Problem

Understanding the financial market has always been an interest of mine. From trying to identify stock patterns to analyzing company financial statements, the thought of knowing the strength of a company through finance really caught my attention. Having some understanding of what a strong company looks like financially, I wanted to look into alternative ways to try and understand the health and strength of a company. In order to create a tangible project that was aligned with my interests and provided a sufficient challenge for me to grow in my knowledge of Data Science, I decided to look into Natural Language Processing. The goal of this project is to build out a model that scrapes the web for articles prior to a company announcing their earnings numbers. Each quarter, Wall Street Analysts set estimates for what they believe the earnings of a company will be for the upcoming quarter. In this model, I will be using Natural Language Processing techniques as well as a sentiment analysis algorithm to analyze my data and create features that can be used within a classification model to try and predict whether or not a company will outperform or underperform their earnings estimates.

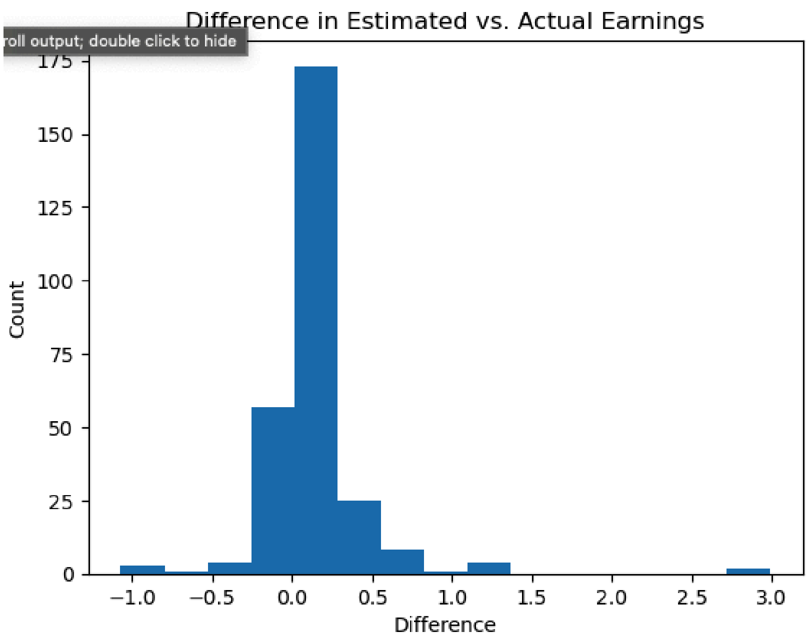
Data

I pulled most of my data from the AlphaVantage API which gave me access to the earnings reports for select companies dating back to the 1990s. The API also gave me access to News Articles which could be filtered by both company and keyword. My initial idea was to pull earnings reports and articles from 5 different companies and research 10 years worth of earnings reports along with approximately 100 articles for each quarterly earnings. However, I quickly found out the News API from AlphaVantage only contained articles from the last two and a half years, allowing me to only research 10 quarterly earnings reports and their associated news articles. Due to this, I decided to focus on the tech industry and pick 30 companies from the top 50 to pull data from. The criteria used to pull the articles included the ticker for the company, the keyword “earnings”, and a date range that covered the 10 days prior to a company issuing their earnings report. The News API I used only returned the title, url, and a brief description of what the article was about. To create my own NLP sentiment analysis, I would need access to each of the related article bodies from their respective websites. This gave me the opportunity to employ web-scraping techniques. I created classes for each website and used the BeautifulSoup library to carefully pull the article bodies without flooding the domain with excess requests or erroring out from various response issues. This process allowed me to pull 10,290 articles (9,604 of which were valid) in a relatively short amount of time. Unfortunately, two of the companies pulled (‘SONY’, and ‘CRM’) did not contain any News articles through this API so the final number of companies tested was 28. I then built out a function for cleaning and preprocessing the articles so they could be processed through some NLP algorithms for further analysis. I used both TFIDF Vectorizer and CountVectorizer to analyze the articles and begin drawing conclusions.

Exploratory Data Analysis

The target variable for my report is the “surprise” column in the earnings data. This represents the difference between the estimated earnings and the actual earnings reported by a company. Figure 1 below shows a distribution of these differences. Any positive number represents a company beating its earnings estimate.

Figure 1:



From this distribution, I can conclude that more often than not over the last 2 years, companies in the tech industry have slightly outperformed their earnings estimates. After further review, I was able to confirm that 86.3% of collected earnings were either at, or above their estimates. With this in mind, I moved on to cleaning and preprocessing the articles. I removed numbers, punctuation, emojis, and lemmatized each article to get it in its raw form. I then ran the articles through both a TFIDF vectorizer (removing stop words), as well as a CountVectorizer (also removing stop words). After vectorizing the articles, I began checking for words that appeared the most as well as words that showed the highest TFIDF score. Figures 3 and 4 below show bar charts of the top 10 words from each matrix:

Figure 2:

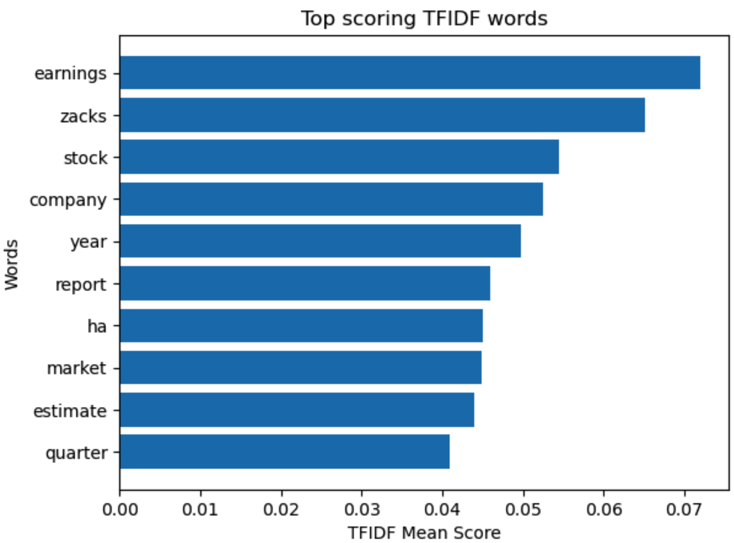
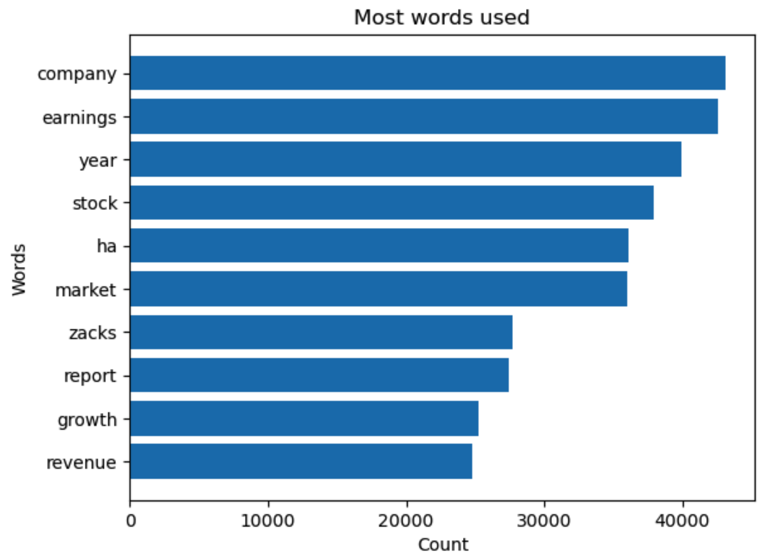


Figure 3:



It became apparent to me that just pulling the top scoring words or the words with highest count would not be the best approach. Additionally, as I began filtering the words by the companies the articles were about, each company name was coming up with high importance so I decided to filter down the words to look for. I went through the top scoring words and created an array of words that scored in the top 300 that I believed could be predictors as to whether or not articles were referencing the company in a good light. Additionally, my hope was that these could be used to determine if the company was going to outperform their earnings estimate. Below is the list of words I came up with:

Keywords:

['buy', 'sale', 'strong', 'beat', 'higher', 'increase', 'positive', 'gain', 'lower', 'trend', 'big', 'bullish', 'increased', 'potential', 'valuation', 'cost', 'decline', 'good', 'bearish', 'sell', 'major', 'loss', 'profit', 'low', 'growing', 'indicating', 'despite', 'leading', 'momentum', 'exposure', 'rising', 'grow', 'rise', 'solution', 'bad', 'worse', 'negative', 'poor']

I filtered my vectorized articles by these keywords and began looking into the difference in word usage and importance for articles that were posted on a company prior to outperforming their earnings vs. ones posted prior to a company underperforming their earnings. Although the difference was not as stark as I would have liked, it was interesting to see the small differences like the outperforming articles including more positive words than the underperforming articles Below are the results:

Figure 4:

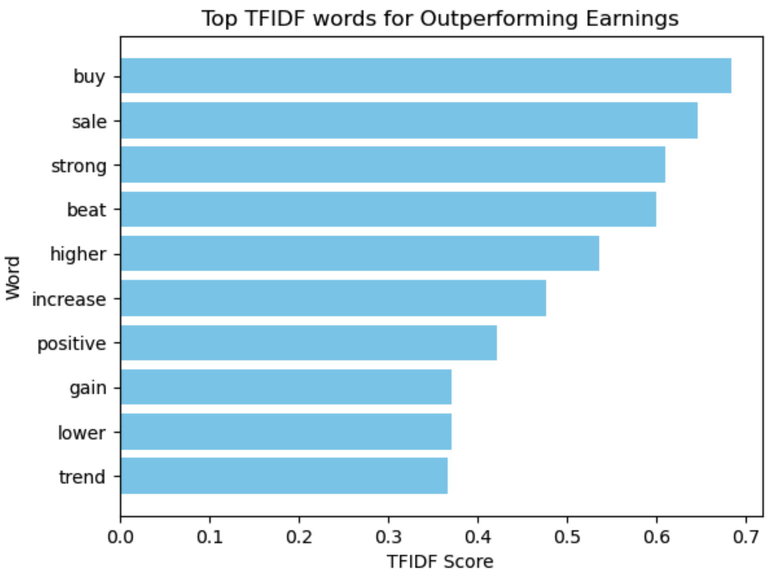
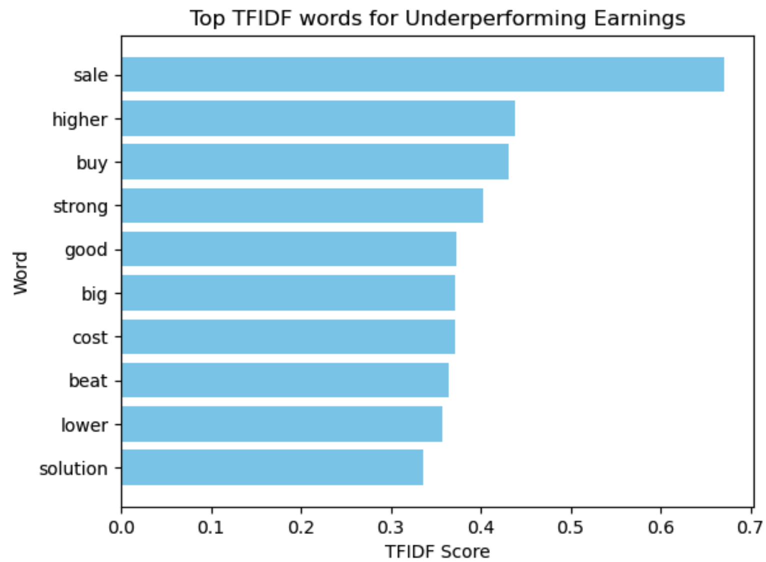


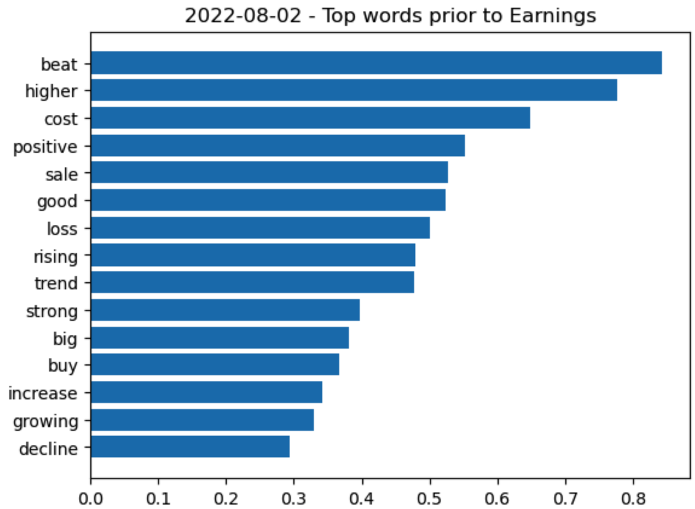
Figure 5:



Along with creating matrices for the words in the articles, I also ran the articles through the Vader Sentiment algorithm to derive a sentiment score for each article. Something I found interesting was that although a company may have reported earnings less than their estimate, the majority of articles posted leading up to their reporting still showed positive sentiment. This tracks with what the TFIDF vectorizers are showing above as although there are small differences, the overall word usage appears to be mostly similar. Additionally, over 80% of the earnings analyzed were outperforming the estimates indicating overall market strength in the tech industry which also could have led to the variance between the two being minimal. In fact, of the approximate 9,500 articles pulled, only 323 returned negative sentiment. I went through and reviewed a few of these to confirm they did speak about the respective company in a negative light. Understanding the market in general showed overall strength, I also looked into individual companies and their respective earnings reports/ associated articles. Figure 6 below shows the average difference in earnings vs. estimates for each company tested. Next to this figure (figures 7 and 8), I pulled two bar charts that show the highest scoring words during a period in which UBER outperformed and underperformed their earnings estimates to see if I could notice any differences. These bar charts still came back inconclusive.

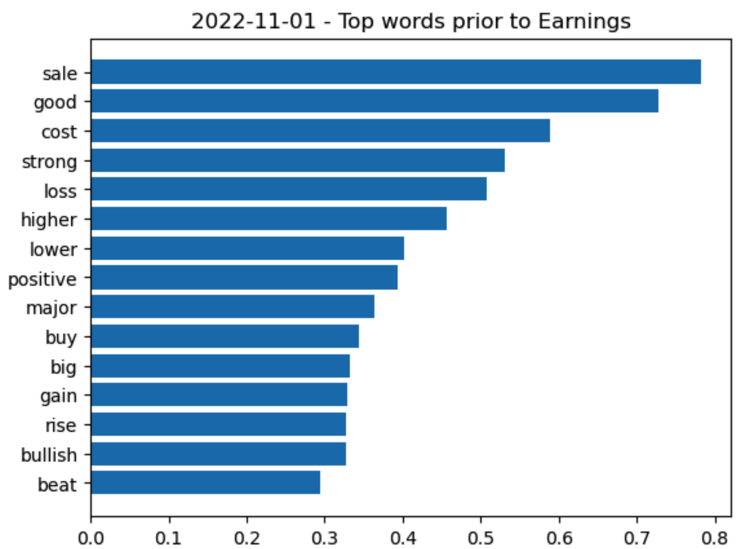
My next step was to aggregate the keywords with their respective scores and group them by ticker and earnings report time frame. I then merged this DataFrame with the earnings report DataFrame to give me a comprehensive dataset that I could use for further evaluation with binary classification models.

Figure 7:



`Difference in Estimated and Actual Earnings: -0.54`

Figure 8:



`Difference in Estimated and Actual Earnings: 0.49`

Figure 6:

ticker	surprise
AAPL	0.048000
ABNB	0.181000
ADBE	0.100000
ADP	0.058000
AMD	0.034000
ANET	0.187000
AVGO	0.025900
BABA	1.155000
CRWD	0.085000
CSCO	0.036000
DELL	0.348000
GOOG	0.031000
IBM	0.091111
INTU	0.351000
META	0.150000
MSFT	0.104000
MU	0.033000
NFLX	0.266000
NVDA	0.030510
ORCL	0.041000
PLTR	-0.003000
PYPL	0.087000
QCOM	0.117000
SHOP	0.040000
SPOT	0.044000
TSLA	0.020000
TSM	0.091000
UBER	-0.064000

Modeling

Seeing as I was unable to draw any concrete conclusions as it relates to earnings predictability from the word matrices, I created and tested some classification models to see what insights I could gain. I created two “final” DataFrames that I used to build out these models. Both included 40 columns, 37 of which were word features derived from the matrices previously created. The remaining three columns were my target variable, a month column which was derived from the date the earnings were reported, and the company ticker which was used to create dummy variables when ran through the models. The difference between these two DataFrames were the values for the keywords. One of them included the TFIDF values, the other used the Count values. Also, it is important to note that the total number of rows was quite light at 277. This is due to the lack of ability to pull articles from years prior to 2022 and the lack of pulling information for more companies. The models I chose to run included Random Forest Classifier and Support Vector Machine. Seeing as 86.3% of all earnings reported outperformed their estimates, the goal was to create a model that could predict above this number. I tried running the models both with and without the sentiment scores attached. This is because as mentioned, only 323 articles returned a negative sentiment with the majority of articles scoring above .90 which I felt had the potential to skew my models. Figure is a list of the models and different hyperparameters tuned that I tested. I could include a matrix for each that shows the precision, recall, and f-1 score, however almost all the results came back with the same scores. Accuracy hovering around 85% but little to no predictive power when it came to predicting when a company was going to underperform.

Figure 9:

Models without Sentiment			
	Random Forest - TFIDF	No hyperparameter tuning	Accuracy: 82.1%
	Random Forest - TFIDF	Criterion: 'entropy', min_samples_leaf: 2, n_estimators: 50, min_samples_split: 5	Accuracy: 83.3%
	Random Forest - Count	No hyperparameter tuning	Accuracy: 88.1%
	Random Forest - Count	Criterion: 'entropy', n_estimators: 50, min_samples_split: 10, max_depth: 30	Accuracy: 88.1%
	SVM - TFIDF	C = .1, kernel = 'rbf'	Accuracy: 86.9%
	SVM - Count	C = .1, kernel = 'rbf'	Accuracy: 86.9%
Models with Sentiment			
	Random Forest - TFIDF	n_estimators: 50, min_sampeles_split: 2, min_samples_leaf: 1, max_depth: 50	Accuracy: 85.7%
	Random Forest - Count	n_estimators: 50, min_sampeles_split: 2, min_samples_leaf: 1, max_depth: 50	Accuracy: 87%
	SVM - TFIDF	C = .1, kernel = 'linear'	Accuracy: 80.9%
	SVM - Count	C = .1, kernel = 'rbf'	Accuracy: 86.9%

If I had to choose a model that I felt performed best, it would be the Random Forest Model with hyperparameter tuning. This is because although it did not show the highest accuracy, it was the only model that even attempted to predict a negative difference in estimated and actual earnings. Below shows the classification report for this model.

Figure 10:

	precision	recall	f1-score	support
0	0.67	0.15	0.25	13
1	0.86	0.99	0.92	71
accuracy			0.86	84
macro avg	0.77	0.57	0.59	84
weighted avg	0.83	0.86	0.82	84

Conclusion:

This project has been incredibly insightful to me not only as it relates to beginning to understand the relationship between news articles and related earnings reports, but also the complexity of Natural Language Processing techniques and the power they hold. Although I would not say any of my models are particularly useful at this point, I do believe there is further research that can be done here to improve predictive power. I believe the data wrangling and cleaning plays the most important roll here and hindered the performance of my models the most. The tests I ran included 28 companies from the same industry to see if I could create a model with predictive power, however there are many other ways to subset the data that could provide different results. I chose my method because I was restricted with the API I was using to only the most recent 2 years worth of data. Without this restriction, it would be interesting to pick only 1 company and obtain articles from approximately 10 or 20 years prior and test them against their earnings reports. Another issue that I ran into was that many articles mentioned more than just 1 company. In order to get around this, I would need to do a much deeper dive into the articles themselves and try segregating the information that referenced each company to try and derive more accurate sentiment scores. Lastly, I would like to try and mess around with the date ranges when pulling articles that relate to a certain earnings report. The dates I used were directly prior to a company announcing their earnings report, however the actual earnings data came from almost a month prior (when the quarter fiscally closed for the company). Pulling articles from the quarter the earnings is posted on could potentially produce a greater signal for predicting the company's earnings. I plan to continue to modify this model and hope to find better results in the future.