SENTIMENT ANALYSIS OF MOVIE REVIEWS USING MACHINE LEARNING

Julia Mitroi
York University

May 6, 2019

Abstract

Sentiment analysis is the automated process of extracting an opinion about a given subject from written or spoken language. Movie reviews analysis of reviewers' sentiments can provide insights to improve movie campaign success and product messaging, and to determine marketing strategy. The goal of this project was to explore and apply machine learning techniques used in sentiment classification for movie reviews, and to assess the benefit of using those techniques for this kind of problem. The project was based on the Kaggle data science competition named "Sentiment Analysis on Movie Reviews" that uses a Rotten Tomatoes reviews dataset. Several feature extraction methods and classification algorithms were implemented to train a classifier on the movie reviews dataset to distinguish between five sentiment categories; and model performance (prediction accuracy) was evaluated and compared to baseline algorithms. The best-performing model, a Keras Sequential Neural Network Model of CNN, BiLSTM, and Dense Network layers, had a classification accuracy score of 64.1%*, and yielded AUC scores between 0.8 and 0.94.*

1. Introduction

Sentiment analysis is one of the most popular applications of Natural Language Processing (NLP). It builds systems that automatically identify and extract opinions, attitudes, and feelings (or "sentiments") from natural language [1]. It can be used to analyze sentiments of diverse datasets ranging from corporate surveys to movie reviews. This has many practical applications, including user review and marketing analysis, social media monitoring, political views analysis, financial prediction, brand monitoring, customer service, workforce analytics and voice of employee, and product analytics.

In the film industry, the quality and volume of movie ratings and reviews is considered to reflect movie quality (a movie being the product being evaluated), and the reviews usually influence consumer purchase decisions and have an effect on daily box office performance (i.e., revenues) [2]. Movie reviews sentiment analysis can thus be an excellent source of information, providing insights that can determine marketing strategy; improve movie campaign success, product messaging, and customer service; test business Key Performance Indicators; and generate leads. If done correctly, sentiment analysis can improve a company's bottom line.

The business objective of this project was to explore and apply state-of-the-art machine learning techniques used for sentiment classification of movie reviews, and to quantify the benefit of using those techniques for this kind of problem. The project was based on the Kaggle data science competition named "Sentiment Analysis on Movie Reviews" that uses a public Rotten Tomatoes movie reviews dataset. The competition was inspired by the work of Socher et al. (2013) [3], the challenge being to label phrases on a scale of the five sentiment values, or classes. Factors such as sentence negation, sarcasm, slang words, misspellings, terseness, and language ambiguity made the task particularly challenging. To solve the Kaggle movie reviews challenge and select a best-performing model, I implemented a supervised machine learning model (Multinomial Naïve Bayes), and deep learning algorithms that incorporated Long Short-term Memory (LSTM) Recurrent Neural Networks (RNN), bidirectional Long Short-term Mem-

ory (bi-LSTM) RNN, and Convolutional Neural Networks (CNN). Deep Learning encompasses a diverse set of algorithms that attempts to imitate how the human brain works by employing artificial neural networks to process data.

2. Related Work

Sentiment analysis has achieved a good amount of progress over the past 10 years, including the proposal of new methods and the creation of benchmark datasets [4]. The first approaches in this field of research (which were lexicon-based) depended on the use of words at a symbolic level (unigrams, bigrams, bag-of-words features), where generalizing to new words was difficult. State-of-the-art sentiment analysis methods are based on supervised machine learning and deep learning, and have long surpassed the sentiment prediction accuracy of lexicon-based methods. The most common current approach is to use pre-trained word embeddings in combination with a supervised classifier [4]. In this framework, the word embedding algorithm acts as a feature extractor for classification. RNN variants, such as the LSTM network [5] or the Gated Recurrent Units [6], are alternatives of a feed-forward network that include a memory state capable of learning long distance dependencies. In various forms, they have proven useful for text classification tasks [7]; [8]. Socher et al. (2013) [3] and Tai et al. (2015) [7] used GloVe vectors in combination with a RNN. As their dataset was annotated for sentiment at each node of a parse tree, they trained and tested on these annotated phrases. Both Socher et al. (2013) [3] and Tai et al. (2015) [7] also proposed several RNNs that were able to take better advantage of the labeled nodes, and which achieved better results than standard RNNs. For example, Socher et al. (2013) [3] introduced the Recursive Neural Tensor Network; its accuracy of predicting fine-grained sentiment labels for all phrases reached 80.7%, an improvement of 9.7% over bag of features baselines; and it was the only model that could accurately capture the effects of negation and its scope at various tree levels for both positive and negative phrases. In their paper titled "Assessing State-of-the-Art Sentiment Models on State-of-the-Art Sentiment Datasets", Barnes et al. (2017) [4] found that both LSTMs and Bi-LSTMs are particularly good at fine-grained sentiment tasks (i. e., data analysis with more than two classes, such as the Rotten Tomatoes movie reviews dataset).

Convolutional Neural Networks (CNN) have also proven effective for text classification. CNNs are generally used in computer vision, however they have recently been applied to various NLP tasks and the results were promising [9] [10]. Kim (2014) [9] used skipgram vectors as input to a variety of CNNs and tested on seven datasets. The best performing setup across datasets was a single layer CNN which updates the original skipgram vectors during training. Overall, the approaches noted in this section of the paper currently achieve state-of-the-art results across multiple datasets [4].

3. Methodology

3.1. Data Understanding

3.1.1 Data Collection

The data for this project was sourced from the Kaggle repository at www.kaggle.com/c/sentiment-analysis-on-movie-reviews. The Rotten Tomatoes reviews dataset is a corpus of movie reviews used for sentiment analysis, originally collected by Pang and Lee (2005) [11], and subsequently parsed to add fine-grained labels by Socher et al. (2013) [3].

3.1.2 Dataset Description

Socher et al. (2013) used Amazon's Mechanical Turk to create fine-grained labels for all parsed phrases in the corpus. A label is an integer between 0 and 4 indicating if the review is positive (4), somewhat positive (3), neutral (2), somewhat negative (1) or negative (0). The dataset is comprised of tab-separated sample phrases, as each sentence in the movie re-

views has been parsed into multiple phrases by the Stanford parser. For example, a sample phrase is “A series of escapades demonstrating the adage that what is good for the goose is also good for the gander, some of which occasionally amuses but none of which amounts to much of a story”, with a score of 1 (somewhat negative), as well as “good for the goose”, with a score of 3 (neutral). Each sentence has a Sentence ID and each phrase has a Phrase ID. Phrases that are repeated (such as short/common words) were only included once in the data.

There are two subsets: training and test movie reviews. The train.tsv file contains the training phrases and their associated sentiment labels, as well as the Sentence IDs. The test.tsv file contains just phrases; using a classification model such as the ones built through this project, sentiment labels could be assigned to each phrase in the test set (or to new reviews data of a similar structure). The training reviews dataset was annotated for the 5 levels of sentiment (positive, somewhat positive, neutral, somewhat negative, and negative) both at the clause level, where each node in a binary tree is given a sentiment score, as well as at sentence level.

3.1.3 Exploratory Data Analysis

The purpose of Exploratory Data Analysis (EDA) is to analyze a dataset and summarize its main characteristics, to see what the data reveals beyond the modeling results. As part of the data pre-processing, I explored the data through visualizations, and by calculating basic statistics such as the number of comments/phrases across label categories. Counts calculation using Python revealed that the Rotten Tomatoes reviews data is composed of 8,529 reviews divided into 156,060 labeled training phrase samples; and of 3,310 reviews divided into 66,292 unlabeled test samples. As noted, the goal of the competition is to use the training samples to build a classification model that can be used to make predictions on the test set.

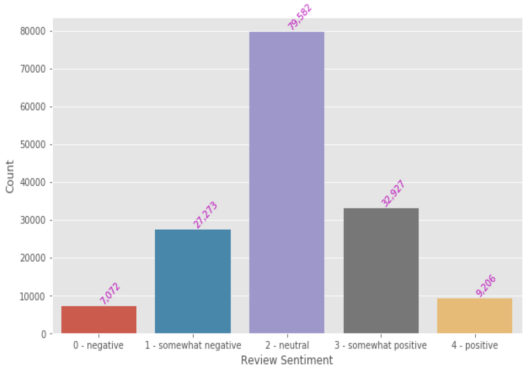

Figure 1: *Distribution of Sentiment Classes*

Exploring the distribution of labels, a bar chart shows that the training data have a class imbalance in that some classes have significantly less/more training data than others. There are 7072 negative, 27273 somewhat negative, 79582 neutral, 32927 somewhat positive, and 9206 positive samples. Sentiment classes follow a normal distribution, with the most frequently distributed class being "2", "neutral". This type of distribution could lead to a model not having sufficient data to learn the less-represented classes, and is something to be aware of when evaluating a model applied to data. The fewest observations are for the negative (0) and positive sentences (4), however the difference is not necessarily as big from the other classes as to justify the need of over/under-sampling or other technics to close the gap. Data augmentation could be used to balance the classes, and I would have attempted it in a subsequent stage of this project. Figure 1 shows a graphical representation (bar chart) of the review counts by sentiment class for the training set.

I also performed vocabulary exploration by creating several word clouds with the most representative words for each of the five classes in the training set. The word cloud on the training data reveals that words or pairs of words such as “movie", “film", “romantic comedy", “soap opera", "feel good", "old fashioned", “New York", “writer", and "director" have the highest frequency in the review phrases. Fig-

Figure 2: *Word Cloud of Training Set*

ure 2 shows the word cloud for the entire training set.

Word clouds for each sentiment class were created as well. Visual examination of the word clouds showed that, as expected, vocabulary for Sentiments 0 and 1 ("negative" and "somewhat negative") have more words with negative connotation, such as "bad", "little", "nothing, "lack"; and vocabulary for Sentiments 3 and 4 ("somewhat positive" and "positive") have more words with positive connotation, such as "best", "good", "well", "fascinating", and "fun". Based on the size of the words in the word clouds, vocabulary size of each class reflects the number of sentences depicted in the counts and bar charts above, namely reduced vocabulary for Sentiments 0 and 4. The models built for classification might consequently have to rely more on the order of the words in the phrases versus the vocabulary, to differentiate between a negative statement and somewhat negative statement, or positive and somewhat positive statement. The Jupyter Notebook with the full EDA and additional data summaries is located on GitHub at https://github.com/jmitroi

3.1.4 Dataset Preparation

In this dataset, the reviews are already split into sentences, thus sentence segmentation, which is the step before tokenization, did not need to be done; since the dataset is annotated for sentiment at each node of a parse tree, the models trained on these annotated phrases. In preparation for modeling, I performed data cleaning. I first used a function that kept only letters from A to Z and digits, then filtered out HTML content, and converted the text to lower case.

3.1.5 Feature Engineering

After cleaning, words needed to be tokenized into numeric format to be passed to supervised machine learning and deep learning models. Models operate on a numeric feature space, expecting input as a two-dimensional array where rows are instances and columns are features. To perform machine learning on text, the text needs to be transformed into vector representations such that we can apply numeric machine learning. This process is called feature extraction or vectorization (or feature engineering), and is an essential step toward language-aware analysis. The sentiment analysis task is typically modeled as a classification problem where a classifier is fed with input text and returns the corresponding sentiment category (e.g. positive, negative, or neutral). On a general level, the machine learning classifier is implemented with the steps and components depicted in Figure 3. In this project, I used the Python's NLTK library to normalize and vectorize the input text; as well as a pre-trained word vector, GloVe. GloVe is an external embedding that enhances the precision of a deep learning network because it integrates new information (lexical and semantic) about the words, information that has been trained and distilled on a very large corpus of data.

Tokenization, the first step of normalization, is done by splitting apart words whenever there's a space between them. For this, I used the Twitter and Treebank Word tokenizers, as well as Keras library's text preprocessing functions. After tokenization, I used lemmatization (NLTK) to further reduce dimensionality. Lemmatizing means grouping together variant forms of the same word (e.g., "walk", "walking", "walks", are reduced to the base form, "walk"). I did not perform stop word removal

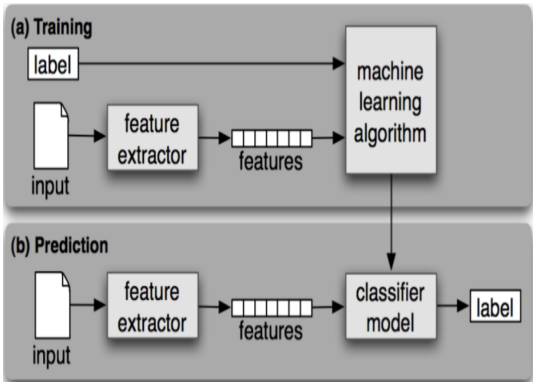

Figure 3: *Modeling Pipeline*

on this dataset to filter out "stop-words" (e.g., “the”, “is”, “are”), as RNNs – the main models that I built – are superior at learning context from previously encountered information. For movie reviews, an informal phrase such as "this movie is shit" would have opposite meaning to "this movie is the shit", hence I would like that type of information to be available to the models.

3.2. Classifiers

The classifiers used in this project were from the scikit-learn (Multinomial Naïve Bayes) and Keras (Sequential Network) packages. The scikit-learn (sklearn) library in Python provides unsupervised and supervised learning algorithms, including classification, and interoperates with the Python numerical and scientific libraries NumPy and SciPy. Keras is a minimalist Python library for deep learning that was developed to make implementing deep learning models as fast and easy as possible for research and development.

3.3. Modeling

Once the feature extraction process was complete, the next step was the training process, which is the first step of modeling, where the model learns the text input and the corresponding output (tag or label) based on the samples used for training. For this project, the data were already split into training and test sets. The goal of modeling was to create a model that generalized well to new data. The training dataset contained the examples used for learning, and was used to fit the parameters of the classifiers. The validation dataset (which I created while coding the models) provided an unbiased evaluation of a model’s fit on the training dataset while tuning the model’s hyper-parameters; it was used to assess how well the trained model works on examples it hadn’t seen during training. I obtained the validation set by using k-fold cross-validation with 10 folds for the machine learning model, while for the deep learning models the validation set was obtained by iteratively applying the validation function of the Keras sequential model.

Questions that I intended to answer through my modeling work for this project were: How do RNNs and CNN perform for this kind of problem? Can CNN and RNNs be combined to achieve better results? What is the benefit for modeling of using pre-trained word embeddings such as Glove? How can more ‘traditional’ NLP techniques, such as lemmatization, be used to improve performance? Can recent research findings (e.g., [4]) be applied to improve my results? How well does an RNN (LSTM) perform compared to a simpler classification algorithm?

The first algorithm that I built, *Multinomial Naïve Bayes* (MNB), was based on supervised machine learning. MNB is a classification algorithm well-suited for discrete data such as multi-category movie ratings, as each rating has a certain frequency to represent, and in text learning the count of each word predicts the class or label. It works similar to the Gaussian Naive Bayes, however the features are assumed to be multinomially distributed; in practice, this means that this classifier is commonly used with discrete data. It makes a simplified yet powerful assumption that each word has independent probability within its class, therefore not considering the position of words within text and taking the traditional ‘bag-of-words’ NLP approach; and it estimates sentiments by

counting the words and their likelihoods in each class together with the class' prior probability. Due to its simplicity and efficiency, I used MNB as a baseline model for my project. However, for the same reasons that it is easy to implement and makes simplified assumptions, this classifier can have limited performance. Its evaluation for suitability for this project is discussed below in Section 4.

After building a baseline machine learning model, I used the *Keras Neural Network Sequential Model* to analyze sentiments, testing several architectures consisting of embedding layers and CNN, LSTM RNN, and dense (fully connected) neural network (DNN) layers. For this purpose, I first built a function that could perform either architecture, and adapted it when executing the function by changing the parameters, and validating on 1000 samples. As noted in the "Related Work" section of this paper, LSTM and bi-LSTM RNNs combined with pre-trained word embeddings are the state-of-the-art for sentiment analysis, and perform particularly well for multi-class sentiment prediction [4]. RNNs are especially well adaptable to NLP tasks due to their nature and their ability to process sequence data; they are closer to how humans look at a sentence when we are reading it than other types of algorithms. Based on this background, I implemented LSTM and bi-LSTM RNNs for my sequential models. For one of the models, I also tried a CNN-DNN combination, without LSTM layers. Embedding layers, which I used in two of the models, are useful as they allow avoiding the sparsity and high dimensionality of one-hot encoded vectors. Even if already using an embedding layer, a pre-trained word embedding, which I also used for two of the models, can improve model quality since it contains information from thousands of lines of varied text; it is a type of transfer learning, which can be useful when we don't have extensive datasets ("big data") for modeling, as was the case in this project. The table in Figure 4 lists the classification models that I built, and their characteristics. The Jupyter Notebooks with the Python code for the models are located on GitHub at https://github.com/jmitroi

Feature engineering	Classification model	Cross-validation/ Callbacks	Val accuracy	Avg acc. classif. score	AUC scores
NLTK Keras	Mult. Naïve Bayes	k-Fold Cross-Validati on (10 folds)	-	46.37	-
NLTK Keras	**Keras Sequential:** Embedding Layer+LSTM Layer+DNN Layer	*(Model was subsequently improved by adding early stopping)*	59.90	60.4	.90, .81 .80, .75 .92
GloVe	**Keras Sequential:** CNN Layer+BiLSTM Layers+DNN Layer	Early Stopping	62.8	62.1	.91, .84 .82, .79 .93
GloVe	**Keras Sequential:** CNN Layer+BiLSTM Layers+DNN Layer	Early Stopping + ReduceLROnPlateau	64.1	64.1	.87, .85 .81, .80 .94
NLTK Keras	**Keras Sequential:** Embedding Layer+CNN Layer+DNN Layer	Early Stopping + ReduceLROnPlateau	63.3	63.3	.89, .79 .79, .76 .94

Figure 4: *Project Models*

In total, I built four variations of the Keras sequential model, two in conjunction with base NLTK/Keras normalization (traditional NLP method) and two in conjunction with GloVe word embeddings. For the first sequential model that I attempted, I did not use early stopping, which resulted in overfitting. I added this as a Keras callback method to the next three models, and to two of them I also added the ReduceLROnPlateau callback to reduce the learning rate when the validation loss increased from one epoch to another. The rationale behind the Keras function ReduceLROnPlateau() is that models benefit from reducing learning rate once learning stagnates; and it is good practice to apply it towards improving model performance. For all models, I used the Keras Nesterov Adam (Nadam) optimizer, which is a variation of the Adam optimizer with Nesterov accelerated gradient momentum.

4. Results

Evaluating the Multinomial Naïve Bayes model using the sklearn accuracy_score() metric, the cross-validation accuracy was low at 46.37%. This means that using this model, for each 100 review phrases it classified, only 46.4 were classified as having the correct sentiment.

Before building the Keras sequential model, I also built a baseline (or "dummy") model

to determine if the newly built models will result in an improvement, by comparing to the baseline. As this is a classification problem, the baseline model depicts what would be the accuracy of a model that always predicts the most recurrent class in the dataset ("neutral"). The accuracy for the baseline model was 21.3%.

After running the Python code multiple times and tuning the hyper-parameters on each of my four sequential models based on accuracy, by adjusting the filters on the models' network layers, the kernel size, and/or dropout levels, the best performing model achieved a 64.1% validation accuracy, 64.1% average classification accuracy score, and AUC (Area Under the ROC Curve) scores of 0.87, 0.85, 0.81, 0.80, and 0.94 for the 5 sentiment classes of negative, somewhat negative, neutral, somewhat positive, and positive classes respectively. This best-achieving model had the following architecture: CNN layers, followed by BiLSTM layers, followed by DNN layers; and I used GloVe pre-trained word embeddings to vectorize before running the model on the training data. The AUC measure is particularly useful for classification problems with a class imbalance, as was the case for the movie reviews data in this project, where some classes had significantly less data than others (as per the EDA). AUC scores above 0.80 are acceptable for classification purposes; and AUC scores above 0.90 are considered to be "excellent". In this project, an AUC score of 0.94 for the positive class, for example (depicted in Figure 5), is interpreted to predict with 94% certainty that the sentiment of an input phrase labeled by the model as positive is positive; in other words, the classifier is very good at minimizing false negatives (positive phrases which are classified as belonging to a non-positive category) and true negatives (non-positive phrases which are classified as not being in the positive sentiment class). The CNN-DNN sequential network that I also attempted did not surpass the CNN-BiLSTM-DNN network's performance, achieving an accuracy score of 63.3%. The precision and recall scores based on confusion matrices created to evaluate the models noted above, and full Python programs, are in the output of the Jupyter Notebooks on GitHub (https://github.com/jmitroi).

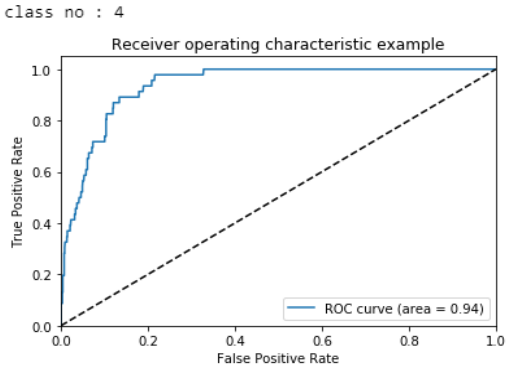

Figure 5: *AUC for Positive Sentiment*

5. Discussion and Future Work

In this project, I developed and compared several classification algorithms in conjunction with feature extraction methods, and selected a best-performing model for multi-class sentiment analysis. Among the approaches that I used, including Multinomial Naive Bayes and LSTM/BiLSTM networks, the BiLSTM RNN showed the most promising result. In line with state-of-the art results in the area of sentiment analysis research, LSTM/BiLSTM networks are particularly suited for fine-grained sentiment tasks. Factors that could have further improved model performance are: more training data, balanced data across classes (e.g., data augmentation), and training the models in a cloud environment, as running times can be significant with local computer resources; as well as using advanced fine-tuning techniques like Random Search and Genetic Algorithm to improve performance. The 64.1% model accuracy that I achieved is not high for a machine learning/deep learning model, but it is not low for multi-class sentiment analysis, where the highest accuracy typically achieved is below 70%; in contrast with binary text classification (positive vs. negative sentiments), making fine-grained sentiment distinctions is not trivial for

an algorithm, and more research on improving performance for multi-class sentiment analysis algorithms is needed. Most of the work on sentiment analysis in recent years has been around developing more accurate sentiment classifiers by dealing with some of the main challenges and limitations in the field. These include text subjectivity and tone, context and polarity, irony and sarcasm, comparisons, emojis, and defining "neutral". Sentiment analysis should be paired up with tools that can identify audience, content, and tone, to understand the context of the communication, as even in the best circumstances, multi-class text classification is usually only $65 - 70\%$ accurate, and the accuracy drops even further when the process is applied to text in languages other than English [12]. By undertaking the project, my work could contribute to efforts to find optimal models for sentiment analysis of social media content, which could potentially be extrapolated to other, non-social-media types of text classification such as employee sentiment analysis based on corporate survey data.

References

[1] Pang, B. & Lee, L. (2008). Opinion Mining and Sentiment Analysis. Found. Trends Inf. Retr. 2, 1-2 (January 2008), 1-135. doi:http://dx.doi.org/10.1561/1500000011.

[2] Duan, W., Gu, B. & Whinston, A. B. (2008). Do Online Reviews Matter? - an Empirical Investigation of Panel Data. Decision Support Systems, Vol. 45, No. 4, pp. 1007-1016.

[3] Socher, R., Perelygin, A., Wu, j., Chuang, J., Manning, C., Ng, A., & Potts, C. (2013). Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. Conference on Empirical Methods in Natural Language Processing (EMNLP 2013).

[4] Barnes, J., Klinger, R., & Schulte im Walde, S. (2017). Assessing State-of-the-Art Sentiment Models on State-of-the-Art Sentiment Datasets. In: Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, Copenhagen, Denmark, doi:10.18653/v1/W17-5202.

[5] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8):1735–1780.

[6] Chung, C.J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. CoRR, abs/1412.3555.

[7] Tai, K.S., Socher, R, & Manning, C.D. (2015). Improved semantic representations from tree-structured long short-term memory networks. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL).

[8] Tang, D., Qin, B., Feng, X., & Liu, T. (2016). Effective LSTMs for target-dependent sentiment classification. In Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers.

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

[10] Flekova, L. & Gurevych, I. (2016). Supersense embeddings: A unified model for supersense interpretation, prediction, and utilization. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL).

[11] Pang, B. & Lee, L. (2005). Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In ACL, pages 115–124.

[12] Heires, K. (2015). Sentiment Analysis: Are You Feeling Risky? Retrieved from https://digitalreasoning.com/blog/sentiment-analysis-feeling-risky/