In-Depth Analysis

The goal of the project is to predict stock market movements from market and news data. There are some features that are useful in the model and others that are disregarded because they do not tend to add additional information. It is worth acknowledging that there were several iterations to arrive at the features chosen the hyper-parameters used in modeling however all instances are not discussed. Rather, the final choice of these features and parameters are discussed.

Analysis goal and target

The ultimate goal of this project is to use features from market and news data to predict stock market movement. To do this the desired model outcome is to predict whether the feature r_{ti} =returnOpenNextMktres10 is positive or negative. The Two Sigma challenge on Kaggle asks for the result be presented as a value between -1 and 1. Here a value of -1 means 100% certainty that the target is negative and a value of 1 means 100% certainty the target is positive. Thus, the model outputs a distribution with values $\hat{y}_{ti} \in [-1, 1]$. The scoring metric for this competition is composed of this output distribution and several other values in the market data:

$$x_t = \sum_i \hat{y}_{ti} r_{ti} u_{ti}$$

$$score = \frac{\bar{x}_t}{\sigma(x_t)}$$

where r_{ti} is the 10-day market-adjusted leading return for day t for instrument i and u_{ti} is 0/1 universe feature that controls whether the asset is used in scoring that particular day. Notice, the definition of this scoring metric is essentially a 10-day Sharpe ratio. The larger this value the better the trading strategy.

Feature engineering and word embedding

There were a few simple features that were create directly from features in the market and news data. First, the company names were assigned a numerical value to be used to distinguish companies while modeling. Another feature and a metric commonly used in stock analysis is the log return which is defined as log10(Close/Open) per day. Finally, we take rolling averages over 5 and 10 trading days. The features in which averages are taken are returnOpenPrevRaw10 and sentimentClass. The sentimentClass is considered one of the best measures for judging the sentiment of articles and therefore is considered a good choice for creating more features on. The returnOpenPrevRaw10 was seen to be the most significant feature in previous iterations of modeling and therefore creating more features from it seemed logical.

There are multiple features in the news data that are text-based and therefore cannot be directly used in modeling. Namely, the features subjects and headlines have text information. The subjects have a series of subject codes to indicate different topics within the article. The headlines are the explicit headlines for the articles. To make use of the information content in these features the text is converted to numerical values using a series of analysis steps. First, the information in each feature was vectorized using a CountVectorizer(). This gives the words as word counts. The resulting matrix for the subjects feature was X_subject.shape = (4155955,

1552) and for the headlines X_headlines.shape = (4155955, 167317). This is too high of dimensionality to model practically. Hence, the second step is to reduce the dimensionality of the vectorized text data. To do this the word matrix was broken down to principle components using TruncatedSVD(). The word matrices were then converted to a compact number of features. A total of 10 components for the subjects and 10 components for the headlines was chosen. The individual components total variance ratio is shown in the plot below and, in the legend, the total variance explained by the 10 components used is shown. Thus, a total of 20 new features is added to the modeling data.

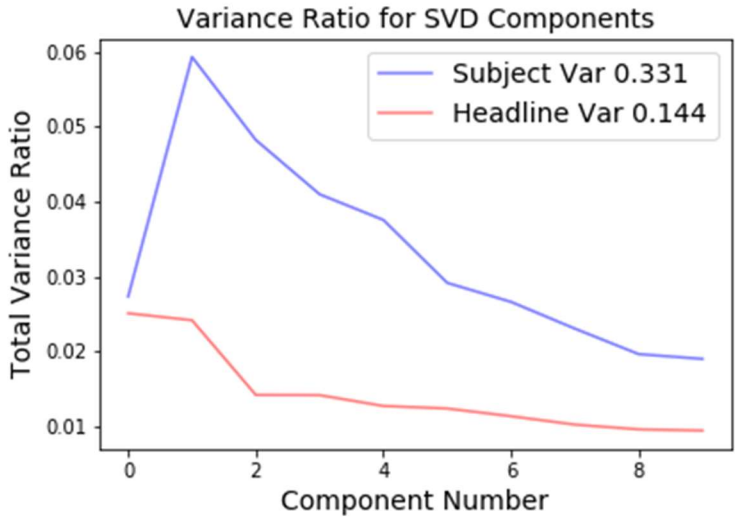

Figure 1 Ratio of total variance for the subject and headline principles components. The legend shows the total variance ratio explained by the 10 components utilized.

Machine Learning Modeling

Several machine learning algorithms were tested and then they were used and ultimately an ensemble approach was chosen. Hyper-parameters for each model were tuned using GridSearchCV() with 5-fold cross validation. There tended to be little (if any) different in the performance when choosing different hyper parameters. The exception was with KNN in which the accuracy score varied with the number of neighbors parameter. Ultimately, the default parameters were used for most other algorithms for the final model because parameter tuning provided little improvement. Here the algorithms that were tested are.

XGBoost : Gradient boosting algorithm that uses decision trees. Made to be efficient and flexible and provides parallel tree boosting.

LightGBM : Gradient boosting framework that uses tree based learning algorithms. Created to support lower memory usage and higher training speed. Sacrifices on some accuracy for speed.

Naïve Bayes : Supervised training algorithm that applies Bayes' theorem with the "naïve" assumption of conditional independence between every pair of features.

KNN : Non-parametric learning algorithm that classifies data into different classes based on the feature space nearest neighbors.

Random Forest : An ensemble learning method for classification that operates by constructing many decision tress and taking the mode of the classes in a classification problem.

The final model chosen was an ensemble model using several of the algorithms above. The final model used XGBoost, KNN and Random Forest algorithms and then used stacking with Logistic Regression as the meta classifier algorithm. Using LighGBM in the stacking method produced errors in running the code therefore this algorithm was avoided during the ensemble method. The table below shows the final results for the accuracy and scoring metric for each model. Notice, the ensemble stacking method produces the best results. The best result for the scoring metric is 0.9486 which is ranked as 37 out of 693 users on the Kaggle leaderboard.

Model	Accuracy	Score
XGBoost	0.5867	0.9241
LightGBM	0.5738	0.8275
Naïve Bayes	0.5052	0.0810
KNN	0.5251	0.4241
Random Forest	0.5299	0.6401
Stacking	0.5885	0.9486

Figure 2 Accuracy and competition scoring metric for different models used. The stacking model uses XGBoost, KNN and Random Forest then a Logistic Regression for the meta classifier.

Ensemble Stacking Classifier and Feature Importance

The final model chosen takes XGBoost, KNN and Random Forest then uses a Logistic Regression algorithm as the meta classifier. The resulting confidences from the test data are plotted in the histogram below along with an ROC curve for the stacking model. Notice, the auc for the stacking algorithm of 0.588 is actually worse than that of XGBoost alone, 0.624. This is just due to the nature of how the algorithm operates. Since XGBoost also gives a good accuracy and score it is beneficial to look at additional outputs from this algorithm. In particular we can look at the feature importance from the model to see which features played the most important role in training.

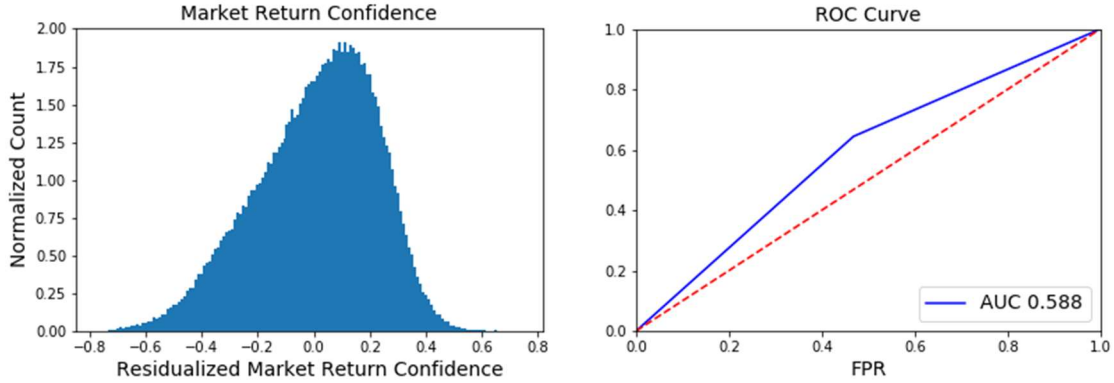

Figure 3 Left: confidence ratio for 10-day market return. Right: ROC curve for stacking model.

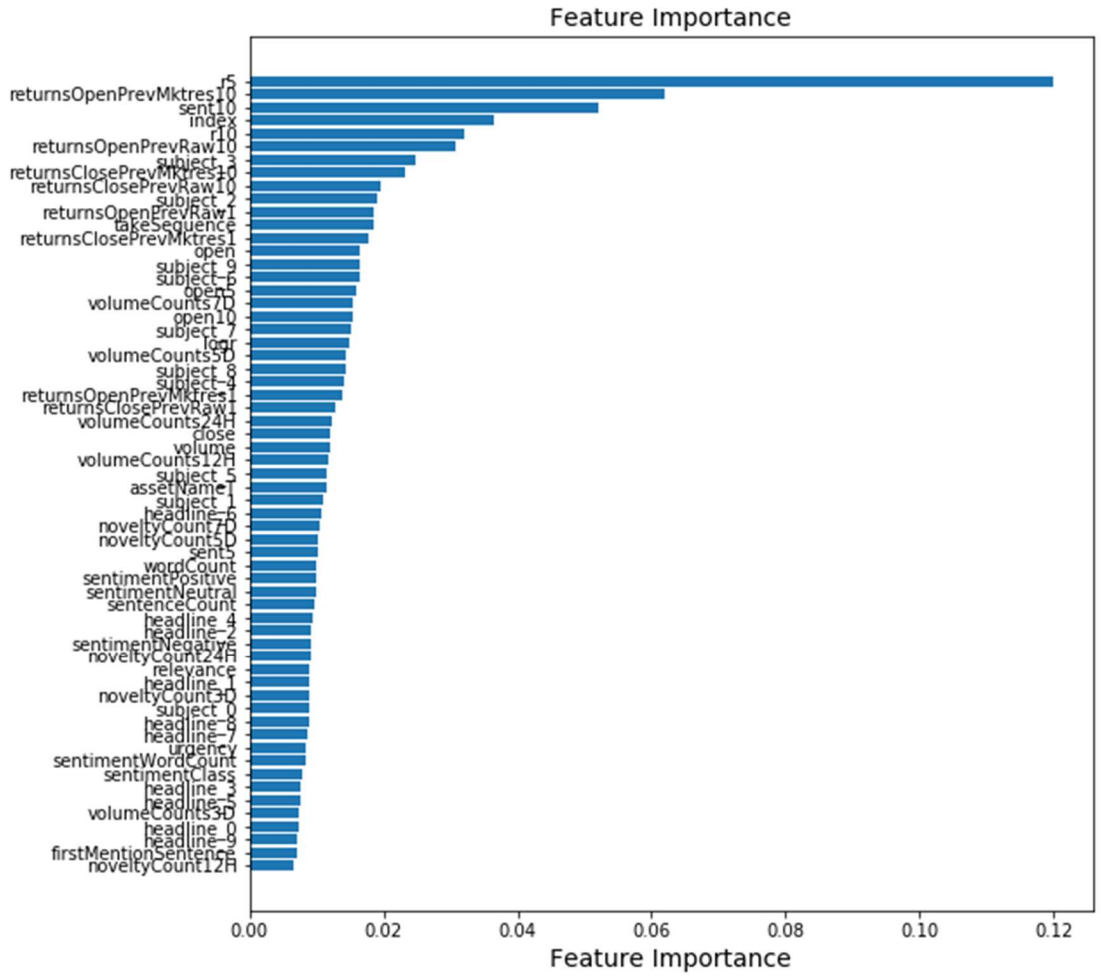

Figure 4 Feature importance during training of XGBoost algorithm.