Predicting Stock Movements Based on News Sentiment Using NLP

Problem statement:

Our objective was to use sentiment scores from financial news headlines – extracted through natural language processing (NLP) – and other features outside of historical stock price trends to predict next-day stock price direction (up or down) with at least 60% accuracy.

Traditionally, stock prices are analyzed using time-series data and technical indicators, however we know that in practice markets often react to qualitative news content: mergers, leadership changes, economic events, and more. The rise of NLP therefore presents an interesting avenue to exploit this information. Our project diverges from conventional models by using NLP to extract sentiment from news headlines to predict market behavior.

Plan:

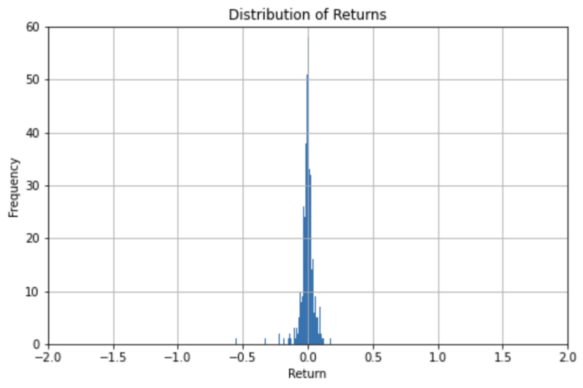
To say that we "succeeded" in our exploratory task, we want to find at least one working model that is capable of classifying news headlines as indicators of stock price increase or decrease (by the next trading day), and able to do so with accuracy above random chance (which let's say is approximately > 60-65%).

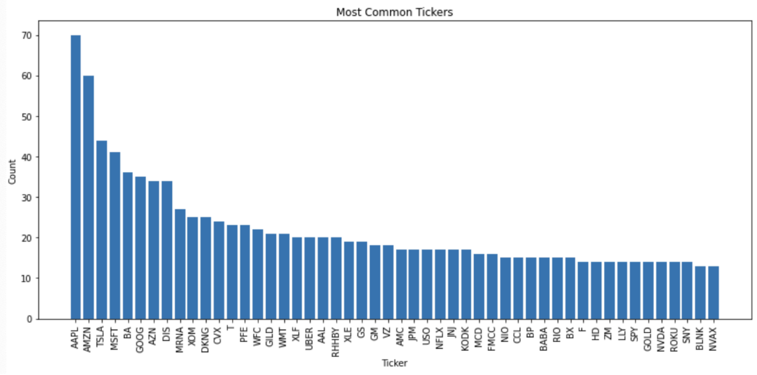
The dataset which we work with (Kaggle: "Financial News Headlines + Tickers") here contains news headlines with their associated dates and tickers (connecting the headline to the associated company). We then use the "yfinance" library to fetch market data from Yahoo Finance, and find the stock price changes (returns) of the next trading day which followed the publication of each headline. We'll note that our dataset only contains headlines from July and August 2020. Potential constraints include: the subjective nature of "sentiments", random stock price movement, and the fact that our dataset is pretty small: might be difficult to discern true changes in stock as a result of headlines vs. random movement.

Data Wrangling & EDA:

We should note that we lost a significant portion of headlines from our original dataset when trying to connect stock returns to each headline, but nonetheless we managed to connect the vast majority: retained 10,791 out of a total of 14,088.

We also made sure to remove large (impossible) outliers, and performed general sanity checks by looking at the distribution of returns and of company tickers. We were comforted to see that the distribution of returns is well-centered around 0 and that there isn't a single company which is much more frequent than the others.





Pre-processing & feature engineering:

With the relatively small dataset size, we tried to add as many numerical features as possible (with some from NLP as well), which are meant to be simple and interpretable. The non-NLP features which we added were:

- Headline length (characters)
- Word count
- Day-of-week dummies (dummy variables for each day of the week)

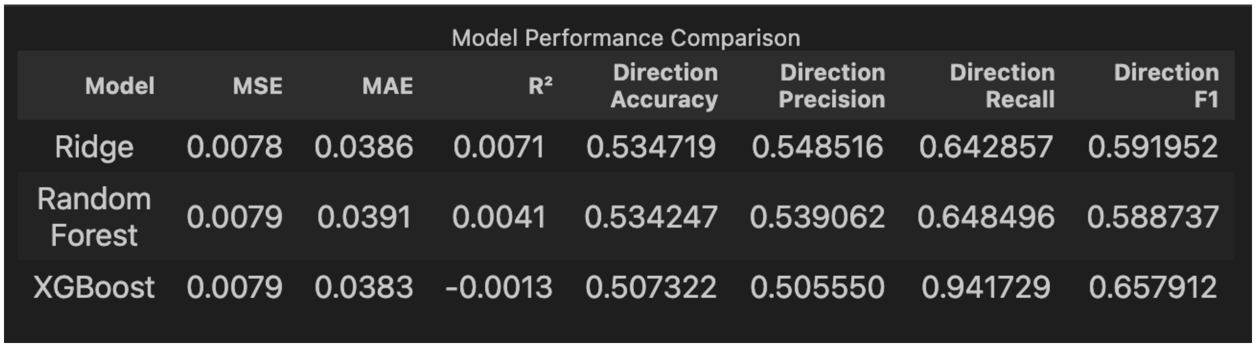
NLP features:

- VADER sentiment scores: model assigning polarity scores (positive, negative, compound) to short texts, designed to work well on headlines.
- FinBERT sentiment scores: model pre-trained on financial text, classifying sentences into sentiment categories (positive, negative, neutral), designed for financial contexts.
- TF-IDF features: Numerical representations of most important words in headlines, where each word's weight reflects how unique it is compared to other headlines.

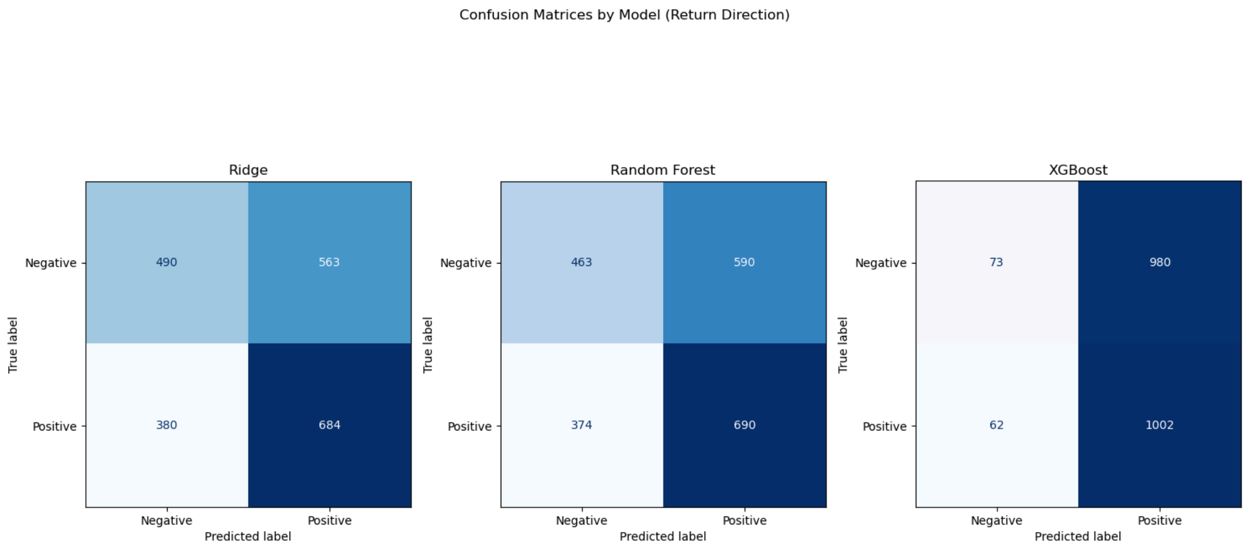
Modeling:

With our data cleaned and having additional features, it was time to model. We opted to try three types of models: Ridge, Random Forest, and XGBoost. Since Ridge only uses one hyperparameter (alpha), we used a Grid Search CV for hyperparameter tuning since we can exhaustively test a small range of values. Unlike Ridge, the other two models have many more parameters [(Random Forest: n_estimators, min_samples_spit, min_samples_leaf, max_depth), (XGBoost: subsample, n_estimators, max_depth, learning_rate)], so we instead use Randomized Search CV which is more practical to find strong configurations quickly. We'll also note that we only scaled the features which required it (such as headline length, word count, and sentiment scores), leaving dummy variables and TF-IDF features alone.

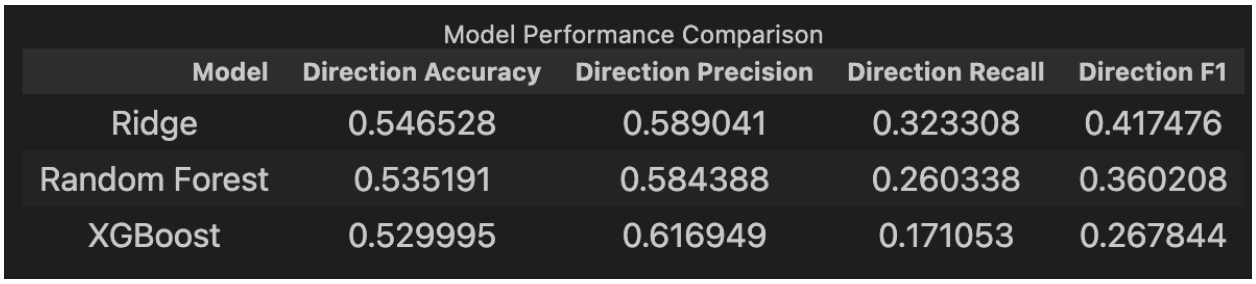
1. At first we approached this problem like a regression problem, where our target was the raw return. However our models' performance for this was barely above random chance, and we got some pretty strange behaviors overall (such as XGBoost predicting nearly only positive returns, and generally disappointing metrics):


Model Performance Comparison

Model	MSE	MAE	R²	Direction Accuracy	Direction Precision	Direction Recall	Direction F1
Ridge	0.0078	0.0386	0.0071	0.534719	0.548516	0.642857	0.591952
Random Forest	0.0079	0.0391	0.0041	0.534247	0.539062	0.648496	0.588737
XGBoost	0.0079	0.0383	-0.0013	0.507322	0.505550	0.941729	0.657912

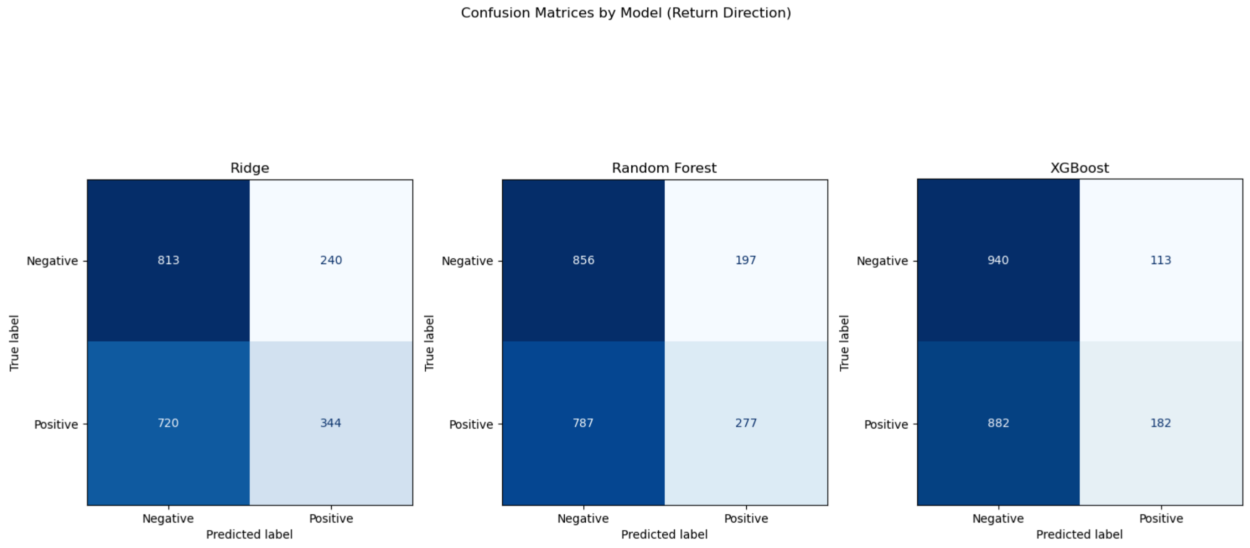


2. We then opted to treat this problem as a classification problem, aiming to look at whether the return is positive or negative. This gave slightly improved results, but still nothing outstanding, and we still got this weird behavior with several models (in particular XGBoost) where there is a heavy tendency to predict largely in one category (in this case negative). At this point, it started to look like the returns might just be too small to be distinguishable from random market noise, and that our model is essentially being asked to predict randomness.

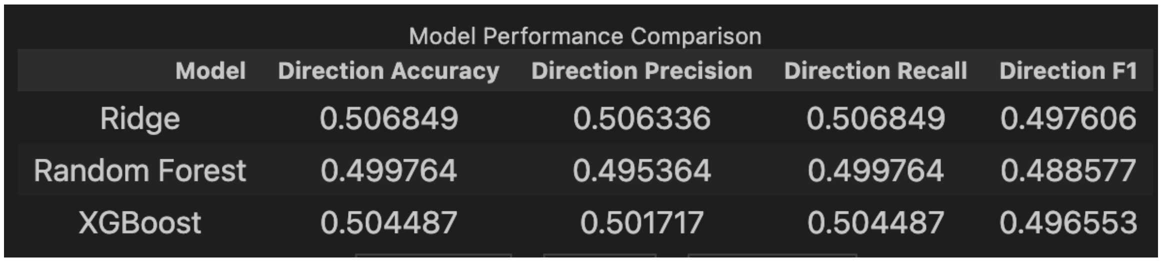


Model Performance Comparison

Model	Direction Accuracy	Direction Precision	Direction Recall	Direction F1
Ridge	0.546528	0.589041	0.323308	0.417476
Random Forest	0.535191	0.584388	0.260338	0.360208
XGBoost	0.529995	0.616949	0.171053	0.267844



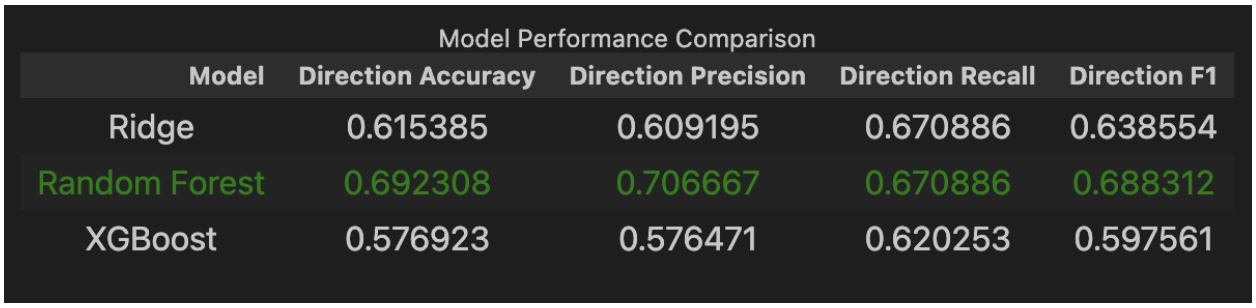
3. To counteract the random market noise, we then thought about making a third classification category: the “neutral” bucket. Although this was a good idea, it actually just made performance worse, because we introduced yet another boundary for the models to learn.



Model Performance Comparison

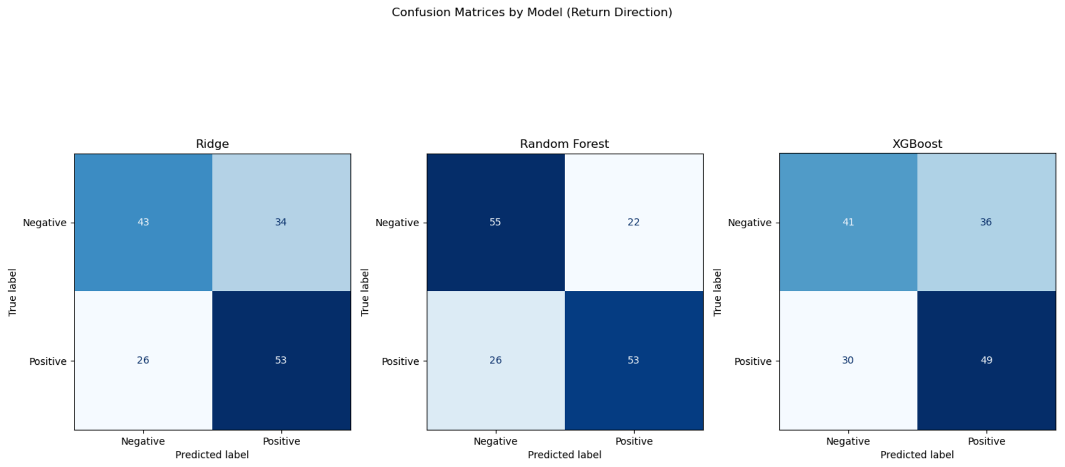
Model	Direction Accuracy	Direction Precision	Direction Recall	Direction F1
Ridge	0.506849	0.506336	0.506849	0.497606
Random Forest	0.499764	0.495364	0.499764	0.488577
XGBoost	0.504487	0.501717	0.504487	0.496553

4. Finally, we came across the best set-up by deciding to completely omit the “neutral” bucket, and making its size quite a bit larger, such that we really are only looking at returns that are significantly large in the positive or negative direction. In some sense, we are attempting to completely eliminate random market fluctuations, and let our model make predictions on headlines which may have caused big changes. This gave us great performance, reaching up to ~ 70% accuracy for Random Forest in particular.



Model Performance Comparison

Model	Direction Accuracy	Direction Precision	Direction Recall	Direction F1
Ridge	0.615385	0.609195	0.670886	0.638554
Random Forest	0.692308	0.706667	0.670886	0.688312
XGBoost	0.576923	0.576471	0.620253	0.597561



Most important Random Forest model features were:

1. FinBERT positive sentiment
2. FinBERT neutral sentiment
3. Headline length
4. FinBERT negative sentiment
5. VADER neutral sentiment

It’s clear that we saw massive improvements with this set-up and are predicting well-above random chance. Also it is promising that the most important predictive features of our best performing model (Random Forest) was largely the sentiment scores (particularly FinBERT), indicating that our model is tackling our problem in the intended way: using NLP sentiment scores, particularly those geared towards financial contexts, to predict directionality of stock returns!

Further research:

- Expand dataset size: collect more historical headlines and corresponding returns since tree-based models (like our best performing Random Forest model) thrive with more data.
- Refine text representation: experiment with richer NLP embeddings rather than just TF-IDF.
- Compare with further models: try other ensemble methods.
- Alternative targets: can test predicting volatility, abnormal returns, etc.

Recommendations for use:

- Integrate directional predictions as a signal: use the model's ~70% accuracy to guide short-term trading signals (ex: filter trades where the model shows strong confidence in up/down moves)
- Combine with human expertise: deploy the model as a decision-support tool, not a fully automated system – analysts can then cross-check flagged headlines to prioritize attention
- Iteratively improve & monitor: start with deployment on a limited portfolio, track real-world performance, and update the model regularly with new headlines and returns.