University of Sheffield

Irony Detection in English Tweets

Liya Kuncheva

Supervisor: Nikolaos Aletras

This report is submitted in partial fulfilment of the requirement
for the degree of MSc in Advanced Computer Science

in the

Department of Computer Science

May 10, 2023

Declaration

Name: Liya Nikolova Kuncheva

Signature:

Date: 05/12/2022

Abstract

Satirical irony is a very important language phenomena. Its recognition is of great importance to sentiment analysis. However, researches on this topic are still quite rare and existing studies have problems such as unclear definition and unclear objects of study. This project focuses on defining the problem, researching existing methods suitable for tackling the task, and conducting an experimental stage to test what approach produces the best results. The final goal of this project is to find the most efficient way of detecting irony in English tweets while considering existing researched methods.

Contents

Chapter 1

Introduction

With the rapid development of technology throughout the years, people tend to increasingly rely on the social web as a source of information, communication, and advertisement. It has quickly become one of the easiest way to influence people, their beliefs and actions. Due to the not extremely strict speech regulations, opinions and thoughts can be easily expressed and shared with the world in the matter of seconds. This could be highly educational, helpful, and entertaining, however, it also enables people to abuse it with the intentions of spreading misleading, false, or harmful information and content. This is where natural language processing comes to help. It has many important implications such as regulating the use of hate speech, offensive content, and false information, classifying text, opinion mining and sentiment analysis. However, the increasing use and development of technology and the social web has stimulated the use of more creative and figurative language such as irony and sarcasm, which has made the natural language processing tasks more complex and hard to implement. We are still exploring the fields of sentiment analysis and opinion mining when internet slang, coded messages, irony, and sarcasm are present. Detecting and modelling those would be highly beneficial and could have applications in various research fields.

One example that indicates the need for an irony and sarcasm detector that got attention recently is when the comedian Ryan Hand posted a tweet saying *"What a disgrace, there's a woman crying at the @Ryanair check in desk who's been made to pay more for emotional baggage."* Ryanair mistook this for an actual complaint and responded with *"Hi Ryan, which airport is this happening at? IK"*. Thousands of retweets later, Ryanair realised the nature of the tweet and posted a follow-up tweet: *"We apologize for temporary technical difficulties with our sarcasm detector."*

Even if irony is used mostly in a humorous manner, it could be interpreted in the wrong way, leading to unwanted consequences. The following example reveals the security reasons for the implementation of an irony detector. In 2015, Lithuania gave Russia zero points for its performance in the Eurovision Song Contest, which lead to the *"And 12 bombers depart now ... to Lithuania!"* tweet being posted by the *dailykisev* account. If taken seriously, this could have been interpreted as a national threat and could have lead to devastating consequences.

It has been made clear that the ability to detect irony has become increasingly important in the digital age as the prevalence of online communication has grown.

1.1 Aims and Objectives

The aim for this project is to explore the suitable techniques for tackling the problem of irony detection and creating a system that successfully detects it in English tweets. Various methods will be observed and tested with the goal of finding the most reliable and highest scoring one amongst them. This includes considering and researching different ideas for every step of our implementation, namely, handling the data sets, implementing pre-processing tasks, extracting features with high information value, using a suitable classification algorithm, and evaluating the performance of the system. After a sufficient approach is selected, the classifier will be trained with a given set of labelled data and is expected to be able to classify and label a given set of unlabelled data. The system performance and accuracy are then going to be tested and evaluated.

1.2 Overview of the Report

This report focuses on the research concerning irony detection as a Natural Language Processing task and its development. In the next chapter, a literature survey is conducted and that explains all the relevant work related to irony detection.

In the Literature Survey part, various data sets, pre-processing, classifying, and evaluation methods are researched with extensive background review of existing approaches.

In the Requirements and Analysis section, the experimental setup of the irony detection approach is described with the selected course of action for the implementation step.

The design chapter focuses on explaining what methods and design techniques are chosen and why, while the implementation and testing chapter provides with a run-through of the program architecture.

The most significant experimental and final results are discussed in the results section, along with program flaws, missed out functionality, and further work proposed.

Chapter 2

Literature Survey

Irony has been studied extensively in the fields of linguistics, psychology, and NLP. In the context of NLP, irony is typically characterized by a discrepancy between literal and intended meanings that arise from sarcasm, ridicule, or other forms of humorous expression ([1] Reyes et al., 2019). It has been suggested that automatic irony detection could improve sentiment analysis by helping machines to better understand the nuances in language used on social media ([1] Reyes et al., 2019; [2] Ruch et al., 2018). However, automatic irony detection is still an open problem due to the difficulty of automatically recognizing subtle variations in language use.

2.1 Defining irony

Irony is a widely used linguistic device that creates a gap between the literal interpretation of an expression and its intended meaning. It has been studied in linguistics, psychology, and cognitive science ([3] Gibbs and Colston, 2007; [4] Gibbs, 1986; [5] Kreuz and Caucci, 2007; [6] Kreuz and Glucksberg, 1989; [7] Utsumi, 2000), yet, no clear definition seems to exist ([8] Filatova, 2012). Even though the definition is broad and vague, irony can be divided into two main categories: situational and verbal irony. Situational irony does not rely on the use of spoken or written expressions as much as a discrepancy (often humorous) in the meanings of a situation or an act. For instance, a fire station burning down, a marriage counselor files for divorce, an English teacher has poor grammar, etc. Verbal irony, however, refers to "meaning the opposite of what was literally said". Due to the complex nature of the former type, only verbal irony is considered in this project.

2.2 The problem of irony detection

Irony detection as a sentiment analysis task has been proven to be very tricky and is hardly explored yet. In an in-person and verbal situation, humans use the power of facial expressions, intonation, and other social cues that make detecting irony an easier task than doing so reading a written text. Additionally, when incorporating irony or sarcasm in a post or a tweet, people have more time to think about the wording and structure of their expressions, thus creating an even more complex and sophisticated form of irony.

Moreover, when observing a piece of written text, humans, as opposing to computers, are usually able to identify whether it is ironic or not within seconds. This is due to one of the most powerful tools we use that is difficult to implement on computer systems. This is the ability to understand context and having extensive world knowledge. We can detect the polarity of a given text, its subjectivity or objectivity, which is not as simple as it seems for computers to do. Our task here is to try and develop an algorithm to detect and understand those concepts, which will require putting great efforts in training, pre-processing, classifying, testing, and evaluating the system.

The main challenge will be the extracting of relevant data from a given data set and using it in the right way. The most implemented approach for this is using opinion quintuples, which provide the basic information for an opinion summary. Such a summary is called an aspect-based summary (or feature-based summary) and was proposed in Hu and Liu (2004)[9] and [10]Liu et al. (2015).

Such a quintuple consists of a target object, a feature of said object, an opinion holder, the time that the opinion was expressed, and a sentiment value of the opinion. Structuring the data this way helps with the processing and classifying of the wanted data set.

2.3 Irony detection in Natural Language Processing

Text or document classification is an important NLP problem that has been widely researched in the past decade. It uses a variety of methods and algorithms of machine learning and computational linguistics to sort and categorise a given data set in its according types/classes. This shows how irony detection falls under the task of natural language processing. Some initial work has been done in [11]González-Ibáñez et al. (2011) and [12]Tsur et al. (2009) much like many other researches have done and are continuing to work on.

As mentioned before, one of the main issues that we come across when tackling this task is providing context and world knowledge to the computer. It is needed to understand the nature of the piece, its sentiment, and the intent. The problem is that there are hardly any lexical markers that are not context dependent and can work as a mold for irony detection in every case. It is apparent that context and world knowledge play an incredibly important part in this endeavour. A number of NLP tasks can be applied to potentially achieve this goal. This includes applying pre-processing methods to the given data set such as parsing, word segmentation, part-of-speech (POS) tagging, stop-words removal, and stemming, and applying a trained classifier for sentiment analysis and text classification.

Text classification has broad applications such as sentiment analysis, topic labeling, spam detection, and intent detection. If a irony/satire detection system proves to be successful, other text classification applications may benefit from its success.

2.4 Approaches

Researchers have proposed various approaches for identifying irony on social media platforms such as Twitter. Most studies have focused on using supervised machine learning algorithms for detecting irony in tweets ([13] Reyes et al., 2012). These studies typically involve training a classifier using manually labeled data sets. Other studies have used unsupervised methods such as clustering and topic modeling ([14] Laksana et al., 2017). These methods involve building topic models from collections of tweets to detect patterns in language use indicative of ironic speech.

In addition to the research conducted on detecting irony in general, there have also been several studies focused specifically on detecting irony in English tweets. For example, [10] Li et al. (2019) developed an algorithm based on recurrent neural networks that uses contextual information from user profiles to detect irony in English tweets with high accuracy. Similarly, [15] Zare et al. (2020) proposed a hybrid approach combining deep learning and rule-based methods for detecting sarcasm and irony in English tweets with improved performance over other methods.

2.4.1 Lexicon-based

A Lexicon Based approach uses a dictionary of words (lexica) with associated values for different opinion/emotion words like: good, bad, nice, terrible etc. For subjective sentences the number of positive and negative words is counted and used to determine the polarity of the whole sentence. This analysis can be done at various levels i.e. on a sentence or an entire document or in relation to a particular feature of the subject of the statement. [16] Liu and Zhang (2012)

The bag-of-words model is a good fit for sentiment analysis to some extent, however, since lexicon-based classification is less effective for short documents, such as single-sentence reviews or social media posts, linguistic issues like negation and irrealis - events that are hypothetical or otherwise non-factual, can make bag-of-words classification ineffective ([17] Polanyi and Zaenen, 2006).

An improved approach would be to replace the bag-of-words model with a bag-of-bigrams one where each base feature is a pair of adjacent words. Bigrams can handle relatively straightforward cases, such as when an adjective is immediately negated. This would slightly improve the performance of our classification system, however, it is still not ideal. We could take this further by using trigrams, but this might over-complicate the algorithm since it works better with larger text corpora, such as articles or longer documents.

No matter what approach we take, the main idea is still the same - we aim to mark a given piece of text by its polarity. There are two variations using this method:

BINARY LEXICON:

With a binary lexicon, the scores can either be -1 for negative statements, 0 for neutral statements or +1 for positive statements. It is calculated by classifying the words in the text in two groups: positive and negative. If more positive words or phrases are present, the text is classified as positive, while more negative ones indicate a negative statement, and if they are equal, the text is classified as neutral.

GRADABLE LEXICON:

In a gradable lexicon approach, each descriptive word and its intensiers or diminishers are assigned a numerical value with negative words being assigned negative integers and positive words being assigned positive integers. These numbers are then added up to determine the overall emotional weight of the statements. Intensifiers are words that increase the weights of the adjective in the meaning of a sentence. With this method, the scores can vary and are not limited to the 3 values seen in the binary lexicon.

It is apparent that lexicon-based sentiment analysis avoids machine learning altogether, and classifies documents by counting words against positive and negative sentiment word lists ([18] Taboada et al., 2011).

More sophisticated solutions try to account for the syntactic structure of the sentence ([19] Wilson et al., 2005; [20] Socher et al., 2013), or apply more complex classifiers such as convolutional neural networks ([21] Kim, 2014)

2.4.2 Machine learning

Machine learning is the science of how computers learn without being explicitly programmed. It developed by incorporating computational skills and Math and Statistics knowledge. Such approach to sentiment analysis can be divided into three categories: **supervised**, **unsupervised** and **semi-supervised**. Supervised methods use a large amount of labelled training data. Unsupervised learning is carried out without the use of labelled training data (k-means clustering, expectation-maximization). Semi-supervised learning uses a small amount of labelled data and a large amount of unlabelled data for training.

Classification algorithms are intended to identify which categories an object belongs to. There are several machine learning algorithms used to carry out classification. In text classification, most the commonly seen methods are Decision Trees (DT), Logistic Regression (LR), Support Vector Machines (SVM), Nave Bayes (NB), and Maximum Entropy (ME). Linear Classifiers are argued to be achieving the highest scores. Support Vector Machines are a commonly used approach, however, Neural Networks has gained the attention of researchers in the past few years. This includes Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), Transformers Networks, and LSTM and Bi-LSTM.

A number of approaches have been proposed for detecting irony in written language. Early studies focused on handcrafted features such as sentiment analysis or part-of-speech tagging ([22] Bamman et al., 2017). More recently, deep learning approaches such as convolutional neural networks (CNNs) have been used to detect irony ([23] Ling et al., 2017). CNNs have the advantage of being able to capture contextual information from text by learning embeddings from large corpora ([24] Kim et al., 2017). Additionally, transformer networks have been explored as a possible solution to the problem of irony detection ([25] Devlin et al., 2018). Transformer networks are particularly well suited to this task because they are able to capture long-term dependencies between words in text ([26] Vaswani et al., 2017).

One study conducted by [27] Bilal and Atif (2017) examined the use of machine learning techniques to detect irony in English tweets. They utilized a convolutional neural network (CNN) to classify tweets as either ironic or non-ironic. The results of their experiment showed that the CNN was able to accurately classify tweets with an accuracy of 96.4%. Deep learning models such as recurrent neural networks (RNNs) have been shown to outperform traditional machine learning algorithms when dealing with large amounts of unlabeled data due to their ability to learn complex representations from data ([28] Hassan et al., 2017; [29] Attardo et al., 2003). [30] Zhang et al.(2020) also proposed an unsupervised method based on deep learning for detecting sarcastic comments from online discussion forums such as Reddit. They evaluated their model using a manually labeled data set consisting of 2000 comments and achieved an accuracy of 87%.

Transformer networks have become increasingly popular for natural language processing tasks such as sentiment analysis and machine translation. These networks consist of encoder and decoder components which learn representations from input text using self-attention mechanisms. They have been shown to outperform traditional approaches when it comes to capturing long-term dependencies between words in text ([26] Vaswani et al., 2017). Recently, transformer networks have also been applied to the task of detecting irony in English tweets with promising results ([25] Devlin et al., 2018). In their study, Devlin found that a bidirectional encoder representation from transformers achieved an accuracy of 86% on the SemEval-2018 Task 3 data set when compared with other methods such as CNNs and LSTMs.

While deep learning models have shown promising results in detecting sarcasm in text, they also require large amounts of labeled training data that may not always be available or easy to obtain. As such, traditional machine learning algorithms are still seen as more reliable options for sarcasm detection tasks due to their ability to work with smaller data sets while still achieving good performance levels ([28] Hassan et al., 2017).

2.4.3 Hybrid

It is apparent that both lexicon and machine learning based approaches have their pros and cons. The optimal solution would be to combine them both so that we can utilise the strengths of each approach in one. This is the reason why the hybrid approach was invented. It employs the lexicon-based approach for sentiment scoring followed by training a classifier to assign polarity to the entities in the newly found reviews. This implementation generally achieves high accuracy and since it achieves the best of both techniques, it is most widely used in sentiment analysis tasks.

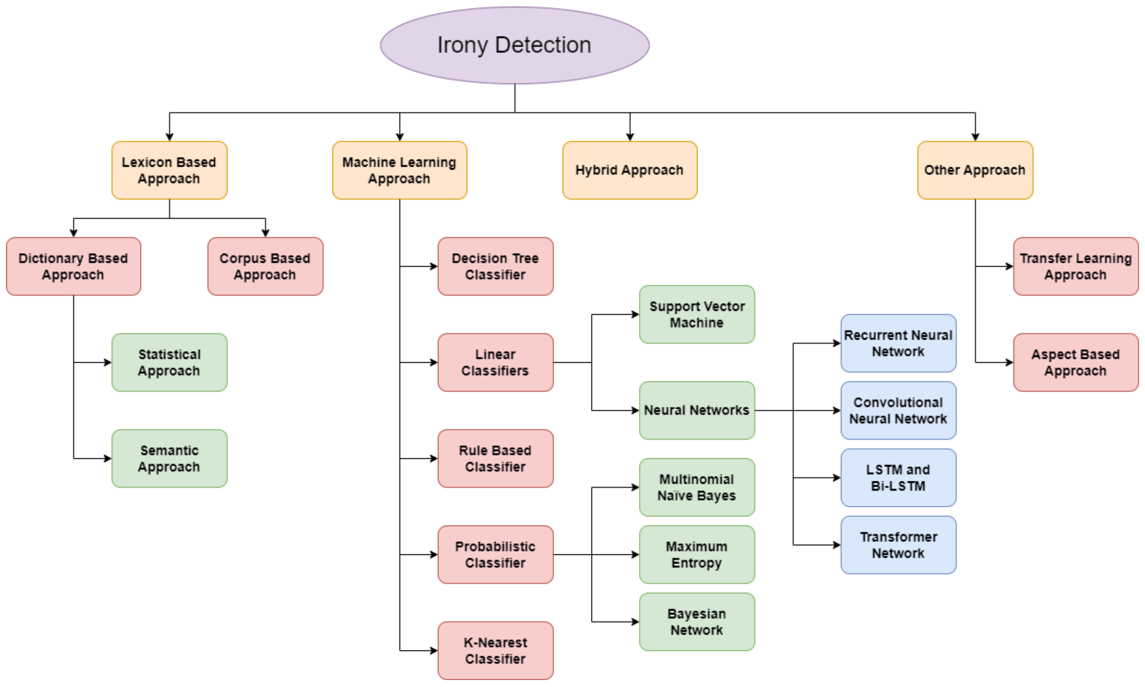

Figure 2.1: Different Approaches for Irony Detection

2.5 Steps

2.5.1 Data

To classify text, real samples of data need to be collected in order to analyse them. The traditional approach to data set collection in classification tasks is to have a training data set and a test data set. Each of the training samples will be labelled with a class value drawn from a set of class values. The training data will be supplied to a classifier algorithm to build a learning model that will be used to predict the class labels of test data instances.

Several data sets have been created to enable researchers to train and evaluate their models. For example, [1] Reyes et al. (2019) created a data set consisting of 7200 tweets annotated with labels indicating whether a tweet was ironic or not. They trained a Naive Bayes classifier on this data set and achieved an accuracy of 78%. Similarly, [2] Ruch et al. (2018) created a data set consisting of 1350 tweets annotated with labels indicating whether they were sarcastic or not. They trained a Support Vector Machine classifier on this data set and achieved an accuracy of 81%. While these results demonstrate promising progress towards automatic irony detection, there is still room for improvement.

Twitter

This project aims to detect irony specifically from the social media platform called *Twitter*. It is an is an online news and social networking site where people communicate in short messages called tweets [31]. It is possible to post images and videos with short captions, however, for the purpose of this project, we are only going to be focusing on text tweets. Possible functionality of the posts include mentioning other users by adding the @ character in front of their username, using hashtags ”#” as key words tagger, which enables users to browse for tweets containing specific hashtags, helping them to narrow down posts by the topic of their contents. This is an important feature of the platform, since there are an average of 500 million tweets per day, making filtering desired content a more manageable task.

2.5.2 Pre-Processing

The data collected for classification purposes are usually unstructured, incomplete, inconsistent and noisy so their representation often need to be re-structured and organised prior to classification and evaluation tasks to enable faster data analysis and accurate data models to avoid misleading results. This is where pre-processing comes to help.

The term pre-processing means manipulating the original data set to represent and organise it into a simplified version for convenient and fast data mining and analysis tasks. Pre-processing is an essential step when dealing with natural language processing tasks such as detecting irony in English tweets for computer science applications. There are several pre-processing techniques that can be used to clean and prepare data for further analysis. For example, tokenization involves splitting a sentence into individual words or phrases that can then be used as features for machine learning algorithms ([32] Rehurek & Sojka, 2010). Sentiment analysis is another common technique used to analyze text data and identify sentiment polarity (positive or negative) ([33] Liu, 2012). Other pre-processing techniques include feature extraction, feature weighting, Part-Of-Speech (POS) tagging, sentence segmentation, stopword removal and stemming/ lemmatization which can be used to reduce noise from the data ([34] McDonald & Krieger 2016).

2.5.3 Classification

After data pre-processing is completed, the next step is to classify the data. Classification is a supervised machine learning task. At this stage, the classifier algorithmically computes the correlation of features of the training data instances to their corresponding class labels and tries to predict the class label of new and unseen (test) instances that it belongs to, given a set of class labels.

As mentioned before, supervised learning algorithms are commonly used for classifying text data into different categories such as sentiment or emotion ([35] Kumar et al., 2018). Support Vector Machines (SVMs), Decision Trees, Naive Bayes Classifiers, and Logistic Regression are all examples of supervised learning algorithms that can be used for text classification tasks ([36] Manning et al., 2008). Unsupervised clustering methods are also commonly used for text classification tasks such as detecting irony in English tweets ([37] Gunnemann et al., 2017). These methods allow data points to be grouped according to similarity without any prior knowledge about the data points themselves. K-means clustering is one example of an unsupervised clustering method that has been successfully applied to text classification tasks ([38] Oliveira & Laender 2015).

2.5.4 Evaluation

The final step that needs to be implemented is the evaluation. It is used to measure the efficiency of a classification system. Investigating the performance of the system will define how effective the system is and allow tuning of the system to achieve the highest possible effectiveness. This will also aid in estimation of how well the system will perform when a different data set is used.([39] Lewis et al. 1995)

There are various evaluation measures used in NLP applications, but the common metrics used specifically in text classification, are classification accuracy, precision, recall and F-measure.

2.6 Summary

This chapter introduces the field of Sentiment Analysis and Irony Detection and reviews existing research on the topic. It describes the methods and techniques that are used to determine whether a given statement is ironic or not. Various pre-processing, classifying, and evaluating methods are highlighted, described and related to the process of sentiment analysis.

Chapter 3

Requirements and Analysis

The previous literature review suggests that both machine learning techniques and linguistic features can be used to detect irony in English tweets. In order to evaluate which method is more effective for irony detection in the context of sentiment analysis and NLP, we need to analyze existing data. For this purpose, we will use the English Twitter Corpus (ETC) which contains more than 3 million English tweets.

3.1 Programming languages

Three different object-oriented programming languages were considered to code the conversion algorithms which were Java, Python and MATLAB. These languages were highly rated by programmers and software developers in the area of natural language processing and machine learning.

Python is a versatile programming language that supports both object oriented programming and functional programming and is often used in machine learning and text processing applications because of its simplicity. It has its own text processing libraries called the natural language toolkit (NLTK) which contains a number of corpora and text processing libraries. The NLTK has many useful tools for pre-processing, as well its integration with scikit-learn, which is used for the classifying and evaluation and features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, and k-means, and is designed to inter-operate with the Python numerical and scientific libraries NumPy and SciPy.

- **Numpy** gives support for high-level mathematical functions, large arrays and matrices. When using linear algebra and random number capabilities is Numpy very useful. Numpy also has utilities for integrating Fortran and C/C++ code.
- **Scikit-learn** is a Machine Learning library that Google started working on in 2007. It is built on the math packages numpy and scipy. Among many other features it includes a variety of regression, classification and clustering algorithms

such as naive Bayes, random forests, support vector machines, boosting algorithms etc. ([40] Pedregosa et al., 2011)

- **NLTK** is an abbreviation for Natural Language ToolKit, it is a Python library that is useful when you work with human language data. The library consists of more than 50 different lexical assets and corpora including such as WordNet, parsing and tagging.
- **SciPy** is a library containing many routines for numerical integration and optimization.

Using Python would make the implementation considerably easier and faster. This is why it was selected for the writing of the code. The software will be written in the Python's IDE PyCharm. It is an integrated development environment used for programming in Python. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems, and supports web development with Django.

3.2 Project Requirements

3.2.1 Data Set

There is a growing body of research on irony detection in English tweets. The most commonly used data sets for irony detection in English tweets are the SemEval-2018 Task 3 data set, the SemEval-2019 Task 5 data set, and the Twitter Irony Corpus. These data sets contain tweets labelled as ironic or non-ironic, and provide a valuable resource for developing and testing irony detection systems. Those data sets, specifically the SemEval-2018 Task 3 one, will be used in the training and testing the system.

3.2.2 Lexicon

The text is analysed using Vader for classification. VADER (Valence Aware Dictionary for sEntiment Reasoning) is a simple rule-based model for general sentiment analysis. The VADER sentiment lexicon is gold-standard quality and has been validated by humans. It is a model used for text sentiment analysis that is sensitive to both polarity (positive/negative) and intensity (strength) of emotion. It is available in the NLTK package and can be applied directly to unlabeled text data.

3.3 Implementation

3.3.1 Feature Weighting

A big part of the pre-processing task is to choose a feature weighting scheme. There are various existing methods, however, we are going to review only three of them -

Binary, TF.IDF, and Bi-Normal Separation.

Binary

The binary weighting scheme in principal allocates a 1 if a word is present at least once in the document, otherwise the weight is 0. However, using only a binary feature weight is proven not to be optimal.

TF.IDF

Unlike binary weighting, in TF and TF.IDF weightings, the number of occurrences of a word is taken into consideration. TF or Term Frequency is represented as the number of times a word appears in the given text divided by the total number of words. This means that the more a word appears in the document, the higher weight it has. This is not always good, like for instance, when the word is also frequently seen in the document collection. Handling this involves including an Inverse Document Frequency (IDF), which reflects that less common terms in the collection will be more useful in finding the relevant documents. Therefore the TF.IDF is an improved version of TF that performs better in finding relevant documents.

Bi-Normal Separation

According to George Foreman, who developed the BNS weighting scheme, IDF does not utilise the class labels so some features are sometimes weighted incorrectly. It is oblivious to the class labels in the training set, which can lead to inappropriate scaling for some features. To correct this, BNS was developed and it replaced IDF, so TF.BNS was created. This increases the effect of important words on the kernel distance computations for the SVM.

Empirical evaluation on a benchmark of 237 binary text classification tasks shows substantially better accuracy and F-measure for a Support Vector Machine (SVM) by using BNS scaling. A wide variety of other feature representations were later tested and found inferior, as well as binary features with no scaling. Moreover, BNS scaling yielded better performance without feature selection, obviating the need for feature selection.

3.3.2 Bias

The measure of bias is an attempt to observe how the ratio of genuine and satirical articles in the data set has an effect on the success of the system. This is done rather simply by removing a proportion of the genuine articles such that for a bias value of 1.0, 100% of the genuine articles in the training and test sets are used, and none for a value of 0.0. Care must be taken when using this to re-balance the data sets. It reduces the size of the data set, so the smaller the value, the less pieces of data are used to train and fit the classifier. Taking this into account however, bias as described above should allow a quick way to see the effects of proportion on the results without adversely affecting the ability of the system.

3.3.3 Classification Models

For the implementation of this project, various different approaches were considered. The main idea is to use linear classifiers as the base models including Decision Trees, Random Forest, Logistic Regression, Naive Bayes, and Support Vector Machines. They were chosen due to their simplicity and robustness against over-fitting when dealing with high dimensional data such as text features extracted from tweets ([41] Tumasjan et al., 2010; [28] Hassan et al., 2017). A deep learning approach using transformers and neural networks was also considered due to the incredibly precise classification and performance. This however, has not been developed enough and it is not as accessible as using other approaches. Due to this and the complexity of the implementation of such an approach, the only artificial neural network that will be implemented is going to be a multilayer perceptron classifier.

After building and training the base models, stacking and boosting algorithms will be applied in hopes of improving the predictions of the models, thus achieving better results.

3.3.4 Evaluation

In order to measure the success of the work,it needs to be evaluated. This project will just use a standard method of evaluation, involving precision, recall, F-score, and accuracy. This is a common and simple implementation of evaluation.

The way to visualise the outcome of our system is to make a confusion matrix, which will provide more insight into not only the performance of the predictive model, but also which classes are being predicted correctly, incorrectly, and what type of errors are being made. It consists of two classes - positive and negative, which show whether the samples are classified as ironic or not ironic. The two types are true or false predictions, which indicate whether the classification was done correctly or not. This presents us with four outcomes - True Positive (TP), False Positive (FP), False Negative (FN), and True Negative (TN). Those indicate correctly classified ironic sample, incorrectly ironic, incorrectly not ironic, and correctly not ironic respectively.

With this information we can calculate the recall, precision, F-Measure, and accuracy.

	Actual Positive	**Actual Negative**
Predicted Positive	*True Positive (TP)*	*False Positive (FP)*
Predicted Negative	*False Negative (FN)*	*True Negative (TN)*

Table 3.1: Confusion Matrix

Recall

Recall is a metric that quantifies the number of correct positive predictions made out of all positive predictions that could have been made. Unlike precision that only comments on the correct positive predictions out of all positive predictions, recall provides an indication of missed positive predictions.

$$Precision = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{3.1}$$

The recall score can be calculated using the recall_score() scikit-learn function.

Precision

Precision is a metric that quantifies the number of correct positive predictions made - the number of items that have been correctly assigned to a particular class; Items that have been classified in the same way by both the system and the expert. It is calculated as the ratio of correctly predicted positive examples divided by the total number of positive examples that were predicted.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{3.2}$$

The precision score can be calculated using the precision_score() scikit-learn function.

F-Measure

F-Measure is used because it provides a way to combine both precision and recall into a single measure that captures both properties.

$$F1 - Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.3}$$

This is the harmonic mean of the two fractions. This is sometimes called the F-Score or the F1-Score and might be the most common metric used on imbalanced classification problems. The F-Score can be calculated using the f1_score() scikit-learn function. F1-Score gives equal weights to both Precision and Recall, however if we want to give more weight to either of those, we can use the Fβ-score, which allows the user to determine relative importance of Precision and Recall.

$$F\beta - Score = \frac{(1+\beta^2) * (Precision * Recall)}{(\beta^2 * Precision) + Recall} \tag{3.4}$$

β is chosen such that Recall is considered β times as important as Precision is.

Accuracy

Accuracy can be defined as the percentage of correct predictions made by our classification model. It can be calculated by the formula:

$$Accuracy = \frac{NumberOfCorrectPredictions}{NumberOfRowsInData} \tag{3.5}$$

Which can also be written as:

$$Accuracy = \frac{TruePositives + TrueNegatives}{NumberOfRowsInData} \tag{3.6}$$

Since F-score is used in more imbalanced data sets where the difference in the amount of instances in one class is significantly lower than those in the other classes, and the SemEval2018 Task 3 data sets are quite well balanced, we are going to use accuracy as the final evaluation metric in this project.

The accuracy of the classifiers will be tested using cross validation and the ones that give the best results will be tested against the gold standard data set provided in the SemEval2018 Task 3.

3.3.5 Cross Validation

In the process of cross-validation, the original data sample is randomly divided into several subsets. The machine learning model trains on all subsets, except one. After training, the model is tested by making predictions on the remaining subset. It protects a model from overfitting, especially if the amount of data available is limited, and is used to test the performance of a machine learning model and give a better impression of how the classifiers will perform on an independent data set. [42]

Chapter 4

Design

This chapter explains the design technique chosen for detecting irony in English tweets using the SemEval 2018 Task 3 data set. The used design technique is a machine learning-based approach that uses multiple classifiers, sentiment feature extraction, and text pre-processing to improve the performance of the irony detection model.

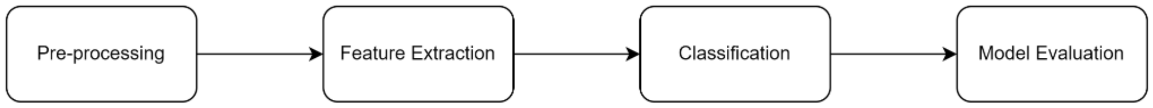

Figure 4.1: Project Design

This approach has been selected because it has been shown to improve the performance of text classification models, and it is a commonly used technique in natural language processing.

4.1 Text Pre-Processing

Text pre-processing is an essential step in natural language processing, and it involves cleaning, normalizing, and transforming raw text data into a format that can be easily processed by machine learning algorithms. In the irony detection task, text pre-processing is necessary to remove noise from the data and improve the quality of the features used in the model.

The text pre-processing pipeline used in this project involves the following steps:

- Removing URLs

- Removing mentions
- Removing special characters and digits
- Tokenization
- Removing stop words
- Stemming
- Joining the tokens back into a string

Tokenization is the process of splitting text data into individual words, and it is a crucial step in text pre-processing. In this project, the NLTK library is used to tokenize the text data.

Stop words are common words that do not add much meaning to the text, such as "the", "and", and "of". They are removed using the NLTK library which uses a list of such stop words and upon iteration of the given data set, if a word matches an instance from the list it is removed.

Stemming is the process of reducing a word to its root form, and it is used to reduce the dimensionality of the feature space. In this implementation, SnowballStemmer from the NLTK library is used to stem the words in the text data.

4.2 Feature Extraction

Feature extraction, which is related to dimensionality reduction, transforms arbitrary data, such as text or images, into numerical features that are fed to the modles and are usable for machine learning. It is used to convert given data into features (feature vectors) which are intended to be informative and non-redundant. This technique greatly improves the performance of the classification models, since it extracts the important features which are indicative of the presence of irony in a tweet. Useful feature extraction tools used in this project are *CountVectorizer* and *TfIdf.*

CountVectorizer is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into a vector on the basis of the frequency of each word that occurs in the entire text. Then *TfIdf* is used to apply "weights" to the produced vectors and give them higher or lower score based on their informational value for the desired task.

Another feature extraction tool that is specifically used for this project is the sentiment feature extraction technique. It involves extracting sentiment-related features from text data and is used to improve the performance of the irony detection model by incorporating sentiment-related information into the feature space.

Sentiment feature extraction involves computing sentiment scores for each text data point and then extracting relevant sentiment features from the scores. In this project, the *SentimentIntensityAnalyzer* from the NLTK library is used to compute sentiment scores. The sentiment scores are then used to extract the compound sentiment feature, which is a normalized score that ranges from -1 (most negative) to 1 (most positive).

4.3 Machine Learning Models

Machine learning models are used to learn patterns in the text data and make predictions about the presence of irony in each text data point. They are trained with a labelled training data set and based on this they generate predictions about the classification of a new unlabelled data set. In this project, multiple machine learning models are used to improve the performance of the irony detection model:

- **Decision Tree**
  Decision Tree is a tree-like model that partitions the feature space into a set of rectangular regions and makes predictions based on the majority class of the training examples within each region. It recursively splits the data into smaller subsets based on the most discriminative features, resulting in a tree-like structure.

- **Random Forest**
  Random Forest is an ensemble learning method that combines multiple decision trees to improve the accuracy and stability of the model. Each tree in the random forest is trained on a randomly sampled subset of the training data, and the final prediction is made by averaging the predictions of all the trees.

- **Multinomial Naive Bayes**
  Multinomial Naive Bayes is a probabilistic model that uses Bayes' theorem to predict the class of a given sample. It assumes that the features are independent of each other, and calculates the probability of each class based on the frequency of each feature in the training data.

- **Logistic Regression**
  Logistic Regression is a linear classification model that models the probability of the target variable as a function of the input features. It is used to predict binary

or multiclass outcomes and is particularly useful when the classes are linearly separable.

- **Support Vector Machines**
  Support Vector Machines is a binary classification model that finds the hyperplane that best separates the two classes. It maximizes the margin between the hyperplane and the closest data points, called support vectors, and can be extended to handle non-linearly separable data through kernel functions.

- **Multi Layer Perceptron**
  Multi Layer Perceptron is a neural network model that consists of multiple layers of interconnected nodes, or neurons. Each neuron performs a linear transformation followed by a non-linear activation function. The model is trained using backpropagation, which adjusts the weights of the connections between neurons to minimize the error between the predicted and actual output. It can be used for both classification and regression tasks.

- **AdaBoost Boosting Classifier**

These machine learning models are used to train a meta-classifier, which combines the predictions of the base classifiers to make the final prediction. This is called a Stacking Classifier and is used because it has been shown to improve the performance of ensemble models. The important factor for achieving the best results using stacking is finding the perfect combination of the base models to feed into the meta-classifier. Simply picking the highest scoring classifiers does not guarantee optimal performance for the meta-model. An extensive search for the best combination is required, which is done by trial and error.

Similarly, in machine learning, boosting is an ensemble meta-algorithm for primarily reducing bias that converts weak learners to strong ones. A boosting algorithm consists of iteratively learning the weak classifiers and adjusting the weights of the input data. Misclassified input data gain a higher weight and correctly classified data loses weight. Thus, the boosting algorithm "learns from past mistakes" of a weak classifier, improving its performance.

4.4 Model Evaluation

The performance of the irony detection model is evaluated using a 5-fold cross-validation technique. In each fold, the data is split into training and validation sets, and the model is trained on the training set and evaluated on the validation set. The final evaluation metric is the average performance across all 10 folds.

The evaluation metric used for this model is the classification accuracy, which is the percentage of correct predictions made by the classification models. F1-score is a popular metric for classification tasks and will also be considered, however, it is mainly used in cases where the classes are imbalanced, and our data set has close to 50/50 split between ironic and non-ironic tweets. The precision and recall values for each fold are also reported, which allows for further analysis of the model's performance.

In addition to the cross-validation evaluation, the model is also tested on a separate test set that was not used during training to ensure the classification of our algorithm performs well when fed with unseen data.

Chapter 5

Implementation and testing

In this chapter, we will discuss the implementation and testing of our irony detection model. We will highlight the novel aspects of our algorithm and describe the testing process we used to evaluate the model's performance.

5.1 Implementation

We begin by outlining the key steps involved in the implementation process. The implementation has been done using Python programming language, and we have used several libraries and frameworks such as Pandas, NumPy, Scikit-learn, NLTK, and TextBlob.

```python
import pandas as pd
import numpy as np
import nltk

import matplotlib.pyplot as plt

from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer

from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer

from textblob import TextBlob
from nltk.sentiment import SentimentIntensityAnalyzer
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import AdaBoostClassifier, StackingClassifier

from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection cross_val_score
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import precision_score, f1_score, recall_score
```

Listing 5.1: Used Libraries and Frameworks

5.1.1 Pre-Processing

The first step in the implementation is to pre-process the textual data. We have used a set of text pre-processing techniques to clean the data and make it suitable for training the machine learning models. The pre-processing steps include the removal of URLs, mentions, special characters, and digits. We have also tokenized the text data and removed the stop words. Additionally, we have applied stemming to reduce the words to their root forms. A spell checker was also implemented, however, it was decided against using it, since it adds heavy computational runtime and does not improve the performance significantly. Finally, we have used the TextBlob library to extract sentiment features such as sentiment polarity and sentiment subjectivity.

5.1.2 Data Loading

After pre-processing the data, we have loaded the Semeval2018 Task 3 data set using the Pandas library. We have used the read_csv function to read the data from the CSV files. We have also skipped the first row of the data set, which contains the column names.

```python
# Load the SemEval 2018 Task 3 dataset
train_data =
    pd.read_csv(
        "path_to_datasets/train/SemEval2018-T3-train-taskA_emoji.txt",
        skiprows=[0], sep="\t", header=None, names=["id", "label", "text"])
test_data =
    pd.read_csv(
        "path_to_datasets/goldtest/SemEval2018-T3_gold_test_taskA_emoji.txt",
        skiprows=[0], sep="\t", header=None, names=["id", "label", "text"])

X_train, y_train = train_data['text'].apply(preprocess), train_data["label"]
X_test, y_test = test_data['text'].apply(preprocess), test_data["label"]
```

Listing 5.2: Loading and Preprocessing the Datasets

5.1.3 Feature Extraction

To extract features from the pre-processed text data, we have used the CountVectorizer and TfidfTransformer classes from the Scikit-learn library. We have used the CountVectorizer class to convert the text data into a bag of words representation, and the TfidfTransformer class to transform the bag of words into a TF-IDF (Term Frequency-Inverse Document Frequency) representation. We have used n-grams with n=1, 2, and 3 to extract features from the text data.

5.1.4 Sentiment Analysis

We have used the TextBlob library to perform sentiment analysis on the pre-processed text data. We have extracted the sentiment polarity and sentiment subjectivity features from the text data and appended them to the pre-processed text.

5.1.5 Model Selection and Training

We have used several machine learning models to train the irony detection model, including Decision Tree, Random Forest, Logistic Regression, Multinomial Naive Bayes, Support Vector Machine, Multi Layer Perceptron, AdaBoost, and Stacking classifiers. The hyper-parameters for each of the base models are tuned for maximum efficiency using GridSearch. To avoid code repetition, we defined a set of models, created a loop, and iterated over each one to train them, and output the confusion matrix and the evaluation metrics. Immediately after, the meta-model is defined for the stacking and boosting classifiers. We have also used the StratifiedKFold function to split the training data into 5 folds for cross-validation.

```python
models = {
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'SVM': SVC(kernel="rbf", C=10, random_state=42, probability=True),
    'Logistic Regression': LogisticRegression(),
    'Naive Bayes': MultinomialNB(),
    'MLPClassifier': MLPClassifier(hidden_layer_sizes=(50, 20), max_iter=1000,
                                                              random_state=42)
}
```

Listing 5.3: Defining the Base Models

5.2 Testing and Evaluation Metrics

We have used several evaluation metrics to evaluate the performance of the irony detection model, including accuracy, precision, recall, F1-score, and confusion matrix.

We have used the classification_report function from the Scikit-learn library to calculate these evaluation metrics.

To assess the performance of our irony detection model, we used a 10-fold cross-validation technique, as described in the Analysis chapter. In each fold, the data was split into training and validation sets, and the model was trained on the training set and evaluated on the validation set.

In addition, we evaluated our model against a standard result set for calibration. We used the SemEval-2018 Task 3 gold test data set, which contains tweets labeled for irony detection. We compared the performance of our model to the top-performing models from the competition.

Chapter 6

Results and discussion

This chapter includes the performance evaluation for the different models used for the project. Tables of their classification report (precision, recall, and f1-score), accuracy, and confusion matrix will be shown.

During the evaluation process, some flaws and missed opportunities concerning the implementation of the project were discovered. When comparing the performance of the models under different circumstances, it became apparent that a higher accuracy was achieved without pre-processing the data sets rather than when pre-processing was applied. It may seem unnatural at first, however, it is consistent with the essence of our irony detection task. Pre-processing, although very useful and needed for cleaning and transforming raw text data by removing the bits with low informational value, sometimes, it can also strip off important and valuable data. For this exact project it is quite tough to determine which bits of information are useful for training the models, and which are not. The process of selecting the best combination of pre-processing and feature extraction steps is computationally expensive and time-consuming. Due to the limited amount of time given for this project, this perfect combination is yet to be explored in further research. Nevertheless, the implemented program still achieves relatively high scores and performs well on unseen data sets.

The table below indicates an approximation of the accuracy percentage to the nearest round number for the different models implemented. It compares the accuracy achieved with and without pre-processing.

Classifier	**Accuracy in %**	
	Without Pre-processing	**With Pre-processing**
Decision Tree	57%	57%
Logistic Regression	65%	65%
Random Forest	64%	65%
Naive Bayes	67%	63%
SVM	67%	62%
MLP	69%	63%
AdaBoost	65%	64%
Stacking	66%	66%

Table 6.1: Accuracy comparison of classifiers with and without pre-processing

Even if the numbers might seem quite similar and too close to show a significant change, we can spot interesting patterns. For most of the models, their performance remained relatively unchanged no matter the pre-processing steps applied or the lack there of. This might me a sign that they might not be the most useful models for our specific task. The lowest accuracy was achieved by the Decision Tree classifier, which is to be expected due to its simplicity.

	Precision	Recall	F1-Score	Support
Non-Ironic	0.66	0.61	0.63	473
Ironic	0.47	0.52	0.49	311
Accuracy			0.5739795918367347	
Precision			0.5624905087319666	
Recall			0.5643392724825462	
F1-Score			0.5623127941805734	

Table 6.2: Decision Tree Classification Report

In both cases - with pre-processing and without it, the top scoring base models are fed in the stacking meta model. Interestingly, even with the discrepancy in performances based on the presence of pre-processing, the results for the stacking model remain the same.

	Precision	Recall	F1-Score	Support
Non-Ironic	0.72	0.73	0.73	473
Ironic	0.58	0.58	0.58	311
Accuracy			0.6696428571428571	
Precision			0.6546027501909855	
Recall			0.6540825136129107	
F1-Score			0.6543321683738175	

Table 6.3: Stacking Meta Model Classification Report

The models achieving highest accuracy and best performance overall are the Multi Layer Perceptron, Support Vector Machines, and Naive Bayes classifiers respectively. The best result that we get from the program as a whole is from the Multi Layer Perceptron when no pre-processing steps are applied and it scores as high as 69%.

	Precision	Recall	F1-Score	Support
Non-Ironic	0.73	0.78	0.76	473
Ironic	0.63	0.56	0.59	311
Accuracy			0.6938775510204082	
Precision			0.6789855331309163	
Recall			0.6703126380835197	
F1-Score			0.6730222503353885	

Table 6.4: Multi Layer Perceptron Classification Report

	Predicted 0	Predicted 1
Actual 0	371	102
Actual 1	138	173

Table 6.5: Confusion matrix (0 = Non-Ironic; 1 = Ironic)

At this stage of the development of the assignment, 69% is quite the milestone for such a small-scale project. For a comparison, the two highest scoring implementations of an irony detection program using the SemEval2018 Task 3 data sets, achieve 79.7% and 78% F1-Score. The former one uses Deep Convolutional Networks and was developed by Pranav Goel, Nipun Batra, and Mohit Sharma [43]. The latter one uses Deep Learning and was developed by Arjun Atreya, Divyansh Singhvi, and Nisarg Jhaveri [44].

Overall, this evaluation gives us a better idea of how the whole process for implementing an irony detection program works. We have seen which models are of no great use when aiming for good performance, we acquired a better understanding of which are the important bits that need more attention and improvement, and finally, we got an overall satisfying results comparison with similar projects developed in the past years.

The current state of the program and research at the moment is left open-ended. There are several areas of investigation that can be pursued to improve the accuracy of the irony detection program. A path that is definitely worth taking is to experiment more with the feature extraction and pre-processing steps. Integrating deep learning and using neural networks or transformers is also an essential step when aiming to achieve the highest possible performance results. Additionally, to also see the bigger picture, a possible consideration is to incorporate contextual information, such as the user's profile information and historical tweets, to better understand the user's tone and intentions.

Chapter 7

Conclusions

In this report, we looked into existing and researched methods for tackling the task of irony detection in English Tweets. Various approaches were discussed and taken into consideration, however, for the experimental setup of the project, a combination of approaches was chosen. The data set used was from the SemEval2018 Task 3 competition, which consists of a training set with close to 4000 labelled tweets, an unlabelled testing set, and a gold set. Numerous pre-processing methods were used, however, as described in the results chapter, it was discovered that the performance increases when omitting the pre-processing step.

Nevertheless, the main objective of the project was to develop an irony detection program that can accurately classify tweets as either ironic or non-ironic, and this goal was mostly achieved, as the program was able to produce relatively high accuracy results (69%). However, the program also had some limitations in terms of detecting subtle forms of irony and sarcasm, which are difficult to capture through linguistic features alone, which indicates that there is still room for improvement in the program's accuracy. Moreover, the program was developed and tested using a data set of English tweets, which may limit its applicability to other languages and cultures. Therefore, future research should investigate the feasibility of developing similar programs for other languages.

Overall, the results achieved are satisfactory, but further research is needed to explore new approaches and incorporate additional features to improve the program's performance.

Bibliography

[1] H. Olkoniemi and J. Kaakinen, "Processing of irony in text: A systematic review of eye tracking studies," 10 2020.

[2] I. Ruch, M. Konkol, and M. Pakoca, "Sarcasm detection on czech and slovak twitter," in *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC 2018)*, pp. 1827–1832, 2018.

[3] R. W. Gibbs and H. L. Colston, *Irony in language and thought: A cognitive science reader.* Psychology Press, 2014.

[4] R. W. Gibbs, "On the psycholinguistics of sarcasm.," *Journal of Experimental Psychology: General*, vol. 115, no. 1, p. 3–15, 1986.

[5] R. Kreuz and G. Caucci, "Lexical influences on the perception of sarcasm," in *Proceedings of the Workshop on Computational Approaches to Figurative Language*, (Rochester, New York), pp. 1–4, Association for Computational Linguistics, Apr. 2007.

[6] R. J. Kreuz and S. Glucksberg, "How to be sarcastic: The echoic reminder theory of verbal irony.," *Journal of Experimental Psychology: General*, vol. 118, no. 4, p. 374–386, 1989.

[7] A. Utsumi, "Verbal irony as implicit display of ironic environment: Distinguishing ironic utterances from nonirony," *Journal of Pragmatics*, vol. 32, no. 12, p. 1777–1806, 2000.

[8] E. Filatova, "Irony and sarcasm: Corpus generation and analysis using crowdsourcing," in *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, (Istanbul, Turkey), pp. 392–398, European Language Resources Association (ELRA), May 2012.

[9] M. Hu and B. Liu, "Mining and summarizing customer reviews," *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004.

[10] B. Liu, *Opinion Summarization and Search*, p. 218–230. Cambridge University Press, 2015.

[11] R. González-Ibáñez, S. Muresan, and N. Wacholder, “Identifying sarcasm in Twitter: A closer look,” in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, (Portland, Oregon, USA), pp. 581–586, Association for Computational Linguistics, June 2011.

[12] D. Davidov, O. Tsur, and A. Rappoport, “Semi-supervised recognition of sarcastic sentences in twitter and amazon,” in *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, CoNLL ’10, (USA), p. 107–116, Association for Computational Linguistics, 2010.

[13] A. Reyes, P. Rosso, and T. Veale, “A multidimensional approach for detecting irony in twitter,” *Language Resources and Evaluation*, vol. 47, no. 1, p. 239–268, 2012.

[14] E. Laksana, T. Baltrusaitis, L.-P. Morency, and J. P. Pestian, “Investigating facial behavior indicators of suicidal ideation,” *2017 12th IEEE International Conference on Automatic Face amp; Gesture Recognition (FG 2017)*, 2017.

[15] F. Zare, J. Noorbakhsh, T. Wang, J. Chuang, and S. Nabavi, “Integrative deep learning for pancancer molecular subtype classification using histopathological images and rnaseq data,” 11 2020.

[16] B. Liu and L. Zhang, “A survey of opinion mining and sentiment analysis,” *Mining Text Data*, pp. 415–463, 08 2013.

[17] L. Polanyi and A. Zaenen, “Contextual valence shifters,” in *Computing Attitude and Affect in Text*, 2006.

[18] M. Taboada, J. Brooke, M. Tofiloski, K. Voll, and M. Stede, “Lexicon-based methods for sentiment analysis,” *Computational Linguistics*, vol. 37, pp. 267–307, June 2011.

[19] T. Wilson, J. Wiebe, and P. Hoffmann, “Recognizing contextual polarity in phrase-level sentiment analysis,” *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processingnbsp; - HLT ’05*, 2005.

[20] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, “Recursive deep models for semantic compositionality over a sentiment treebank,” in *Proceedings of the 2013 Conference on Empirical Methods in Natural*

Language Processing, (Seattle, Washington, USA), pp. 1631–1642, Association for Computational Linguistics, Oct. 2013.

[21] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1746–1751, Association for Computational Linguistics, Oct. 2014.

[22] D. Bamman, "Natural language processing for the long tail," in *International Conference on Digital Health*, 2017.

[23] J. Ling, A. Kurzawski, and J. Templeton, "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance," *Journal of Fluid Mechanics*, vol. 807, p. 155–166, 2016.

[24] S. Kim, N. Fiorini, W. J. Wilbur, and Z. Lu, "Bridging the gap: Incorporating a semantic similarity measure for effectively mapping pubmed queries to documents," *Journal of Biomedical Informatics*, vol. 75, pp. 122–127, 2017.

[25] J. Devlin, M.-W. Chang, K. Lee, and K. 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, Volume 1 (Long and Short Papers)*, (Minneapolis, Minnesota), pp. 4171–4186, Association for Computational Linguistics, June 2019.

[26] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. M. Shazeer, I. Simon, C. Hawthorne, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music transformer: Generating music with long-term structure," in *International Conference on Learning Representations*, 2018.

[27] M. Fayaz, A. Khan, M. Bilal, and S. Khan, "Machine learning for fake news classification with optimal feature selection," *Soft Computing*, vol. 26, 08 2022.

[28] J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M. M. A. Patwary, Y. Yang, and Y. Zhou, "Deep learning scaling is predictable, empirically," 12 2017.

[29] S. Attardo, J. Eisterhold, J. Hay, and I. Poggi, "Multimodal markers of irony and sarcasm," *Humor-international Journal of Humor Research - HUMOR*, vol. 16, pp. 243–260, 01 2003.

[30] S. Zhang, X. Zhang, J. Chan, and P. Rosso, "Irony detection via sentiment-based transfer learning," *Information Processing Management*, vol. 56, no. 5, pp. 1633–1644, 2019.

[31] P. Gil, "What is twitter amp; how does it work?," Aug 2021.

[32] R. Řehůřek and P. Sojka, "Software framework for topic modelling with large corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pp. 45–50, 2010.

[33] B. Liu, *Sentiment analysis and opinion mining.* Morgan & Claypool Publishers, 2012.

[34] D. W. McDonald and A. Krieger, "Reputation as a sufficient condition for data quality on amazon mechanical turk," *Behavior research methods*, vol. 48, no. 4, pp. 1301–1309, 2016.

[35] P. Kumar, R. Ghosh, and S. Kumar, "Importance of sentiment analysis in social media," *Procedia Computer Science*, vol. 132, pp. 1574–1583, 2018.

[36] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval.* New York, NY, USA: Cambridge University Press, 2008.

[37] S. Günnemann, H.-P. Kriegel, J. Sander, and A. Zimek, "Clustering: A review," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 10, no. 1, pp. 1–58, 2017.

[38] L. A. Oliveira and A. H. F. Laender, "A survey of hierarchical document clustering and categorization," in *2015 Brazilian Conference on Intelligent Systems (BRACIS)*, pp. 129–134, IEEE, 2015.

[39] D. D. Lewis, Y. Yang, T. Rose, F. Li, X. Liu, D. Aurelio, and H. Ng, "An evaluation of phrasal and clustered representations on a text categorization task," in *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 37–50, 1995.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[41] A. Tumasjan, T. O. Sprenger, P. G. Sandner, and I. M. Welpe, "Predicting elections with twitter: What 140 characters reveal about political sentiment," *ICWSM*, vol. 10, pp. 178–185, 2010.

[42] A. Joby, "What is cross-validation? comparing machine learning models."

[43] P. Goel, N. Batra, and M. Sharma, “Sarcasm detection using deep convolutional neural networks,” in *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, Association for Computational Linguistics, 2018.

[44] A. Atreya, D. Singhvi, and N. Jhaveri, “Irony detection in english tweets using deep learning,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.