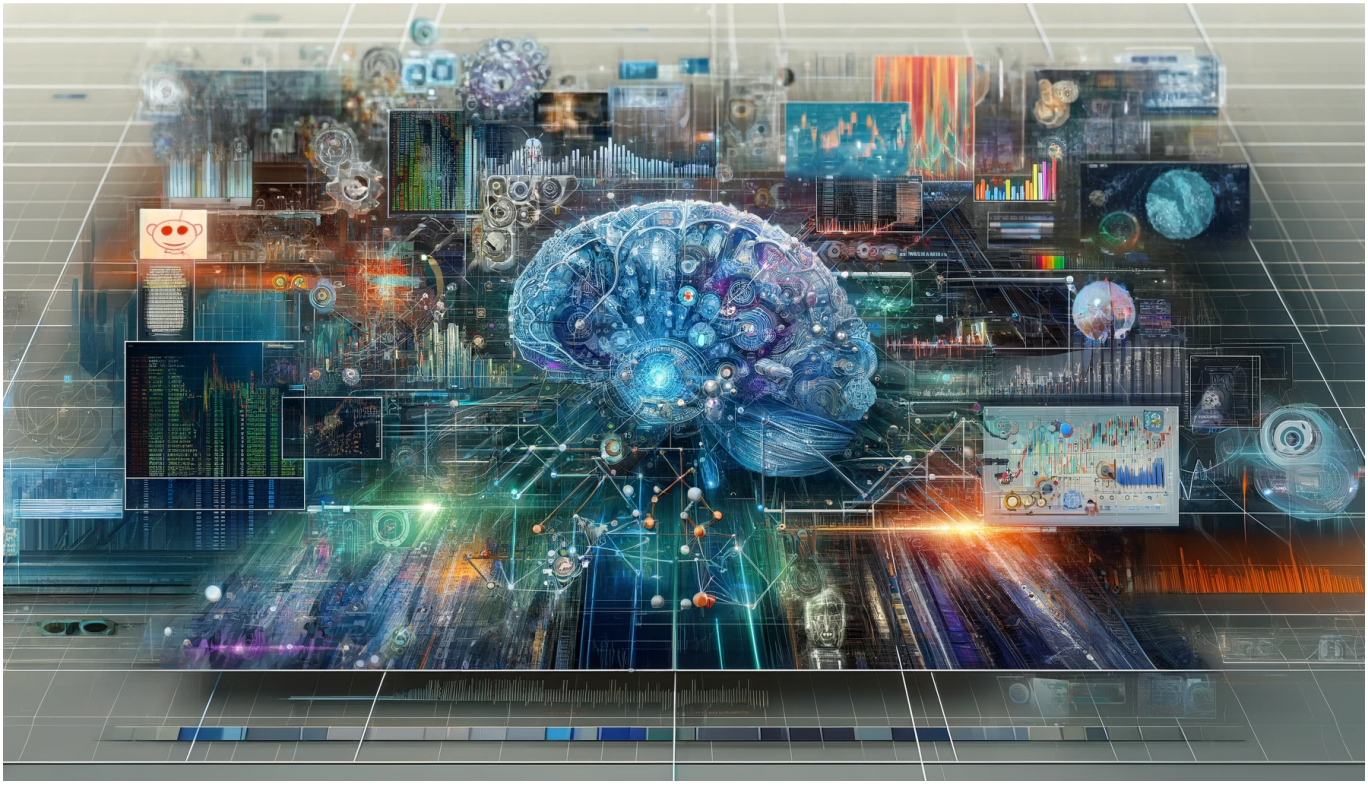
Neural Nets and Deep Learning-Final Project Report

CSC675-Dr. Pauca

Nikhil Rajkumar
rajkn22@wfu.edu
Wake Forest University
Winston-Salem, NC, USA

Raniery Mendes
mendrc18@wfu.edu
Wake Forest University
Winston-Salem, NC, USA

John Billos
billjr20@wfu.edu
Wake Forest University
Winston-Salem, NC, USA



ABSTRACT

This project leverages natural language processing (NLP) and machine learning techniques to analyze and classify posts from the r/WallStreetBets subreddit, aiming to distinguish between informative financial discussions and less substantive posts, colloquially known as "shitposts." [1] Utilizing a dataset extracted from Reddit, encompassing over 300,000 posts filtered by relevance to stock mentions and financial outcomes, we apply a variety of models including Naive Bayes, K-Means clustering, a Simple Feed Forward Neural Network (SFFN), an LSTM, and a Transformer model to predict potential positive returns based on historical data. This study not only contributes to the academic understanding of financial sentiment analysis but also provides practical insights for retail investors navigating crowd-sourced financial advice.

KEYWORDS

Neural Networks, Natural Language Processing, Machine Learning, Text Classification, Social Media(Reddit) Analytics, Retail Investment Strategies

[1]The term "shitpost" is used here to describe content that is typically of low quality or relevance. It is recognized as colloquial and somewhat informal language within online communities. The use of this term is intended for analytical accuracy based on common usage and is not meant to offend or distract from the professionalism of the report.

ACM ISBN …$

1 INTRODUCTION

A brief discussion of the selected problem and dataset.

1.1 Problem Statement

Reddit is a popular social media website with crowd-sourced information. There is a popular Subreddit called WallStreetBets where stock market enthusiasts can share information regarding the markets. However, taking financial information from strangers on the internet could be highly risky.

Consider the natural language processing classification task of distilling two types of sentences into discrete classifications. One type of sentence is humorous where the other type of sentence is non-humorous. How could a model be trained or engineered to tackle this problem? Plenty of models present themselves to handle the language processing task, but the prerequisite task is to answer what makes a sentence humorous and non-humorous purely based on the text without a human reaction to reading it. This project on r/WallStreetBets lends itself to a similar problem. The online forum is filled with retail investors who take pride in well-founded research and information crunching. These retail investors then make a post describing why they are bullish or bearish on the stock with their evidence as to why they believe that. At the same time as these informed retail investors, there are individuals who jump on the "moon" train with posts like "Tesla to the MOON", "GameStop to the MOON", or "DOGECOIN to the MOON!!!" without well-founded financial analysis or research. These types of posts are known as shit posts. Even if the shit posts require little effort to publish publicly, some still focus on a stock or company that grows tremendously. Because of this fact, our project focuses on classifying posts from the subreddit that should produce positive returns from negative returns should the creators of this project start investing.

1.2 Dataset

As this is a deep learning task, a large amount of data is required to train an effective and robust model. This project creates a text dataset instead of using one from Kaggle or another platform. Reddit's API became limited in November 2023 to the point where it is only possible to retrieve the last 1,000 posts from a given subreddit. This amount of data is simply not enough! In fact, our early models trained on 500 of these posts could only achieve an accuracy of slightly above random guessing. This project discovered archived reddit post dumps from every year since its inception. We downloaded 2019-2024 dumps and found all r/WallStreetBets posts. From this data extraction, we discovered over 3,000,000 r/WallStreetBets posts of which over 1.4 million were dated between December 2020 and February 2021 when the GameStop short squeeze was in full effect. From these 3 million posts, we applied a filtering criteria that the post must mention a stock, that stock must have a current price on Yahoo Finance, and that stock must have a historic price on Yahoo Finance around the post submission date (+ 1 week). Using this filtering process 229,144 posts were distilled from the original 3 million. Each of these filtered posts were stored with additional information features: stock symbol, historical price, current price, growth since post, publication date, current date, and label. The label is binary with 1 representing a stock post that resulted in sufficient positive return and 0 representing a stock post that resulted in non-sufficient return. The filter for sufficient positive return is 6% growth annualized with a minimum of 2% growth between the post and the current date. The filter considers 6% growth annualized to be sufficient as it beats the best high-yield saving account offered which is Betterment at 5.5% a year. Using this filter, there are 71,301 posts with a 1 label and 157,844 posts with a 0 label in the dataset. For all unsupervised learning models, the text of these posts is provided as the training data and the labels are used to determine performance metrics.

Figure 1: Good Stock Wordcloud

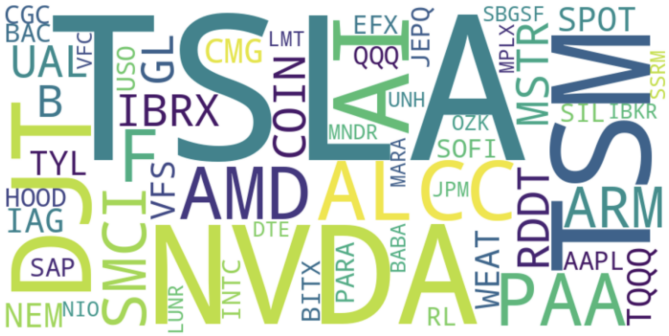


Figure 2: Bad Stock Wordcloud

In terms of EDA, we create wordclouds of the Ticker Symbols for both good and bad stocks. This gives us a visual trend, if any, of well performing and poor stocks. For example, in Figure 1 Tesla (TSLA), Nvidia (NVDA), and Advanced Micro Devices (AMD) appear to be popular stocks with positive returns on Reddit. And in Figure 2, Meta, X, SPY are stocks with bad returns. It is worthy to note that both wordclouds have common stock ticker symbols.

2 LITERATURE REVIEW

The paper "Deep Learning in Finance and Banking: A Literature Review and Classification" by Huang et al. [2] provides a comprehensive overview of how deep learning (DL) technologies are applied in the finance and banking sectors. It thoroughly examines various DL models such as Feedforward Neural Networks (FNN), Multi Layer Perceptrons(MLP), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), LSTMs (Long Short Term Memory RNNs), and Reinforcement Learning (RL), detailing their specific applications in stock market predictions, credit risk assessments, and portfolio management. The review highlights the advantages of DL in handling large datasets and complex patterns, improving predictive accuracy in financial markets. It also discusses the challenges associated with DL applications, including overfitting, data quality, and computational demands. The paper concludes by emphasizing the need for further research to enhance the robustness and applicability of DL models in finance, suggesting

future directions such as the integration of multimodal data and the exploration of more sophisticated hybrid models.

Going a little deeper into the paper, the authors note the following:

- FNNs are highlighted for their basic utility in capturing relationships in stationary data.
- LSTMs are particularly noted for their ability to handle sequential data, making them ideal for the volatile and time-dependent nature of stock prices.
- RL models are applied in scenarios requiring decision-making policies, such as dynamic trading strategies that adapt to market conditions.

We build on the paper by implementing a Simple FNN Network, LSTM (both done in paper) and a simple transformer (extra). This is elaborated on in future sections. The paper highlights how FNNs are good for capturing relationships in our financial data, and therefore that was our first baseline of a neural network implementation. Seeing that this was also a part NLP project (finding patterns in textual posts), LSTMs have good traits of modeling such data better. Finally, we implement a transformer to try something new that the paper does not speak of at all.

3 METHODOLOGY

Our work is all stored on a **Github** repository at:
https://github.com/jrbjrb1212/Stonks-Predictor/tree/main

3.1 Naive-Bayes

We start by creating a bag-of-words representation. We call **Sci-kit learn's** *CounteVectorizer()* on the **Text** column of our dataset. Seeing this is an implementation of Supervised Learning, we use the **Label** column as a reference. We run a 80-20 train-test split. On these splits we train an instance of **Sci-kit learn's** *MultiNomialNB()*. Finally we call *predict()* and *accuracy_score()* to attain and accuracy of 80.16%. Figure 3 highlights a visual evaluation of our model. The confusion matrix can visually show how accurate our supervised learning model models the Reddit posts.

3.2 K-Means

We start by vectorizing the text of the posts in the training set using the *CounteVectorizer()* included in **Sci-kit learn**. The dimensionality of the large matrix is reduced with each row representing a vectorized post using a simple SVD. This is paramount to the training step the text data in the numpy matrix is sparse by different post lengths. We then test the K-Means algorithm, implemented in **Sci-kit learn**, on the truncated matrix with cluster sizes between [2, 10]. We scored each clustering attempt using the silhouette score average with the best score being 0.72. The best score is around 3 clusters and we stick with that, this number was decided when optimizing for parameters, and the silhouette score to number of cluster is in Figure 4. Further metrics and plots are highlighted in the results sections.

3.3 Simple Feed Forward Network (SFFN)

In line with the progression of class, after attempting clustering on the problem we moved to a basic neural network. **Keras** was used to manually create the layers of the model. Unlike class, we had to include an embedding layer for the natural language text input. After the embedding layer, a flattening layer was added to flatten the embedded input vectors. A dense layer of 64 neurons with ReLU activation was included following flattening. The output of the dense layer feeds into a single neuron output layer with sigmoid

Figure 3: Confusion Matrix - Naive-Bayes

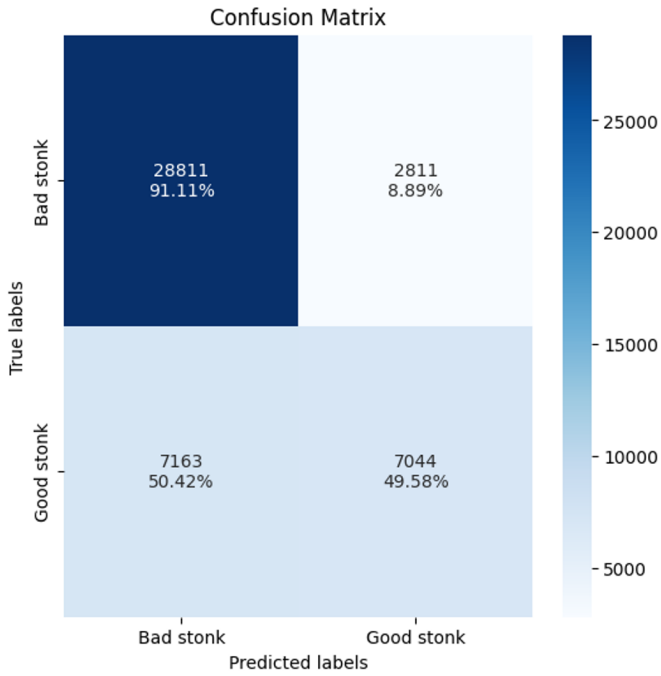


Figure 4: Optimal Clusters - K-Means

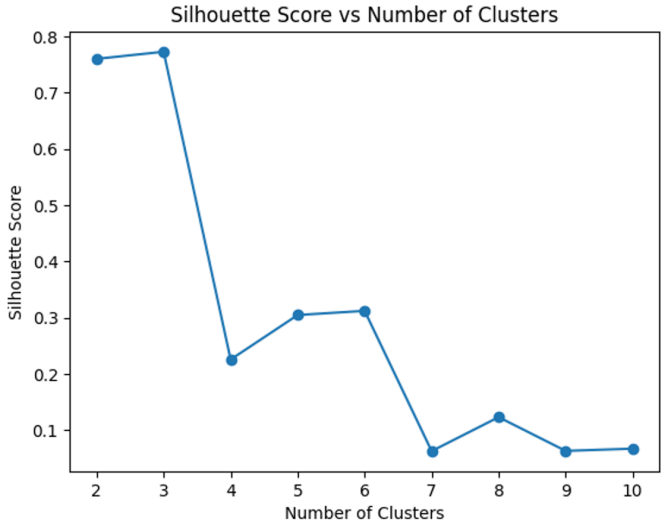


Figure 5: SFFN - Hyperparamter Optimization - Loss Plots

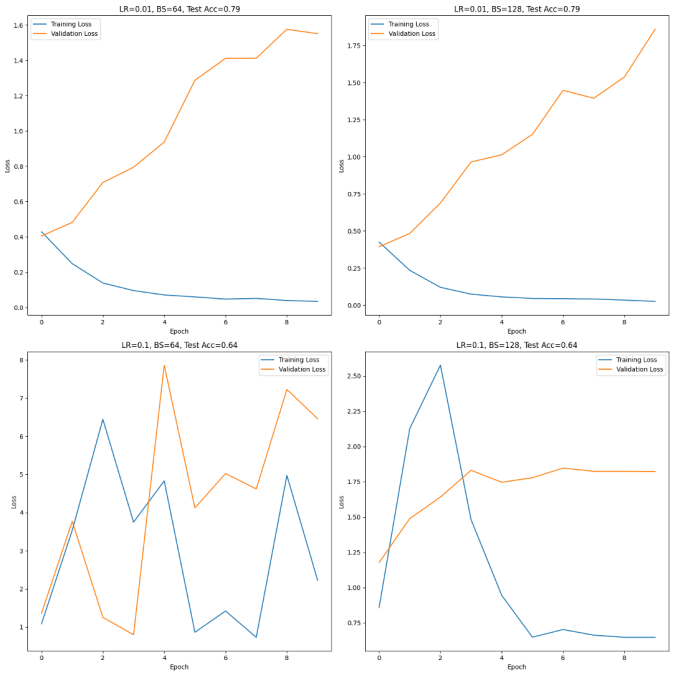
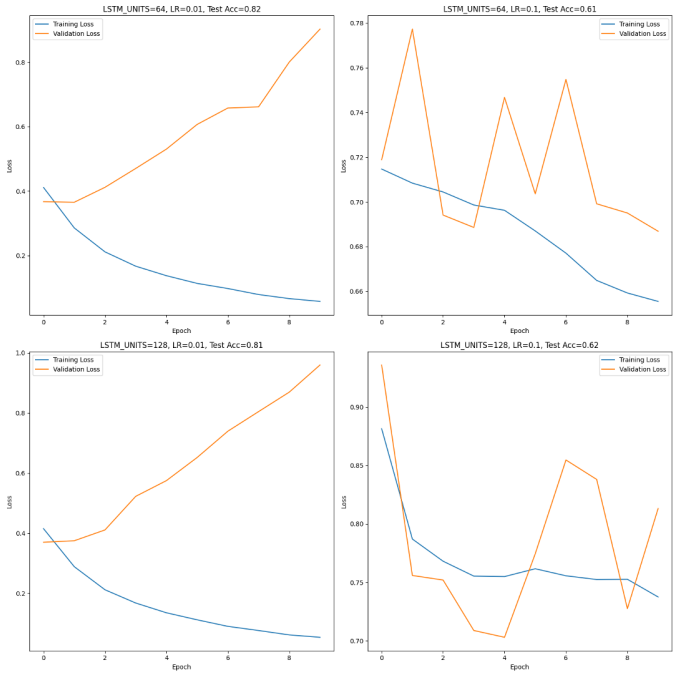


Figure 6: LSTM - Hyperparamter Optimization - Loss Plots



activation that decides if the post is a good or bad investment post. This model architecture was tuned on the hyperparameters of learning rate and batch size. Batch size in this context refers to the number of training samples that are run through the model before weights are updated. Figure 6 highlights the training and validation losses of our models corresponding to learning rates of [0.01, 0.1] and batch sizes of [64, 128] each.

3.4 Long Short Term Memory Model (LSTM)

This model was loosely covered in class. LSTM models are distinct from feed forward networks as they contain gates that allow information to be cycled and recalled in later training iterations. The model relies on LSTM units, which is a process unit in the model that can input, output, and forget data. The units improve the vanishing gradient problem that feed forward networks face by having the forget gate. Our LSTM model is implemented with **TensorFlow**. Our first layer is an embedding layer to embed the raw text input data to a specific vector size (100). The output is fed into a layer of LSTM units that do not have return sequences. The output of the LSTM units layers feed into a single neuron output layer with a sigmoid activation function to decide if the post is a good or bad investment post. Figure 5 highlights the training and validation losses of our models corresponding to learning rates of [0.01, 0.1] and batch sizes of [64, 128] each.

3.5 Transformers

Transformers were not discussed in class. This type of model has been popularized in the last couple of years as they are the foundational training model for large language models. The model type was introduced to solve natural language processing problems such as humor or sarcasm detection that need to weigh different parts of text differently. This weighing is achieved through the idea of attention. In the context of Subreddit posts, some information about the post will be more important than others. For example, an introduction about the company associated with the stock may not be important, but a large customer acquisition may be important. Figure 7 highlights the training and validation losses of our models corresponding to learning rates of [0.001, 0.01] and [4, 6] layers each.

In this report a transformer is implemented using **TensorFlow** and **Keras**. For most applications of natural language processing tasks on classification, transfer learning can be used. This project does used any pretrained models as all the ones tested did not fit within the memory of the hardware we had access to. Instead, we built a smaller transformer architecture from scratch. The model begins with an embedding layer to convert the input text data into vectors that can be trained on. After the embedding layer there is a tunable number of transformer block layers. Each of these blocks contains a multi-head attention mechanism, a feed-forward network with two dense layers, two normalization layers, and two dropout layers. This transformer block design was inspired by OpenAI's ChatGPT and validated by a Medium article [1]. The blocks are stacked sequentially to form the encoder part of the transformer architecture. The output of each block is the input to the next block. The output of the last block is normalized then provided as input to a 1 neuron output layer with sigmoid activation that determines if the post is a good or bad investment post.

Figure 7: Transformers - Hyperparamter Optimization - Loss Plots

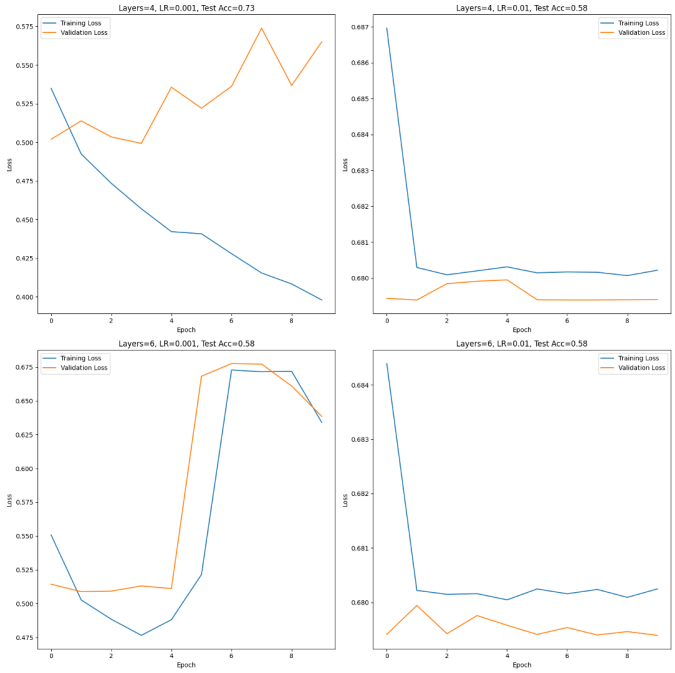
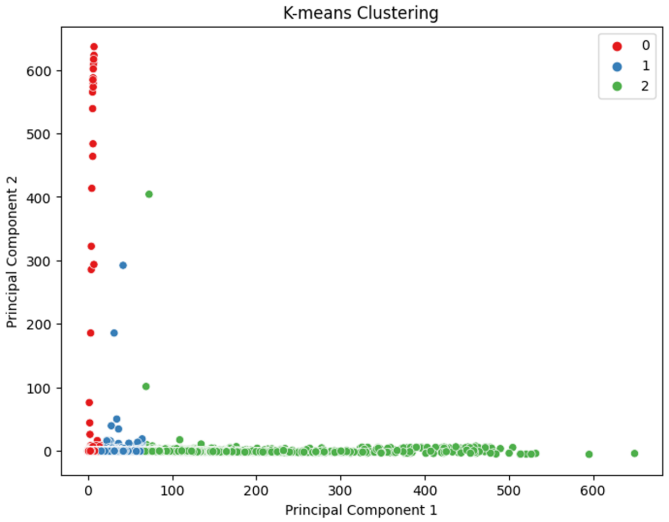


Figure 8: PCA Clusters - K-Means



4 ANALYSIS AND DISCUSSION

4.1 Results of Naive-Bayes

Naive-Bayes was an incredibly powerful deep learning technique to solve the r/WallStreetBets classification problem presented. It takes around 3.5 seconds to train on the **GreenFlash** GPU cluster server. While it only takes 3.5 seconds to train, it produced a test accuracy of 80.161%. The technique routinely produces a baseline for binary classification problems. In this experiment, the bar was set incredibly high. Its great performance can be attributed to the independence assumption or null hypothesis, which assumes that the words of the post are independent of the given class label. In most practical applications, a given r/WallStreetBets post will not affect the stock value of the stock it is about. Given the conditions of the environment, Naive-Bayes is one of the most effective strategies for performing classification of r/WallStreetBets posts. Because the number of bad posts in the dataset was over twice the size of good posts, the model did find a large amount of false negatives.

4.2 Results of K-Means

Compared to the other models trained, K-Means clustering is not performative in this classification problem. The K-Means algorithm is a quick algorithm for clustering, which would make it useful if predicting real time data. However, here we are training models based on historical information that does yield itself to distinct clusters. The submissions in the Subreddit exist along a spectrum with the two ends being the well-research posts and shit posts. However, there are plenty of post types in between that will skew the results of the model. The algorithm's input of tokenized input text makes the algorithm less efficient as the length of the texts vary greatly throughout the dataset. That being said, the unsupervised technique helps us go back to EDA stages to help us understand data better. With a highly dimensional dataset, we conduct Principal Component Analysis (PCA), using **Sci-Kit Learn**, on the dataset to reduce dimensionality, and visualize the clusters. Figure **??** shows the PCA reduced clusters for our model.

4.3 Results of SFFN

The simple feed forward network performs like its name. It is a simple model with simple results. In this case, those results were about as performant in accuracy as Naive-Bayes. The best accuracy on the test set achieved was 79%. This performance may sound disappointing when compared to Naive-Bayes. The feed forward network had a validation data set which had a max accuracy of 81.4%. This accuracy on the validation data is better than Naive-Bayes, which is strong indicator that the Feed Forward Network would perform better if the model was constantly trained with new data. Each network does take around 15 minutes on **greenflash** to train. Compared to the 3.5 seconds of training time that Naive-Bayes takes this model does seem to be a slightly worse option. The hyperparameter testing on the batch size and learning rate provided some useful information that batch size did not significantly affect the accuracy, but the learning rate is incredibly important. Changing the learning rate from 0.01 to 0.1 made the models with the latter rate not readily decrease loss with training data.

4.4 Results of LSTM

The LSTM model was more complex in architecture than the simple feed forward network. This complexity alongside the research that LSTM models perform better well on text data gave it a slight edge of 3% accuracy on the training set and 0.9% accuracy edge on the

validation set. Because of the forget gate in the LSTM units, the LSTM became readily available to discard parts of text sequences that did not strongly contribute to its prediction. All previously discussed models simply cannot do this. The hyperparameter testing on the number of LSTM units and learning rate provided some useful information that LSTM units did not significantly affect the accuracy, but the learning rate is incredibly important. Changing the learning rate did yield non-monotonic decreases in validation loss, but also dropped the test accuracy by 20%.

4.5 Results of Tranformers

During our research period, the transformer architecture seemed to be the most promising in its ability to correctly classify our problem. However, out of all models, the transformer models performed the worst. The best transformer model had a test accuracy of 72.97% and a validation accuracy of 73.46%. This is 6% lower in both test and validation accuracy than the simple feed forward model. In addition to having the lowest accuracies, the transformer models were the largest and most complex leading to the longest training time. Each transformer model took approximately 45 minutes to fully train over 10 epochs on the **greenflash** server. When compared to the training time of Naive-Bayes which is 3.5 seconds with about 80% accuracy this model training seems incredibly disappointing. Creating and training the transformer models taught us that a more complex model does not always lead to a more accurate, robust, or performant model.

4.6 Overall Takeaways

The most performant set of models trained to classify r/WallStreetBets posts as good or bad stock posts was the Long Short-Term Memory models. The competitive edge that LSTM models offer over Naive-Bayes or the Simple Feed Forward Network models is by accuracy percentages of less than 2% in the former and 5% in the latter. Because this type of model could be used in a short or long term algorithmic training bot, the LSTM model should be used. For fast assessments of the online forum, a Naive-Bayes model should be used for its ability to be quickly trained on a large amount of data.

5 ACKNOWLEDGMENTS

We would like to thank Dr. Pauca for his help and guidance. Also, we are highly appreciative of TA Benjamin Hezrony for his assistance to Dr. Pauca and the students. We also would like to acknowledge the use of LLMs (Primarily, OpenAI's ChatGPT) for debugging our code and helping with report writing.

REFERENCES

[1] Tejpal Abhyuday. 2020. Part 1: Transformers - Working of Encoder Block. https://medium.com/@tejpal.abhyuday/part-1-transformers-working-of-encoder-block-6765545081f.

[2] Jian Huang, Junyi Chai, and Stella Cho. 2020. Deep Learning in Finance and Banking: A Literature Review and Classification. *Frontiers of Business Research in China* 14, 13 (2020).