Seminar: NLP in Industry, Winter Semester 2020/21

Sentiment Analysis for Stock Price Prediction

Claire Zhao SUN
M.Sc. Scientific Computing, Heidelberg University
Matriculation Number 3630998
pz237@stud.uni-heidelberg.de

Abstract

This project aims to apply various techniques from Natural Language Processing (NLP) to solve a real life problem, namely using Twitter sentiment to predict share price movements. An end-to-end pipeline, including data collection, text preprocessing, sentiment analysis, as well as price prediction, has been implemented, based on extensive experiments with various Machine Learning techniques and existing NLP tools. Further improvement opportunities have also been identified to enhance the current prototype, especially the price prediction model would benefit from accumulation of more data over a longer period of time.

1 Introduction

An exuberant surge in stock market participation from retail investors has been observed during the COVID-19 pandemic lockdown. The most recent trading frenzy in GameStop has highlighted the increasingly impactful role that retail investors play in the stock markets. While traditional institutional investors are serviced by professional advisors and financial analysts who regularly provide 'market color', the amateur individual investors, on the other hand, are mostly left to their own devices. Social media such as *Twitter* and *Reddit* forums have become especially popular amongst the day-traders as they share their investment strategies and opinions about certain stocks in those channels. However, manually scanning these social media platforms to gain a sense of the market sentiment is not only time-consuming and repetitive, but also likely to be subjective due to personal confirmation bias. This is an underserved market segment that would benefit from an analytics tool that systematically collects textual data on stocks from social media feeds and automatically analyses them to extract sentiments. It would help both the *Robinhood* generation of technology-savvy day traders and the traditional institutional investors to obtain a better sense of the market mood based on the collective wisdom of crowds.

This project aims to create a prototype pipeline that programmatically collects Twitter feed data on specific companies and automatically extracts sentiments from them. Moreover, it would detect shifts in sentiment trends and predict share price movements accordingly. The primary goal of this project is to assess the applicability of various Machine Learning (ML) techniques from publicly available Natural Language Processing (NLP) libraries, in order to find a feasible, practical solution to a real-world problem.

2 Related Work

Sentiment Analysis This is a well-researched area: Mäntylä et al. (2016) offered a comprehensive review of the evolution of sentiment analysis, while Pang and Lee (2008) introduced many basic concepts and fundamental approaches. Specifically with Twitter data, Zhao and Gui (2017), Ramachandran and Parvathi (2019) and Pradha et al. (2019) discussed the challenges in preprocessing techniques due to the noisy and colloquial nature of Twitter texts. Kolchyna et al. (2015) compared two approaches for Twitter sentiment analysis: lexicon-based and machine learning-based methods. Furthermore, the advent of pretrained general-purpose language models such as BERT (Devlin et al. (2018)) and its variants, e.g., BERTweet (Nguyen et al. (2020)), TweetBERT (Qudar and Mago (2020)) and TwitterBERT (Azzouza et al. (2020)), offers promising new tools for performing sentiment analysis on social media content.

Price Predictions Alzazah and Cheng (2020) surveyed the recent advances in stock market prediction using text mining; they also compared various machine learning and deep learning methods

used for sentiment analysis. Bollen et al. (2010) analysed Twitter feeds by two mood tracking tools, OpinionFinder and Google-Profile of Mood States (GPOMS), and observed that the time series of public mood states were predictive of daily changes in the composite Dow Jones Industrial Average (DJIA) index value. With regards to predicting individual stock price trends, it is still inconclusive whether social media sentiments provide meaningful alpha. Bujari et al. (2017) illustrated that some forecasting models could predict the next day direction of certain stock movements with 82% of success. However, Mudinas et al. (2019) argue that it is only on some specific occasions sentiment emotions seem to Granger-cause stock price changes, but the exhibited pattern is not universal and needs to be assessed on a case by case basis.

3 Methods and Approach

This project aims to build a model for predicting individual stock price movements from analysing Twitter sentiments. The overview of the full pipeline is presented in Figure 1. The envisioned user-interface only requires manual input of the stock ticker and company name; the subsequent modules (i.e. data collection, text preprocessing, sentiment analysis and price prediction) run automatically to produce the final output, which is the share price prediction for the next day. Currently, the sentiment analysis is only based on Twitter data, however the model's modular architecture provides flexibility to incorporate supplementary data streams from other social media channels (e.g. Reddit, Stocktwits, etc.). The entire pipeline is written in Python, with all supporting libraries publicly available. A detailed illustration of each module's architecture is discussed below.

3.1 Data Collection Module

Raw tweets Raw tweets are programmatically collected via the public *Twitter API*[1] (standard v1.1) by a customized Python script that takes user input for query terms related to a company. Users can also specify the search period. However, the free version of *Twitter API* only returns a subset of tweets published in the past seven days. This is therefore a limiting factor for the model's output quality, as it is unclear whether the sampled tweets returned by the public API are indeed statistically representative of the true underlying sentiments of the entire Twitter platform.

For the purpose of this project, tweets related to five companies (Tesla, Nikola, Nio, GM, Microsoft) have been downloaded on a weekly basis over the period from end-November 2020 to end-January 2021. Query terms include company names, 'cash tags' with stock tickers (e.g. '$TSLA'), and the exclusion of retweets. In addition to the text of raw tweet content, data such as Tweet-ID, user screen name, favourite counts, retweet counts, etc. have also been harvested (but not currently utilized). Due to an initial oversight, tweets gathered (before 09-December) were based on hashtaged search terms (i.e. '#$TLSA' and '#Tesla'), which returned significantly less tweets than search terms with no hashtag (i.e. '$TLSA' and 'Tesla') that were used for later batches of data downloaded (after 09-December).

Share prices Historical daily share prices are extracted from *Yahoo!Finance* via a data downloader *yfinance*[2]. We have collected for the same five companies as above for the same time period, the following trading data: adjusted closing price, unadjusted closing price, daily high, daily low, open price and daily trading volume, on a daily basis. We only uses the 'adjusted closing price' as our model input for this project but might consider including more financial data in the future to enhance the model.

Training dataset In order to establish a baseline for benchmarking, the annotated datasets from SemEval (Nakov et al. (2016)) have been selected as the training data for our model. More specifically, only the most applicable datasets, namely SemEval2016-Task4-SubtaskA (10,000 entries), SemEval2014-Task9-SubtaskA (10,681 entries), and SemEval2013-Task2-SubtaskB (11,338 entries) are included.[3] The training data follows the single-label multi-class classification system with three sentiment labels POSITIVE, NEUTRAL, NEGATIVE. Since these datasets only provide Tweet-IDs and the corresponding sentiment labels, the actual text content has been downloaded via Twitter API. As some Tweets are no longer available (perhaps due to user deletion), a total of 19,653

[1] https://developer.twitter.com/en/docs/twitter-api

[2] https://github.com/ranaroussi/yfinance

[3] SemEval-2017 uses the same dataset as SemEval-2016 while SemEval-2015 uses the same dataset as SemEval-2013; from 2018 onwards, there have been no more three-point classification tasks on Twitter data.

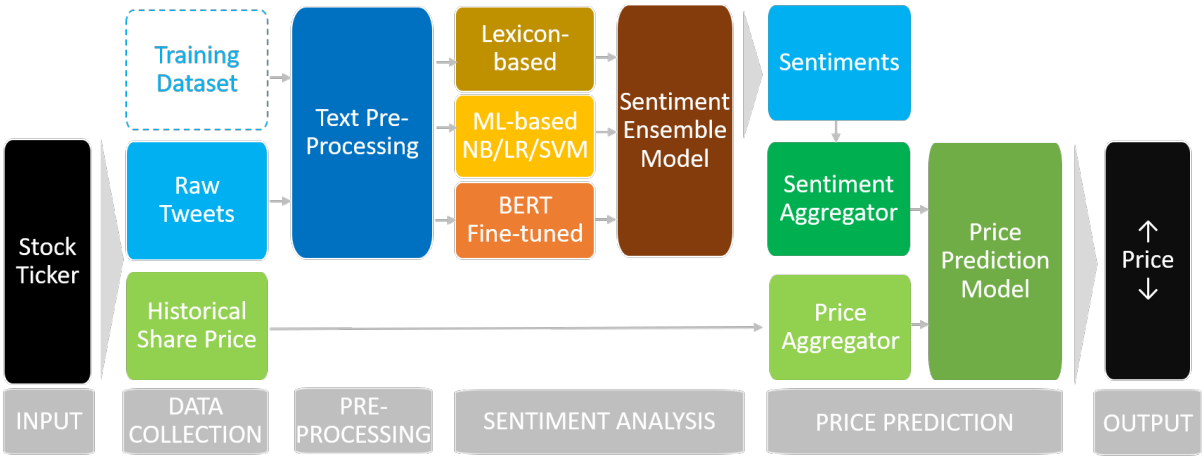

Figure 1: Overview of the end-to-end pipeline for the Twitter sentiment price prediction model

Tweets were collected (from 32,019 Tweet IDs), of which, c. 48% is labeled POSITIVE, 32% NEUTRAL and 21% NEGATIVE.

3.2 Text Preprocessing Module

The following steps have been included in the text preprocessing pipeline:

- remove urls, @username and hashtag symbol # using regular expressions, but keep the hashtaged terms since they might express users' sentiments
- handle negation by replacing *don't, can't*, etc with *do not, can not*, etc.
- restore elongated words such as *coooooool, goooooood* by reducing the number of repeated characters to two, though this does not handle edge cases such as *yeeessss, hahahaha*
- translate emojis and emoticon synsets into corresponding text descriptions, e.g. from *'=)'* to *'happy'* based on a custom-built emoticon dictionary that decodes the most frequently used 150 emoticons gathered from various sources (Wang and Castanon (2015), Wegrzyn-Wolska et al. (2016))[4]. For emojis, the model leverages the existing Python library, *emoji*[5], to transcribe the Unicode emojis to text-based strings, e.g. from ☺ (U+1F600) to *'grinning face'*
- decode acronyms and slangs by means of a proprietary lexicon built from aggregating a variety of sources[6][7][8]
- remove the remaining special characters (but not punctuation marks) after the previous step of translating emoticons

The following procedures have been implemented as optional steps in the pipeline:

- spelling correction: with the help of spell-checker *SymSpell* [9]; though one side effect of this is that it automatically removes all punctuation marks and significantly slows down the speed of the preprocessing pipeline
- remove stopwords: currently using Spacy's default stopword list.[10] This can be easily adapted to a customized stopword list, if necessary
- stemming: the default option is PorterStemmer from NLTK.[11] However, studies have shown that stemming and lemmatisation do not necessarily improve sentiment analysis and models such as BERT can easily deal with different forms of the same word
- remove numerics
- convert to lower-case

[4] https://en.wikipedia.org/wiki/List_of_emoticons
[5] https://pypi.org/project/emoji/
[6] http://www.noslang.com/dictionary
[7] https://www.webopedia.com/reference/text-message-abbreviations
[8] https://www.kaggle.com/aksharagadwe/abbreviations-and-slangs-for-text-preprocessing
[9] https://github.com/mammothb/symspellpy
[10] https://github.com/explosion/spaCy/blob/master/spacy/lang/en/stop_words.py
[11] https://www.nltk.org/howto/stem.html

- tokenization: representing each word as a token (e.g. in preparation of a bag-of-words approach); however, this step is not always necessary since some models, such as BERT, require their own special tokenization procedures

The preprocessing pipeline is designed to be modular and each step can be individually configured to be enabled or disabled. For the baseline experiment setup, we have not included the optional steps mentioned above.

3.3 Sentiment Classification Module

This module performs sentiment analysis on the preprocessing module's clean text output by means of several different classification techniques, including lexicon/rule-based, classical machine-learning-based as well as the state-of-the-art BERT model with fine-tuning. Each sub-module predicts the mostly likely sentiment class that the input text belongs to, and subsequently the predictions are fed into an ensemble model which outputs the aggregated sentiment prediction.

3.3.1 Lexicon-based Classification

For Lexicon/Rule-based methods, two existing applications have been considered: TextBlob[12] and VADER.[13]

TextBlob is a Python library that leverages NLTK and Pattern. It is a simple, lexicon-based method that relies on a predefined dictionary[14] which classifies words by their semantic orientation and intensity. The sentiment analysis function in TextBlob outputs two indicators: subjectivity and polarity. The polarity is a float that lies between [-1,1]; -1 indicating negative sentiment and +1 indicating positive sentiment. The overall sentiment of a sentence is the average of all its words' scores. Therefore, it does not handle sentences with negation words well. Another obvious disadvantage of Textblob is that it only considers words and phrases that are in its predefined dictionary and ignores words that are not.

VADER *(Valence Aware Dictionary and sEntiment Reasoner)* is an open source lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media (Hutto and Gilbert (2015)). The SentimentIntensityAnalyzer from VADER outputs a compounded polarity score that is computed by summing the valence scores of each word in the lexicon, adjusted according to specific rules. The score is normalized to be between -1 and +1, and the typical thresholds are: {positive: $>=$ 0.05; neutral: between -0.05 and +0.05; negative: $<=$ -0.05}. Similar to TextBlob, the main drawback with VADER is that it only considers individual words in a sentence and completely ignores the context.

Experiments have been conducted with both TextBlob and VADER on the training dataset and the results are shown in Figure 3. Sentiment classification by TextBlob performed poorly with a macro F1 score of 48%, whereas Vader performed slightly better at 53%. Based on this, Vader was selected as a candidate to the ensemble model and TextBlob was rejected.

3.3.2 Machine Learning-based Classification

Before applying various machine-learning classification techniques, the text data needs to be converted into numeric values in a vector representation. This is known as *vectorization* or *word embedding*. The simplest approach is to count the number of occurrences of individual words in a document (count vector), or the term frequency-inverted document frequency (TF-IDF). More sophisticated models, such as Word2Vec (Mikolov et al. (2013)) and GloVe (Pennington et al. (2014)), can also capture the semantic similarities between word representations. Word2Vec embeddings are learnt from training a shallow feed-forward neural network while GloVe embeddings are based on matrix factorization.

For this project, we have experimented with four different vectorization methods: 1) CountVectorizer and 2) TfidfVectorizer, both from scikit-learn[15] and fitted on the training dataset, as well as 3) Word2Vec and 4) GloVe, both of which are available as pretrained models from Hugging Face.[16] Through experimentation, we have found that some vectorization methods work better with certain types of classifiers but not others; this is further elaborated in the following section.

Naive Bayes Classifier This classification method is based on Bayes Theorem and it assumes complete independence of each feature for a given class, hence the joint likelihood can be 'naively' multiplied throughout all features in a

[12] https://textblob.readthedocs.io/en/dev/
[13] https://github.com/cjhutto/vaderSentiment
[14] https://github.com/sloria/TextBlob/en/en-sentiment.xml
[15] https://scikit-learn.org
[16] https://huggingface.co

given document. Naive Bayes is a generative, probabilistic classifier, meaning that for a given document, it returns the class with the maximum posterior probability:

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{f \in F} P(f|c)$$

where F represents the feature set and C the class set. The project adopts *CounterVectorizer* and *naive_bayes MultinomialNB* from scikit-learn in the model pipeline.

Logistic Regression This is a discriminative classification method that computes the posterior conditional probability directly: it learns to assign a higher weight to features with greater influence to discriminate between different classes. A softmax function

$$softmax(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}; z = wx + b$$

is applied to calculate the class probability distribution, while the weights (w, b) are learned from a labeled training set by minimizing the cross-entropy loss via an iterative algorithm such as gradient descent. In our model pipeline, we apply *TfidfVectorizer* coupled with *LogisticRegression* from scikit-learn.

Support Vector Machine SVM is a machine learning classification technique that uses a kernel function to map data points into a space where they are linearly separable by a set of hyperplanes. These hyperplanes are chosen in a way to maximize the distance from the nearest data points of each class. We utilize *LinearSVC* from scikit-learn in combination with pretrained Word2Vec embeddings.

3.3.3 BERT Fine-tuning

BERT (Bidirectional Encoder Representations from Transformers) is a general-purpose, natural language processing model released by Google Research in 2018 (Devlin et al. (2018)). Its architecture is mainly based on Transformer, an attention mechanism that learns contextual relations between words (or sub-words). Pretrained on Wikipedia and BooksCorpus with a combination of masked language modeling objective and next sentence prediction, BERT can be adapted to various downstream NLP tasks. For example, to fine-tune for sentiment analysis, a linear prediction layer is appended to the pooled outputs of BERT, which is initialized with pretrained weights; the combined model is then trained on a supervised dataset. Fine-tuning makes changes to all model parameters (Merchant et al. (2020)) but the most prominent impact is on the attention of the last few layers (Kovaleva et al. (2019)).

BERT is open source and various pretrained versions (e.g. base vs large, cased vs uncased, multilingual, etc.) are available for download directly from Hugging Face. We have incorporated the pretrained BERT-base-cased model (12-layer, 768-hidden, 12-heads, 109M parameters, trained on cased English text)[17] in our pipeline and fine-tuned it with the labelled training dataset. Moreover, we have also assessed BERTweet (Nguyen et al. (2020)), a variant of BERT pretrained for English tweets. However, our experiment results show that BERTweet underperforms BERT-base-case after fine-tuning. Hence we only include BERT-base-case in our final pipeline.

3.3.4 Ensemble model

Our sentiment analysis pipeline does not depend on a single output of a particular classification method, since none of the above discussed classifiers is accurate enough based on the training set results. This might be due to the inherently noisy nature of Twitter data. Furthermore, each classifier has its own strengths and weaknesses, and they do not necessarily always agree with each other, we have implemented an ensemble model in order to aggregate the outputs from five selected classifiers, namely:

- BERT-base-case with fine-tuning
- SVM with pretrained Word2Vec embedding
- Logistic Regression with TFIDF vectorization
- Naive Bayes with count vectorization
- VADER's Sentiment Analyzer

These sub-modules have been selected primarily based on their respective performance on the labelled training data set. In addition, we also maximize the diversification of classifier types to take advantage of 'the wisdom of the crowd'. We have experimented using several designs for the ensemble model, including majority vote, a fully connected layer with softmax function, a convolution

[17] https://huggingface.co/transformers/pretrained_models.html

layer plus two fully connected layers, etc. The preliminary conclusion is that the model architecture complexity does not necessarily improve classification performance and the simple majority vote method would suffice in our case.

3.4 Share Price Prediction Module

The share price prediction module forecasts the next day share price movement based on the historical share prices over the preceding n-day period and the corresponding daily Twitter sentiment over the same period. Instead of using the actual closing prices, an aggregator computes the percentage change of the share price vs the previous trading day before feeding it into the price prediction model. Similarly, the output from the sentiment analysis module is aggregated on a daily basis in the following vector representation:

$$S_t = \begin{bmatrix} \%Positive \\ \%Neutral \\ \%Negative \\ \#Tweets \end{bmatrix}$$ [18]

with daily shift in Twitter sentiment defined as $\triangle S_t = S_t - S_{t-1}$. Since tweets are collected on all calendar days including weekends and holidays, whereas share price data is only available on trading days, daily sentiment shifts on non-trading days are then aggregated into the last trading day. For instance, if the model is specified to predict share price based on a 1-day lag, in order to predict next Monday's share price, the model takes last Friday's closing share price and the corresponding sentiment shifts are aggregated as ($\triangle S_{Fri}$+$\triangle S_{Sat}$+$\triangle S_{Sun}$).

We have experimented with RNN, LSTM and CNN-based models for price prediction and trained them on the historical share prices and raw tweets collected from end-November 2020 to end-January 2021. Due to limited data (only 39 days of share prices for each company), the price prediction module is not fully developed at this stage. We include it here anyway for illustration purposes.

RNN & LSTM A Recurrent Neural Network (RNN) is a useful tool for handling sequential data. A defining feature of RNNs is the feedback loop where the output sequence of the previous data input (i.e. hidden state) is combined with the next data input before feeding to the neuron again. This means that RNNs process input data in a sequential manner, rather than consuming all input data at once. A variant of RNNs, known as Long Short-Term Memory (LSTM), allows the network to retain long-term dependencies from many previous time steps, through carefully managed gating mechanisms. During training, parameters of both RNN and LSTM are updated via back-propagation through time (BPTT).

Both RNN and LSTM models have a single layer of neurons with 5 hidden states followed by a fully connected layer. To accommodate for an input with 5 features, there are 66 and 246 trainable parameters in the RNN and LSTM models, respectively.

CNN This alternative version of the price prediction model is inspired by computer vision and image processing. It consists of a single layer of 1D convolutional filters and treats each input feature as a separate input channel. After the convolutional layer and a non-linear tanh activation function, the signal is passed into a fully connected layer to output the final prediction. For an input with 5 features, there are only 36 trainable parameters in the CNN model, much less as compared to the RNN and LSTM model parameters.

The RNN/LSTM/CNN models are trained to output a prediction for the relative percentage change in a stock's share price one-day forward, from which the next day's absolute share price could be easily estimated based on the previous day's closing price. To showcase the end-to-end model pipeline, we include a project demo in the Supplemental Material section.

4 Experiments and Results

4.1 Sentiment Analysis

The flow chart in Figure 2 demonstrates the overall schema of the experiment setup for the sentiment analysis module. The labelled datasets are split into training (80%), validation (10%) and test (10%) sets. After the text preprocessing stage, the clean text is applied to all classification models as indicated in the chart. We have tried different combinations of vectorizers with classifier models on the training set and used the validation set to fine-tune model hyperparameters and select the best performing configurations to choose the submodules for the ensemble model, based on a combination of evaluation metrics such as macro average precision, recall and F1 scores. Finally we used the reserved test dataset to compare our experimental results of all model configurations in Figure 3. The performance of the selected sub-modules for

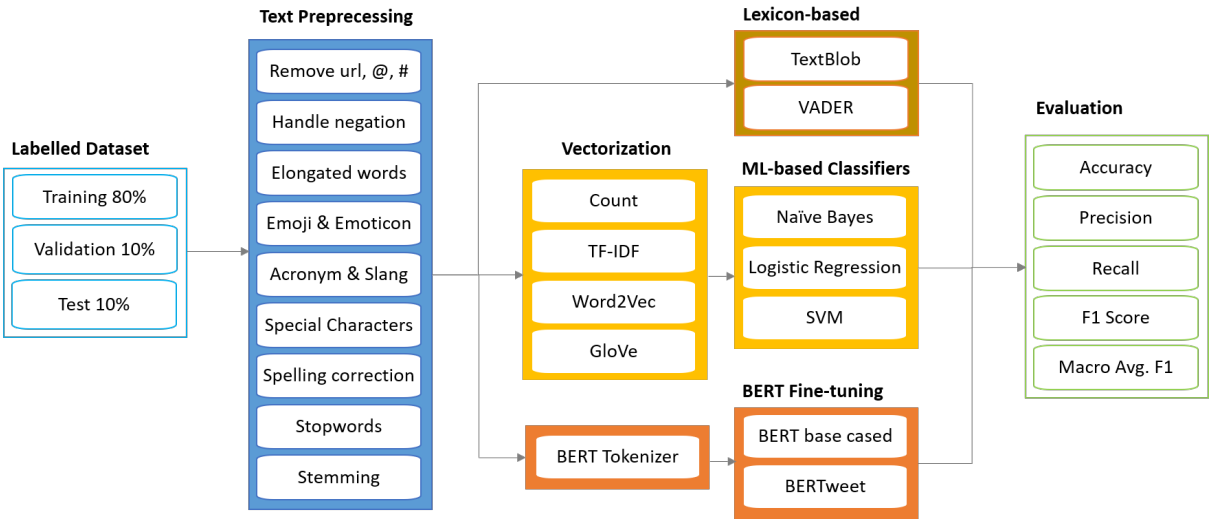

Figure 2: Sentiment Analysis Experiment Setup

the ensemble model are highlighted in the same respective colors as the schematic diagram.

There are a few observations worth highlighting: 1) Since our training data has an inherent imbalanced class distribution (with positive sentiments being the dominant class), we have applied stratified sampling when splitting the training data. Moreover, we have tried applying the Synthetic Minority Oversampling Technique (SMOTE) on the training set. However, we found SMOTE resampling to be very computationally intensive and the results were similar to setting model parameter *class_weight='balanced'* in the respective ML-classifiers.

2) Bigram and trigram vectorizations underperform unigram in both count and TFIDF for all three classifiers.

3) Word2Vec embeddings pretrained on Google news surprisingly perform better than Glove embeddings which are pretrained on Twitter data.

4) Our best ML-classifiers have macro average F1 scores between 59% to 61%, which is not too far from the best-in-class results achieved in the 2016 SemEval competition (63% top score from Nakov et al. (2016)).

Classification Method	Macro Avgerage Precision	Recall	F1-score
Vader	54%	54%	53%
TextBlob	*48%*	*48%*	*48%*
Count_NB (SMOTE)	*58%*	*60%*	*59%*
Count_LR (SMOTE)	*55%*	*56%*	*55%*
Count_SVM (SMOTE)	*53%*	*53%*	*53%*
Count_NB (balanced_weight)	59%	60%	59%
Count_LR (balanced_weight)	*57%*	*58%*	*57%*
Count_SVM (balanced_weight)	*52%*	*53%*	*53%*
Count_NB_unigram,bigram,trigram	*57%*	*58%*	*57%*
Count_LR_unigram,bigram,trigram	*57%*	*57%*	*57%*
Count_SVM_unigram,bigram,trigram	*53%*	*52%*	*52%*
TFIDF_NB (balanced_weight)	*59%*	*58%*	*58%*
TFIDF_LR (balanced_weight)	60%	61%	60%
TFIDF_SVM (balanced_weight)	*57%*	*57%*	*57%*
TFIDF_NB_bigram	*52%*	*53%*	*53%*
TFIDF_LR_bigram	*52%*	*53%*	*52%*
TFIDF_SVM_bigram	*50%*	*50%*	*50%*
GloVe-Twitter-25_NB	*48%*	*49%*	*44%*
GloVe-Twitter-25_LR	*51%*	*53%*	*51%*
GloVe-Twitter-25_SVM	*51%*	*52%*	*51%*
GloVe-Twitter-200_NB	*51%*	*51%*	*48%*
GloVe-Twitter-200_LR	*55%*	*57%*	*55%*
GloVe-Twitter-200_SVM	*55%*	*56%*	*56%*
Word2Vec-News-300_NB	*51%*	*52%*	*48%*
Word2Vec-News-300_LR	*60%*	*62%*	*60%*
Word2Vec-News-300_SVM	61%	62%	61%
BERTweet	*29%*	*30%*	*20%*
BERT Base Cased	64%	64%	64%
Ensemble_FC_Softmax	*64%*	*64%*	*64%*
Ensemble_MajorityVote	64%	66%	64%

Figure 3: Evaluation of classification models on the test set; highlighted models are selected as sub-modules to the ensemble model.

4.2 Price Prediction

As discussed previously, due to the limited time for data collection, the price prediction module is constrained. We have designed simple versions of RNN, LSTM and CNN models, however, we cannot meaningfully train a model for each company with only 39 days of share price data. For demonstration purposes, the experiment setup is shown in Figure 4.

We trained the price prediction model by concatenating the relative percentage change in daily prices from all five companies and also performed the same concatenation procedure on the corresponding daily sentiment shifts. We ran the model with a one-day lag with all input features (% change in price, shift in positive sentiment, shift in negative sentiment, shift in neutral sentiment, total tweet

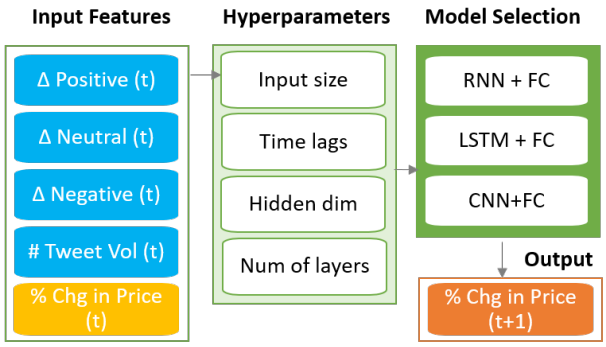

Figure 4: Price Prediction Experiment Setup

volume) and show the results in Figure 5.

This price prediction model is still a work-in-progress and would benefit from having a larger sample set.

5 Proposed Improvements

We have identified a few potential areas to in which we can further improve our project:

- We could benefit from developing our own annotated training set, in order to tune our model to this domain-specific problem.
- Not all collected data is consumed by the current model. For example, we have trading volume data as well as Twitter metadata that we could apply to potentially enhance the model architecture.
- Iteratively refine the preprocessing pipeline to improve data cleansing and further optimize model training and inferencing in order to accelerate processing speed and increase data-handling capacity.
- For industrial use, subscription to Twitter's Enterprise API would be necessary to secure a more comprehensive data feed as well as to further streamline the data collection process.

6 Conclusion

This project is intended to translate research findings into a real world problem solution. Through the process of building an end-to-end sentiment analysis and price prediction pipeline, we broadened our understanding of key concepts and methodologies in NLP and ML.Additionally, we gained awareness of the various challenges of applying research techniques to a practical problem. Finally, we proposed multiple improvements needed for eventual deployment in retail or industrial settings.

References

Faten Alzazah and Xiaochun Cheng. 2020. Recent advances in stock market prediction using text mining: A survey.

Noureddine Azzouza, Karima Aklii Astouati, and Roliana Ibrahim. 2020. Twitterbert: Framework for twitter sentiment analysis based on pre-trained language model representations. pages 428–437.

Johan Bollen, Huina Mao, and Xiao-Jun Zeng. 2010. Twitter mood predicts the stock market. *Journal of Computational Science*, 2.

A. Bujari, Marco Furini, and Nicolas Laina. 2017. On using cashtags to predict companies stock trends. In *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 25–28.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding.

C.J. Hutto and Eric Gilbert. 2015. Vader: A parsimonious rule-based model for sentiment analysis of social media text.

Olga Kolchyna, Thársis Souza, Philip Treleaven, and Tomaso Aste. 2015. Twitter sentiment analysis: Lexicon method, machine learning method and their combination.

Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. 2019. Revealing the dark secrets of bert.

Amil Merchant, Elahe Rahimtoroghi, Ellie Pavlick, and Ian Tenney. 2020. What happens to bert embeddings during fine-tuning?

Tomas Mikolov, G.s Corrado, Kai Chen, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. pages 1–12.

Andrius Mudinas, Dell Zhang, and Mark Levene. 2019. Market trend prediction using sentiment analysis: Lessons learned and paths forward.

Mika Mäntylä, Daniel Graziotin, and Miikka Kuutila. 2016. The evolution of sentiment analysis - a review of research topics, venues, and top cited papers. *Computer Science Review*, 27.

Preslav Nakov, Alan Ritter, Sara Rosenthal, Fabrizio Sebastiani, and Veselin Stoyanov. 2016. Semeval-2016 task 4: Sentiment analysis in twitter. pages 1–18.

Dat Quoc Nguyen, Thanh Vu, and Anh Nguyen. 2020. Bertweet: A pre-trained language model for english tweets. pages 9–14.

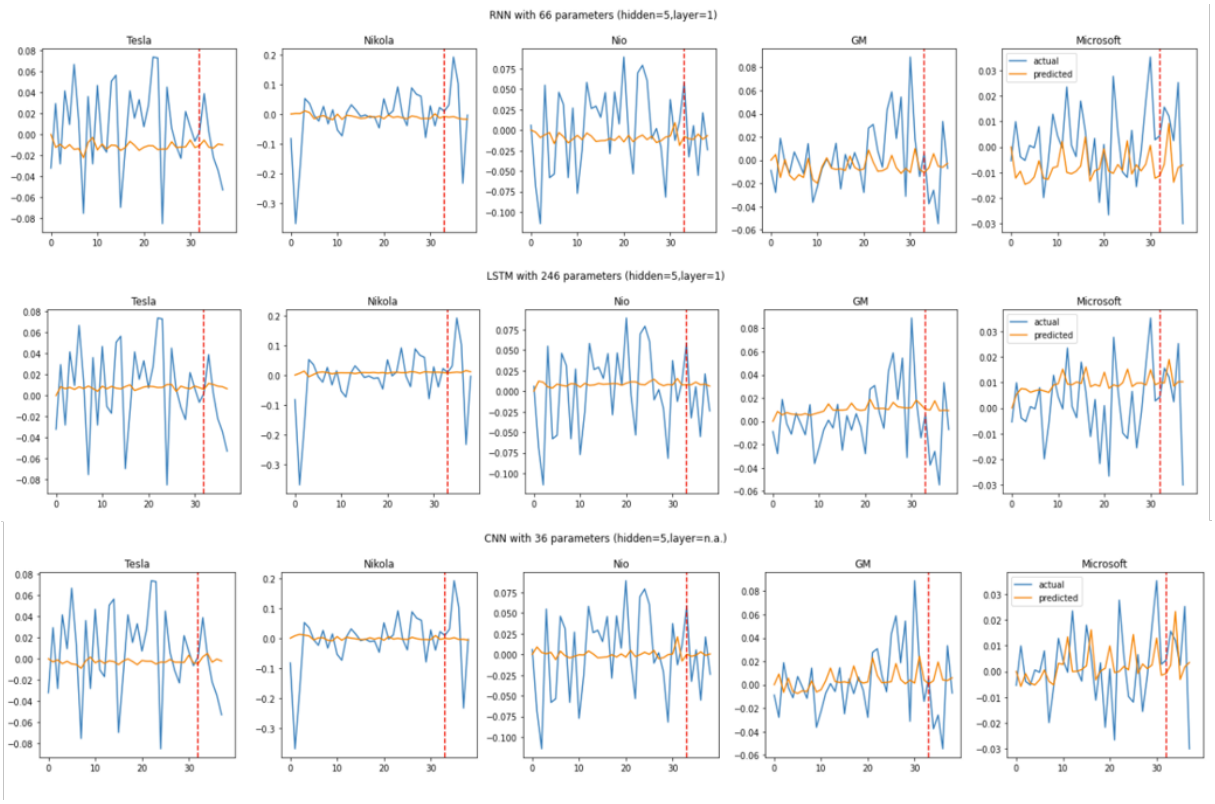

Figure 5: Individual Stock Price Prediction using RNN/LSTM/CNN models

Bo Pang and Lillian Lee. 2008. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2:1–135.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. volume 14, pages 1532–1543.

Saurav Pradha, Malka Halgamuge, and Nguyen Tran Quoc Vinh. 2019. Effective text data preprocessing technique for sentiment analysis in social media data.

Mohiuddin Md Abdul Qudar and Vijay Mago. 2020. Tweetbert: A pretrained language representation model for twitter text analysis.

Dharini Ramachandran and R Parvathi. 2019. Analysis of twitter specific preprocessing technique for tweets. *Procedia Computer Science*, 165:245–251.

Hao Wang and Jorge Castanon. 2015. Sentiment expression via emoticons on social media. *IEEE International Conference on Big Data 2015*.

Katarzyna Wegrzyn-Wolska, Lamine Bougueroua, Haichao Yu, and Jing Zhong. 2016. Explore the effects of emoticons on twitter sentiment analysis. pages 65–77.

Jianqiang Zhao and Xiaolin Gui. 2017. Comparison research on text pre-processing methods on twitter sentiment analysis. *IEEE Access*, PP:1–1.

A Supplemental Material

- Twitter_Sentiment_Training_SUN.html
- BERT_finetuning_SUN.html
- RNN_LSTM_CNN_PricePrediction_SUN.html
- Project_demo_SUN.html

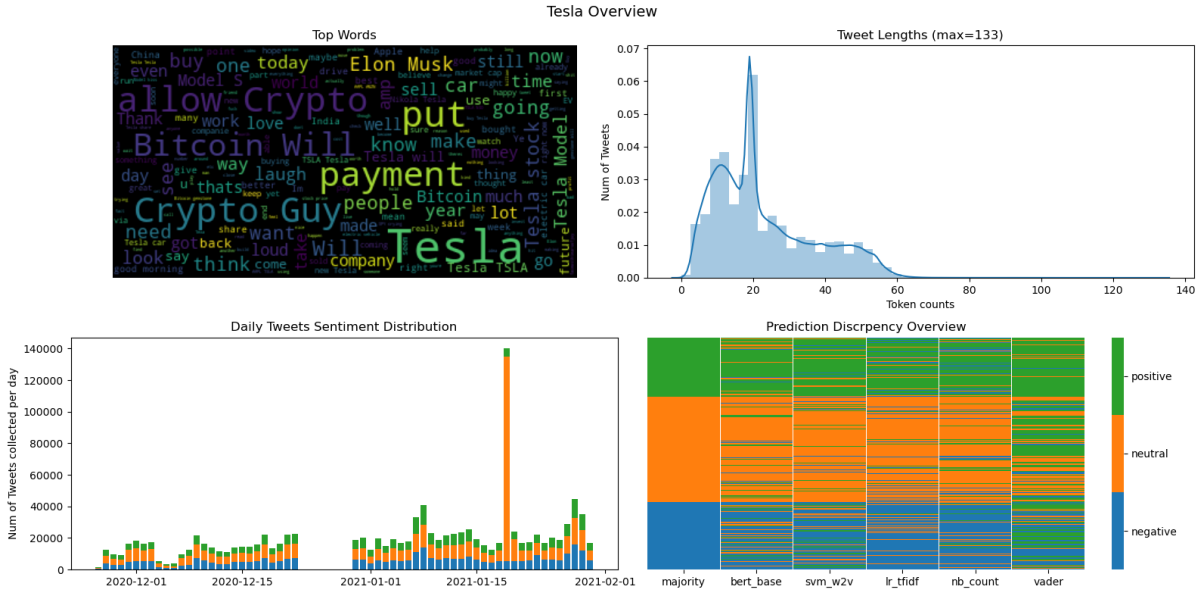

Figure 6: Visualization of Twitter Sentiment on Tesla

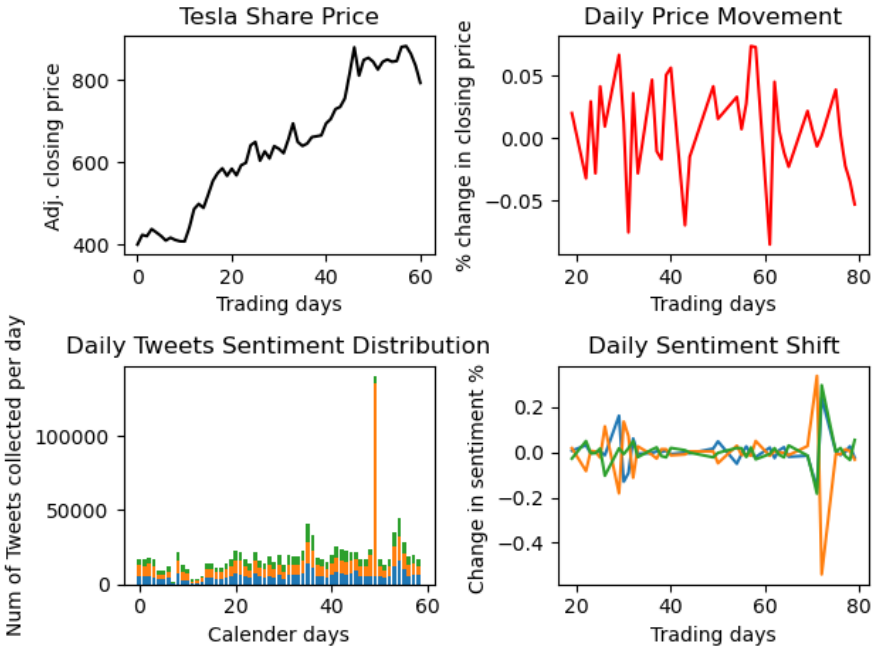

Figure 7: Tesla's Share Price Performance vs Twitter Sentiment Shifts