fully unsupervised cross-lingual mappings of word embeddings: Making the method robustly reproducible as well. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 5546–5554, 2020.

[155] Maria Pontiki, Dimitrios Galanis, Haris Papageorgiou, Ion Androutsopoulos, Suresh Manandhar, Mohammad Al-Smadi, Mahmoud Al-Ayyoub, Yanyan Zhao, Bing Qin, Orphée De Clercq, et al. Semeval-2016 task 5: Aspect based sentiment analysis. In *10th International Workshop on Semantic Evaluation*, 2016.

[156] Chi Sun, Luyao Huang, and Xipeng Qiu. Utilizing bert for aspect-based sentiment analysis via constructing auxiliary sentence. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 380–385, 2019.

[157] Maja Pantic, Nicu Sebe, Jeffrey F Cohn, and Thomas Huang. Affective multimodal human-computer interaction. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pages 669–676, 2005.

[158] Erik Cambria. Affective computing and sentiment analysis. *IEEE intelligent systems*, 31(2):102–107, 2016.

[159] Erik Cambria and Amir Hussain. Sentic computing. *marketing*, 59(2):557–577, 2012.

[160] Push Singh. The open mind common sense project. *KurzweilAI. net*, 2002.

[161] Carlo Strapparava, Alessandro Valitutti, et al. Wordnet affect: an affective extension of wordnet. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation*, volume 4, page 40. Citeseer, 2004.

[162] Federica Bisio, Claudia Meda, Paolo Gastaldo, Rodolfo Zunino, and Erik Cambria. Concept-level sentiment analysis with senticnet. In *A Practical Guide to Sentiment Analysis*, pages 173–188. Springer, 2017.

[163] Antonio Reyes, Paolo Rosso, and Tony Veale. A multidimensional approach for detecting irony in twitter. *Language Resources and Evaluation*, 47(1):239–268, 2013.

[164] Francesco Barbieri, Horacio Saggion, and Francesco Ronzano. Modelling sarcasm in twitter, a novel approach. In *Proceedings of the 5th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 50–58, 2014.

[165] Tomáš Ptáček, Ivan Habernal, and Jun Hong. Sarcasm detection on czech and english twitter. In *Proceedings of the 25th International Conference on Computational Linguistics*, pages 213–223, 2014.

[166] Saif M Mohammad, Xiaodan Zhu, Svetlana Kiritchenko, and Joel Martin. Sentiment, emotion, purpose, and style in electoral tweets. *Information Processing & Management*, 51(4):480–499, 2015.

[167] Cynthia Van Hee, Els Lefever, and Véronique Hoste. Semeval-2018 task 3: Irony detection in english tweets. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 39–50, 2018.

[168] Diana Maynard and Mark A Greenwood. Who cares about sarcastic tweets? investigating the impact of sarcasm on sentiment analysis. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation*, pages 4238–4243, 2014.

[169] Emilio Sulis, Delia Irazú Hernández Farías, Paolo Rosso, Viviana Patti, and Giancarlo Ruffo. Figurative messages and affect in twitter: Differences between# irony,# sarcasm and# not. *Knowledge-Based Systems*, 108:132–143, 2016.

[170] Florian Kunneman, Christine Liebrecht, Margot Van Mulken, and Antal Van den Bosch. Signaling sarcasm: From hyperbole to hashtag. *Information Processing & Management*, 51(4):500–509, 2015.

[171] P. Rosso F. D. I. Hernańdez, V. Patti. Irony detection in twitter: The role of affective content. *ACM Transactions on Internet Technology*, 16(3):19, 2016.

[172] Soujanya Poria, Erik Cambria, Devamanyu Hazarika, and Prateek Vij. A deeper look into sarcastic tweets using deep convolutional neural networks. In *Proceedings of the 26th International Conference on Computational Linguistics*, pages 1601–1612, 2016.

[173] Aniruddha Ghosh and Tony Veale. Fracking sarcasm using neural network. In *Proceedings of the 7th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 161–169, 2016.

[174] Aniruddha Ghosh and Tony Veale. Magnets for sarcasm: Making sarcasm detection timely, contextual and very personal. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 482–491, 2017.

[175] Shereen Oraby, Vrindavan Harrison, Amita Misra, Ellen Riloff, and Marilyn Walker. Are you serious?: Rhetorical questions and sarcasm in social media dialog. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 310–319, 2017.

[176] Yi Tay, Luu Anh Tuan, Siu Cheung Hui, and Jian Su. Reasoning with sarcasm by reading in-between. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 2018.

[177] Njagi Dennis Gitari, Zhang Zuping, Hanyurwimfura Damien, and Jun Long. A lexicon-based approach for hate speech detection. *International Journal of Multimedia and Ubiquitous Engineering*, 10(4):215–230, 2015.

[178] Chikashi Nobata, Joel Tetreault, Achint Thomas, Yashar Mehdad, and Yi Chang. Abusive language detection in online user content. In *Proceedings of the 25th International Conference on World Wide Web*, pages 145–153, 2016.

[179] Pinkesh Badjatiya, Shashank Gupta, Manish Gupta, and Vasudeva Varma. Deep learning for hate speech detection in tweets. pages 759–760, 2017.

[180] Nitin Jindal and Bing Liu. Review spam detection. In *Proceedings of the 16th International Conference on World Wide Web*, pages 1189–1190, 2007.

[181] Yafeng Ren and Donghong Ji. Neural networks for deceptive opinion spam detection: An empirical study. *Information Sciences*, 385:213–224, 2017.

[182] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 168–177, 2004.

[183] Preslav Nakov, Doris Hoogeveen, Lluís Màrquez, Alessandro Moschitti, Hamdy Mubarak, Timothy Baldwin, and Karin Verspoor. Semeval-2017 task 3: Community question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation*, pages 27–48, 2017.

[184] Ryan Lowe, Nissan Pow, Iulian Vlad Serban, and Joelle Pineau. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 285–294, 2015.

[185] Sascha Narr, Michael Hulfenhaus, and Sahin Albayrak. Language-independent Twitter sentiment analysis. *Knowledge Discovery and Machine Learning*, pages 12–14, 2012.

[186] Nuria Bel, Cornelis HA Koster, and Marta Villegas. Cross-lingual text categorization. In *International Conference on Theory and Practice of Digital Libraries*, pages 126–139. Springer, 2003.

[187] Alexis Conneau, Holger Schwenk, L Barrault, and Y Lecun. Very deep convolutional networks for text classification. In *European Chapter of the Association for Computational Linguistics*, 2017.

[188] Jason Lee, Kyunghyun Cho, and Thomas Hofmann. Fully character-level neural machine translation without explicit segmentation. *Transactions of the Association for Computational Linguistics*, 2017.

[189] Joonatas Wehrmann, Willian Becker, Henry EL Cagnini, and Rodrigo C Barros. A character-based convolutional neural network for language-agnostic twitter sentiment analysis. In *International Joint Conference on Neural Networks*, pages 2384–2391. IEEE, 2017.

[190] Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. Multilingual language processing from bytes. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1296–1306, 2016.

[191] Zhilin Yang, Ruslan Salakhutdinov, and William Cohen. Multi-task cross-lingual sequence tagging from scratch. *arXiv preprint arXiv:1603.06270*, 2016.

[192] Alexandra Olteanu, Sarah Vieweg, and Carlos Castillo. What to expect when the unexpected happens: Social media communications across crises. In *18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 994–1009. ACM, 2015.

[193] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1019–1027, 2016.

[194] Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.

[195] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations*, 2015.

[196] Zhilin Yang, Bhuwan Dhingra, Ye Yuan, Junjie Hu, William W Cohen, and Ruslan Salakhutdinov. Words or characters? fine-grained gating for reading comprehension. *arXiv preprint arXiv:1611.01724*, 2016.

[197] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

[198] Francesco Barbieri, Valerio Basile, Danilo Croce, Malvina Nissim, Nicole Novielli, Viviana Patti, et al. Overview of the evalita 2016 sentiment polarity classification task. In *3rd Italian Conference on Computational Linguistics, CLiC-it 2016 and 5th Evaluation Campaign of Natural Language Processing and Speech Tools for Italian*, volume 1749, pages 1–11. CEUR-WS, 2016.

[199] Richard J Gerrig and Yevgeniya Goldvarg. Additive effects in the perception of sarcasm: Situational disparity and echoic mention. *Metaphor and Symbol*, 15(4):197–208, 2000.

[200] Stacey L Ivanko and Penny M Pexman. Context incongruity and irony processing. *Discourse Processes*, 35(3):241–279, 2003.

[201] Aditya Joshi, Pushpak Bhattacharyya, and Mark J Carman. Automatic sarcasm detection: A survey. *ACM Computing Surveys (CSUR)*, 50(5):73, 2017.

[202] Byron C Wallace. Computational irony: A survey and new perspectives. *Artificial Intelligence Review*, 43(4):467–483, 2015.

[203] Aditya Joshi, Vaibhav Tripathi, Kevin Patel, Pushpak Bhattacharyya, and Mark Carman. Are word embedding-based features useful for sarcasm detection? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1006–1011, 2016.

[204] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural*, page 347, 2005.

[205] Petra Kralj Novak, Jasmina Smailović, Borut Sluban, and Igor Mozetič. Sentiment of emojis. *PloS one*, 10(12):e0144296, 2015.

[206] Debora Nozza, Elisabetta Fersini, and Enza Messina. Unsupervised irony detection: A probabilistic model with word embeddings. In *International Conference on Knowledge Discovery and Information Retrieval*, volume 2, pages 68–76. SCITEPRESS, 2016.

[207] Rui Xia, Feng Xu, Jianfei Yu, Yong Qi, and Erik Cambria. Polarity shift detection, elimination and ensemble: A three-stage model for document-level sentiment analysis. *Information Processing & Management*, 52(1):36–45, 2016.

[208] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 17–36, 2012.

[209] Ziqiang Cao, Wenjie Li, Sujian Li, and Furu Wei. Improving multi-document summarization via text classification. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 3053–3059, 2017.

[210] Xiangbo Shu, Guo-Jun Qi, Jinhui Tang, and Jingdong Wang. Weakly-shared deep transfer networks for heterogeneous-domain knowledge propagation. In *Proceedings of the 23rd ACM International Conference on Multimedia*, pages 35–44. ACM, 2015.

[211] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. Sentiwordnet 3.0: an enhanced lexical resource for sentiment analysis and opinion mining. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation*, volume 10, pages 2200–2204, 2010.

[212] CJ Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of Eighth International AAAI Conference on Weblogs and Social Media*, 2014.

[213] Sara Rosenthal, Noura Farra, and Preslav Nakov. Semeval-2017 task 4: Sentiment analysis in twitter. In *Proceedings of the 11th International Workshop on Semantic Evaluation*, pages 502–518, 2017.

[214] Aniruddha Ghosh, Guofu Li, Tony Veale, Paolo Rosso, Ekaterina Shutova, John Barnden, and Antonio Reyes. Semeval-2015 task 11: Sentiment analysis of figurative language in twitter. In *Proceedings of the 9th International Workshop on Semantic Evaluation*, pages 470–478, 2015.

[215] Tomáš Mikolov, Édouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation*, 2018.

[216] Mengting Wan and Julian McAuley. Modeling ambiguity, subjectivity, and diverging viewpoints in opinion question answering systems. In *2016 IEEE 16th International Conference on Data Mining*, pages 489–498. IEEE, 2016.

[217] Shen Gao, Zhaochun Ren, Yihong Zhao, Dongyan Zhao, Dawei Yin, and Rui Yan. Product-aware answer generation in e-commerce question-answering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 429–437. ACM, 2019.

[218] Shiqian Chen, Chenliang Li, Feng Ji, Wei Zhou, and Haiqing Chen. Driven answer generation for product-related questions in e-commerce. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 411–419. ACM, 2019.

[219] Qian Yu, Wai Lam, and Zihao Wang. Responding e-commerce product questions via exploiting qa collections and reviews. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2192–2203, 2018.

[220] Qian Yu and Wai Lam. Aware answer prediction for product-related questions incorporating aspects. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 691–699. ACM, 2018.

[221] Hu Xu, Sihong Xie, Lei Shu, and S Yu Philip. Dual attention network for product compatibility and function satisfiability analysis. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[222] Matthew Dunn, Levent Sagun, Mike Higgins, V Ugur Guney, Volkan Cirik, and Kyunghyun Cho. Searchqa: A new q&a dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179*, 2017.

[223] Bhuwan Dhingra, Kathryn Mazaitis, and William W Cohen. Quasar: Datasets for question answering by search and reading. *arXiv preprint arXiv:1707.03904*, 2017.

[224] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, 2016.

[225] Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. Newsqa: A machine comprehension dataset. pages 191–200, 2017.

[226] Wei Wu, SUN Xu, and WANG Houfeng. Question condensing networks for answer selection in community question answering. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 1746–1755, 2018.

[227] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701, 2015.

[228] Xiaodong Zhang, Sujian Li, Lei Sha, and Houfeng Wang. Attentive interactive neural networks for answer selection in community question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[229] Ivan Srba and Maria Bielikova. A comprehensive survey and classification of approaches for community question answering. *ACM Transactions on the Web (TWEB)*, 10(3):1–63, 2016.

[230] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, Geoffrey E Hinton, et al. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[231] Armand Joulin, Edouard Grave, and Piotr Bojanowski Tomas Mikolov. Bag of tricks for efficient text classification. pages 427–431, 2017.

[232] Jie Zhao, Ziyu Guan, and Huan Sun. Riker: Mining rich keyword representations for interpretable product question answering. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1389–1398, 2019.

[233] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 784–789, 2018.

[234] A Gammerman. Conformal predictors for reliable pattern recognition. In *Computer Data Analysis and Modeling: Stochastics and Data Science*, pages 33–36, 2019.

[235] Lars Carlsson, Claus Bendtsen, and Ernst Ahlberg. Comparing performance of different inductive and transductive conformal predictors relevant to drug discovery. In *Conformal and Probabilistic Prediction and Applications*, pages 201–212, 2017.

[236] Mansi Gupta, Nitish Kulkarni, Raghuveer Chanda, Anirudha Rayasam, and Zachary C Lipton. Amazonqa: a review-based question answering task. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 4996–5002. AAAI Press, 2019.

[237] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1601–1611, 2017.

[238] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[239] Fei Liu, Alistair Moffat, Timothy Baldwin, and Xiuzhen Zhang. Quit while ahead: Evaluating truncated rankings. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 953–956, 2016.

[240] Miroslav Kubat, Robert C Holte, and Stan Matwin. Machine learning for the detection of oil spills in satellite radar images. *Machine Learning*, 30(2-3):195–215, 1998.

[241] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152. IEEE, 2012.

[242] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, 2018.

[243] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, 2020.