Introduction

Our goal for this project was to construct a binary classification model using information from Twitter tweets as well as historical stock price data in order to predict future stock price movement direction (increase/decrease). We defined the target time scale for stock price changes to be weekly movements measured from market open to the next market open, and focused our models on 5 stocks/companies: Apple, Google, Tesla, Amazon, and Microsoft.

Data Sources

Historical Stock Price Data

Yahoo Finance provides access to historical weekly summaries/statistics on stock price data for all of our stocks of interest, including weekly opening and closing price, as well as high and low. For our desired binary label of price movement direction from open to open, this dataset is well-balanced, indicating that no special sampling techniques are required.

Tweet Data

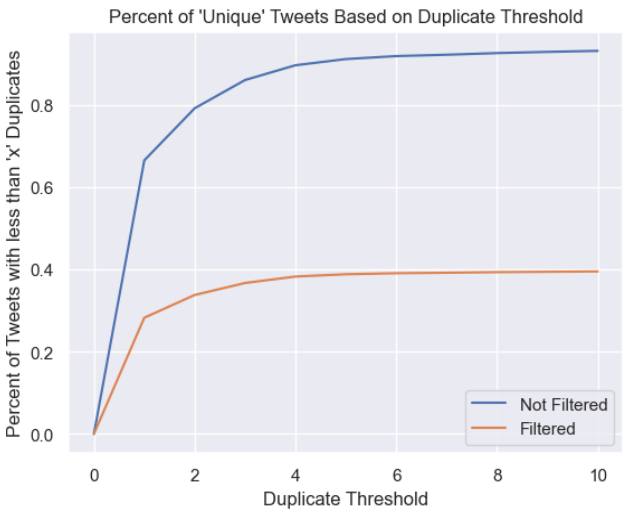
Kaggle provides a dataset containing data on over 3 million unique tweets about top companies from 2015 to 2020, collected from Twitter and gathered through a Selenium-based parsing script that searches for keywords related to these top companies. Features included in the dataset are tweet id (assigned by Twitter), account name of the author, post date (seconds since epoch), tweet text, as well as number of comments, likes, and retweets on the tweet.

Regarding exposure metrics (likes, retweets, comments), the logic is that a tweet with a high amount of exposure should generally be a better estimate of the general sentiment. In our data set about 60% tweets have 0 comments, 0 likes and 0 retweets. Tweets with high numbers on these three metrics are logarithmically rare.

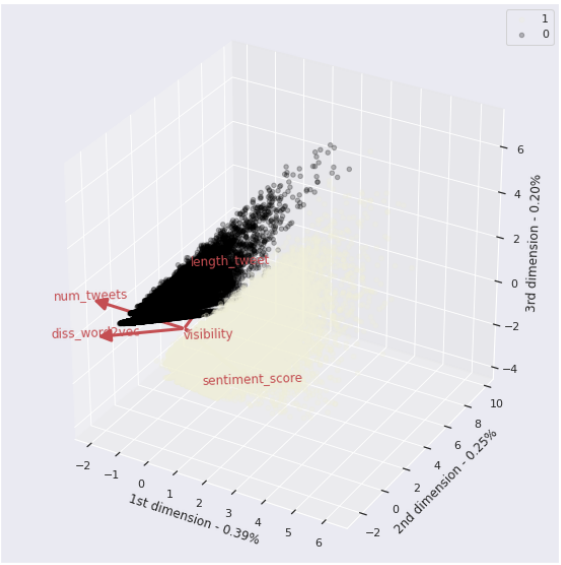
From the correlation matrix, we can see there is a relatively stronger correlation between likes and comments, and between likes and retweets.

	comment_num	like_num	retweet_num
comment_num	1.000000	0.568534	0.351800
like_num	0.568534	1.000000	0.429524
retweet_num	0.351800	0.429524	1.000000

For spam tweets, we needed to filter them out so as to skew the dataset with similarity. A method of filtration we did is to filter by uniqueness. This is due to many spam accounts tweeting the exact same tweet multiple times. With this we can use a duplication threshold, which is the number of duplicates that will classify a tweet as a "bot tweet", and filter it out. However, some news sources do tweet the same thing multiple times, as can be seen in our EDA. One case is the WSJ tweeting a news headline across all their different twitter accounts. The graph on the right shows how the duplication threshold affects our filtration (blue line). The orange line shows filtration with both exposure filtering (having more than 0 in each metric) and



duplication filtering. The result should be a dataset that is rich in exposure, and not a spam or duplicated tweet.



We also conducted an unsupervised analysis to classify the writers. The idea would be that based on some textual features and tweeter features we could determine in an unsupervised way, if the user is a bot or a real user. To do so, for each user we computed their average influence (which is the sum of comment/like/retweets of their tweets), the log number of their tweet, the log length of their tweet, the average sentiment score of their tweet and the average co-similarity between their tweets. We therefore end up with 5 features which we project in a 3 dimensional space thanks to a PCA. We cluster our twitter users based on these features using KMeans. The dark dots would correspond more to the bot user and the white dot to the real user.

Sentiment analysis showed that removing bot data has a tremendous impact on the sentiment and price relationship, therefore the previously mentioned filtration is a crucial step to remove noise in our dataset.

We also did word analysis. The top-10 most correlated words from Bag of Words. Each week corresponds to an observation and the features are the frequency of appearance of the word composing our vocab in the tweets for the respective month. Color shows correlation between the frequency of that word and the target variable. An example is the graph to the right.

Data Cleaning & Feature Engineering

We remove tweets that are likely to be generated by robots or spams based on similarity among tweets of a Twitter account. We only kept tweets where comments, likes and retweets are greater than 6 to ensure some social exposure and are not duplicates. This is highlighted above.

For the tweet texts, after preprocessing them by removing stopwords and tokenizing, we need to transform them into a vector that can be interpretable for our ML model:

- The first way to do so is to build a Bag of Word with a vocabulary size of 10,000. We have been creating a Cython function for this task that allows us to build the BoW in less than 8s for the 1M tweets. Then we encode our text : each token is replaced in the sentence by its rank of apparition in the BoW.
- The second way to do so is to train a word2vec embedding to map each encoded word into a vector based on its context. In our case, we build and train a Continuous Skipgram word2vec following the tensorflow documentation on a subset of our dataset.

Once we have our encoded tweets, we need to apply a transformation to reduce the batch of tweets texts to a vector of features. Indeed, for each week, we have *N* tweets encoded as a *seq_len* vector. We can either take the average value of each word over the tweets to end up with *seq_len* features or count the frequency of apparition of each word in the batch of tweets to end up with *vocab_size* features. One of these batching methods is used to featurize the text data.

Thus our set of data is composed of our vectorized text data, auto-regressive features (the state of the target at time t-1,..., t-k when predicting the state of the target at time t), and a selection of tweet features such as the number of comments, number of likes, and number of retweets. We also add the sentiment score of each tweet to our set of features.

This feature construction is specific to the machine learning model that is limited to taking text features as inputs. We detail the featurization for the deep learning models in the next section.

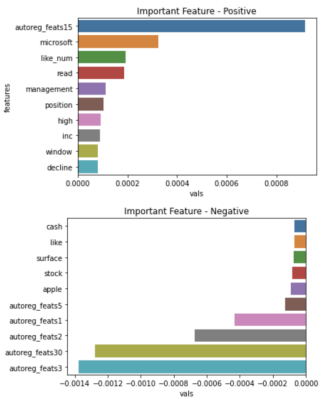
Model Explanation & Performance

Machine Learning Models

The features that we engineered as the input for our machine learning models are mainly autoregressive features, visibility features (number of retweets, number of tweets, unique users, etc.) and text as explained in the previous section.

We find that interpretability is very important for our model, hence we decided to use the Tuned Logistic Regression and Random Forest model to find the best performance of our model. We were also interested in the importance of features for each ticker hence we trained the model individually for each one of them.

Since we are doing a binary classification task, we fit a logistic regression model first to see baseline performance. The best performance validated on the train/test dataset is 0.69 on average. Then we tried tuning the logistic regression model and found that the performance remained constant at 0.69. We also tried Random Forest, after tuning the hyperparameter, 0.68 is achieved. We can conclude that in this case, our baseline models perform best.



Another observation that we saw is that text features seem to influence the outcome the most based upon its importance. We do, however, see that in one of the best performing model, MSFT with accuracy of 0.72, autoregressive features dominates top importance features

Deep Learning Models

For NLP tasks, it has become a convention to rely on deep learning models instead of machine learning models. Indeed, they automatically do the feature engineering for us and are more flexible. In this section, we will detail how the training data set is built to train such a model and which model is used.

Similarly to the training dataset used for the machine learning models, we will use the text data and some autoregressive features as our inputs. For this type of model we don't reduce the dimension of our batch text data (which is a matrix of size *N* x *seq_len*). Instead we will encode each tweet in the batch of tweets via an encoder (in our case it would either be a Bi-LSTM or the output of BERT encoding) resulting in a matrix of size *N* x *encoding_dim*. Then we will apply an attention layer to reduce the number of tweets on which our model should focus on resulting in a *L* x *encoding_dim* matrix.

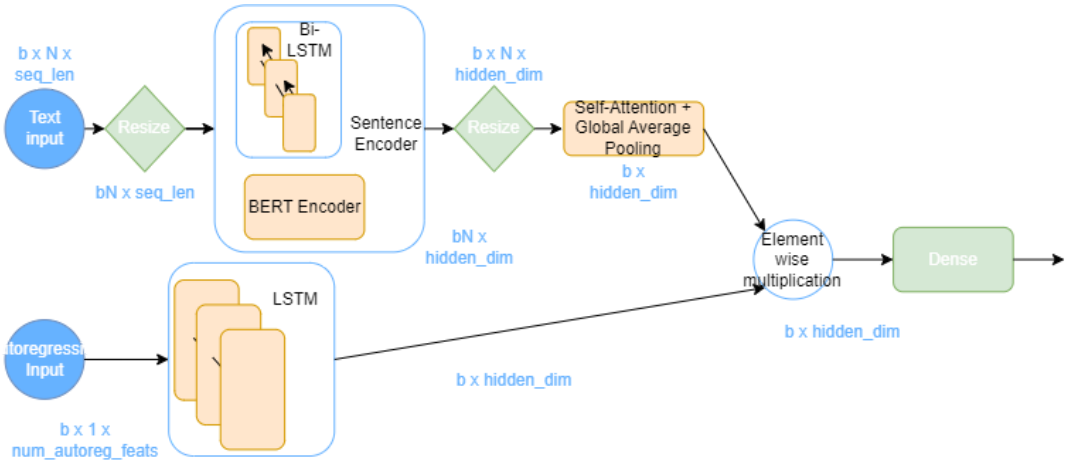
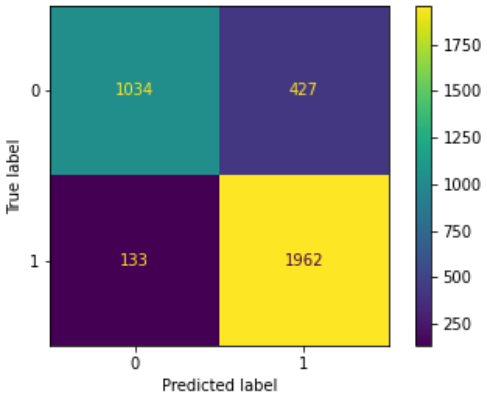


Figure 3 : Architecture of the Multimodal Attention LSTM

Since our GPU has a limited RAM, we split our weekly batch of tweets into sub-batch of the same dimension by padding with null sentences (sentence with only 0 in it). That is to say if we have a batch of tweets of 2250 on week *t*, we split into 5 sub-batch of 500 tweets where the last one has 250 null sentences (that's where the attention mechanism comes handy).

The autoregressive features are encoded by an LSTM and we merge the two encoded inputs via either an element wise multiplication or a concatenation. Then we add a classification head to finish our model. Built this way, our dataset is slightly imbalanced with 35% of negative value and 65% of positive value.



After training for 5 epochs, with a batch size of 32, a sub-batching size of 250 tweets and an attention to only 64 tweets, we end up having around 84% accuracy on our validation/test set and as well as 82% precision. In our implementation, the BERT encoder wasn't able to fit on training data since after 50 epochs of training on one batch it didn't manage to reach 100% accuracy.

Authors: Maximo Oen (MMO2134), Micol Clement (MC5104), Anindra Iswara (AI2462), Shengyuan Cao(SC3912), Eric Wu (EJW2169)