Using Text Mining and Machine Learning to Predict the Impact of Quarterly Financial Results on Next Day Stock Performance

Itamar Snir

The Leonard N. Stern School of Business
Glucksman Institute for Research in Securities Markets
Faculty Advisor: Joel Hasbrouck
April 1, 2017

I. INTRODUCTION

In this project I study whether it is possible to automatically classify the impact of quarterly report on the stock's next day performance, based purely on the textual content of the report. The stock's performance is defined positive in cases where the company's stock excess return over the market return (measured by the S&P 500 index) was positive, and negative otherwise.

I start by creating a massive dataset of over 50 thousand quarterly reports from more than 2000 companies in the NASDAQ index. Then, I explore the best practices known so far to extract information out of text in general, and specifically from financial textual information. I then implement both dictionary and machine learning based approaches to test my hypothesis, using the Python programming language. Finally, I implement a hybrid process in which the dictionary based approach was used as a preprocess to the machine learning classification. The best result is achieved using a combination of the dictionary and machine learning approaches, improving accuracy by about of 5.5% over the random guess baseline of 50%.

II. METHODOLOGY

Behavioral finance researchers have recently intensified their efforts to understand how sentiment impacts individual decision-makers, institutions and markets. Broadly speaking, two types of sentiment have been studied: investor sentiment – beliefs about future cash flows and investment risks that are not justified by the facts at hand, and textual sentiment – the degree of positivity or negativity in texts (Kearney and Liu, 2013). This paper will only focus on the second type.

The most popular methods to extract information from financial related documents can be broadly separated into the dictionary-based approach and machine learning.

I.1 Dictionary Based Approach

In the dictionary based approach a pre-defined word list (dictionary) is used to parse the text. A word is only used in the document representation if it matches one of the terms in the dictionary, and otherwise the term is ignored. A representation of a document will therefore be a vector containing the terms that match the dictionary and their frequency in the document. The most widely used dictionary in the earlier studies was the General Inquirer (GI) built-in dictionary, developed and used by Philip Stone, a specialist in social psychology (Stone et al. (1966)). However, this dictionary was developed based on general English linguistic and was not tailored to the common financial related documents. For example, terms such as “tax” or “debt” would most likely be considered negative in general English documents, but are very common in financial reports and would not necessarily indicate a negative sentiment. Loughran and McDonald (2011) find that almost three-fourths of the negative word counts in the Harvard list are not necessarily negative in a financial context. They therefore developed a new financial-focused dictionary by inflecting each word from the Harvard dictionary to forms that retain the original meaning of the root word, and added finance-specific word lists developed by themselves (L&M lists) to assess sentiment in 10-Ks. These L&M lists have become predominant in more recent studies (Kearney and Liu).

In order to use dictionary based approach to classify sentiment a weighting rule has to be implemented. Two of the simplest methods are a binary approach - each term is assigned a weight of 1 if included in the document and 0 otherwise, and the term-frequency approach in which each term’s weight is equal to its frequency in the document. A more advanced approach is to normalize the term frequency by the inverse document frequency. This method will be described in detail in the next section. Finally, another approach is to use the polarity metric:

$$Polarity = \frac{n_{pos} - n_{neg}}{n_{pos+} n_{neg}}$$

Where n_{pos}, n_{neg} are the number of positive and negative words, respectively (Das, 2014). In this approach each document is assigned a polarity score, so that the higher the score the more likely the document signal is positive. It is important to note that unlike the simpler methods, the polarity metric isn't sensitive to the document's length.

II.2 Machine Learning

Machine learning, in the context of text analytics, relies on statistical techniques to infer the content of documents and to classify them based on statistical inference (Li, 2010). The process to classify the document begins by splitting the corpus into a training set from which the machine can learn the 'hidden patterns' and a test set which is used to evaluate the quality of the model. While there are many possible ways to measure model quality, in this paper I use three of the most common metrics: accuracy, precision and negative predictive value (NPV).

$$Accuracy = \frac{\#\ of\ correctly\ classified\ documents}{Total\ number\ of\ classified\ documents}$$

$$Precision = \frac{TP}{FP + TP}$$

$$NPV = \frac{TN}{TN + FN}$$

Where TP/TN are the number of correctly classified positive/negative document, and FP/FN are the incorrectly classified positive/negative document. These metrics are used due to their simple business interpretation - high accuracy rate would mean an investor can always trust the model, while excelling in only one of the other metrics - either the precision or the NPV, means that an investor would trust the model only in instances where it classifies a document as positive ('buy' signal) or negative ('sell' signal), respectively. While there are myriad of different classification algorithms in the machine learning literature, for the purpose of this project I limit the analysis to three of the most well-known models: Naive Bayes, Support Vector Machines (SVM), and Random Forest. To use them, I import an open-source package already developed in the Python language (Scikit-learn package).

A corpus of textual documents such as quarterly earnings can only be used for machine learning after a significant preprocess of the information, yielding a dataset of instances (documents), features and target variable (the 'signal'), as described in the next section.

II.3 The "bag of words" approach

In order to extract information from the text I base my analysis on the 'bag of words' model in which a document, or in this case a quarterly report, is represented using a bag of its words, while disregarding grammar and word order, but keeping multiplicity. This approach has been proven successful for tasks such as document classification and sentiment analysis. This will be a preprocessing step necessary in order to create a training set for the machine learning algorithms to work on. Each word or set of words (also known as n-gram, with n being the number of words) will be a feature helping the predictive model to learn the characteristics of the text. The bag of words model follows the next stages:

1) **Tokenization** - in order to separate each word in the text I create a 'tokenizer' function. This function essentially tells Python how to split words. I am splitting terms based on spaces, where terms separated by a dash or dot are counted as one token. After some trial and error, I decided to ignore numbers. That is a critical decision since financial reports include many numbers, and it is obvious that the numbers have an important impact. However, my goal is to understand text patterns. The numbers create noise in the data set, making it very hard to distinguish that noise from a meaningful signal.

2) **Stop Words** – It is useful to exclude common English terms from the analysis. Such terms are broadly referred to as 'stop words' in the text analytics lingo and include words such as 'the', 'a', 'and' and many others. This step is very important to reduce the dimensionality of the dataset and to reduce noise.

3) **Punctuation** – It is common to ignore punctuation such as quotes, commas, dots etc.

4) **Stemming** - this is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form. Examples include removal of 'ing' or 'es' from the end of terms. Stemming helps the algorithm understand that terms such as 'move' and 'moving' are similar enough to be counted the same and therefore should be viewed as the same token.

5) **Term Frequency** - After defining the framework to separate text into tokens, a 'bag' for each document can be created. This is achieved by executing a word count for each token. At the end of this step each document is represented as the 'bag' of its words, and each set of documents (commonly referred to as 'corpus') is also represented using the vector of each document.

<u>Example</u>

Consider the following three 'documents':

1) The cat is on the tree.
2) The big dog is running to the tree.
3) Trees need water in order to grow.

After executing the five steps as described above, this corpus of three documents will be represented as follows:

Doc #	Cat	tree	big	dog	run	need	water	order	grow
1	1	1	0	0	0	0	0	0	0
2	0	1	1	1	1	0	0	0	0
3	0	1	0	0	0	1	1	1	1

Note that the terms ‘the’, ‘is’, ‘on’, ’to’ and ‘in’ were all ignored since they are considered ‘stop words’. Also note that terms such as ‘tree’ and ‘trees’ were united to one feature and the term ‘running’ was captured as ‘run’, both due to the stemming step.

II.4 Inverse Document Frequency

Term frequency is a simple method to obtain a weight or score for each term in each document. However, it is also important to consider how many documents a term appears in. One intuitive idea is that reports which contain words that are uncommon across all documents are more likely to be similar to other reports that contain those words, whereas reports that have common joint words are less likely to be similar. This introduces the idea of “inverse document frequency” (IDF) as a weighting coefficient. Hence, the IDF for word j would be:

$$w_j{}^{idf} = \ln\left(\frac{N}{df_j}\right)$$

where N is the total number of documents, and df_j is the number of documents containing word j. This scheme was proposed by Manning and Schütze (1999). Loughran and McDonald (2011) use this weighting approach to modify the word (term) frequency counts in the documents they analyze. The weight on word j in document i is specified as:

$$w_{ij} = \max\left[0, 1 + \ln(f_{ij})\, w_j{}^{idf}\right]$$

where f_{ij} is the frequency count of word j in document i.

II.5 Feature Selection

One key decision in the machine learning process is how to prioritize features, and most algorithms are differentiated based on the way they assign weights to features. A common theme in machine learning is the 'curse of dimensionality' - the more features a dataset contains the more complex the learning becomes, and the amount of training data the model requires increases exponentially. Text classification is considered a high-dimensional problem since almost each term in the text is considered a feature. It is furthermore assumed that most terms are not of high importance. The algorithm is therefore looking for those features that are good discriminants - meaning the combination of those features can distinguish between positive and negative documents with good accuracy. A common step in the preprocessing stage is therefore to "assist" the algorithm by implementing a feature selection step resulting in lower dimension dataset. In this project I try few different feature selection techniques:

- Most common features - only include the few hundred most common terms (After excluding 'stop words'), but ignore terms that appear in more than 90% of the documents as they are too common.
- Best TFIDF score - only include those few hundred features with the best term frequency-inverse document frequency score.
- Dictionary based features - only include terms from the GI/Harvard dictionary and/or the L&M dictionaries. Generally speaking, this method showed better results than the former two feature selection methods.
- Fisher Discriminant (1936) - this is the ratio of the variation of a given word across groups to the variation within group. It should be noted that unlike other methods mentioned, this method is a 'supervised' method, meaning it utilizes the target variable to decide which

features are better. More formally, Fisher's discriminant score for word (w) is defined as follows:

$$F(w) = \frac{\frac{1}{K}\sum_{j=1}^{K}\left(\overline{w}_j - \overline{w}_0\right)^2}{\frac{1}{K}\sum_{j=1}^{K}\sigma_j{}^2}$$

Where K is the number of categories (two in our case) and $\overline{w}_j$is the mean occurrence of the word w in each text in category j, $\overline{w}_0$ is the mean occurrence across all categories, and $\sigma_j{}^2$ is the variance of the word occurrence in category j. Using this method, we can rank the features based on their Fisher score and use only the best ones. This method can also be used as a simple classifier function where each document is assigned the sum of its Fisher scores, and a cutoff point is decided to determine the signal of the document.

III. Data Gathering

In order to be able to leverage the power of text mining and machine learning, it is vital to create a dataset that includes many samples of quarterly reports as well as the performance of the stock before and after the release of the information to the public. For that reason, I gather dozens of thousands of reports automatically. This is achieved by creating a Python code which scrapes the EDGAR database of the SEC and downloads the 10-Qs of many companies listed in the NASDAQ. I was able to obtain 53,157 reports from 2,676 companies. This sample includes all types of companies:

- Companies that no longer exist as a public company, whether because they went bankrupt or because they were acquired.
- Mature companies that have been existing for more than 20 years.
- Young companies such as Facebook or Twitter.

After downloading the 10-Qs I use APIs from Google and Yahoo in order to know what happened to the stock of those companies the next trading day after the release of the 10-Q. This step reduces the dataset size to about 45,000 reports, since for some of the reports the APIs can't find the stock return. In addition, using the same APIs I am able to pull the relevant S&P500 return for each 10-Q, and then create the target variable for the classification - the excess return, which is equal to the difference between the stock return and the market return.

Looking on the entire corpus, the excess return distribution is symmetric around 0, as one might expect, with a very minor skew to negative values:

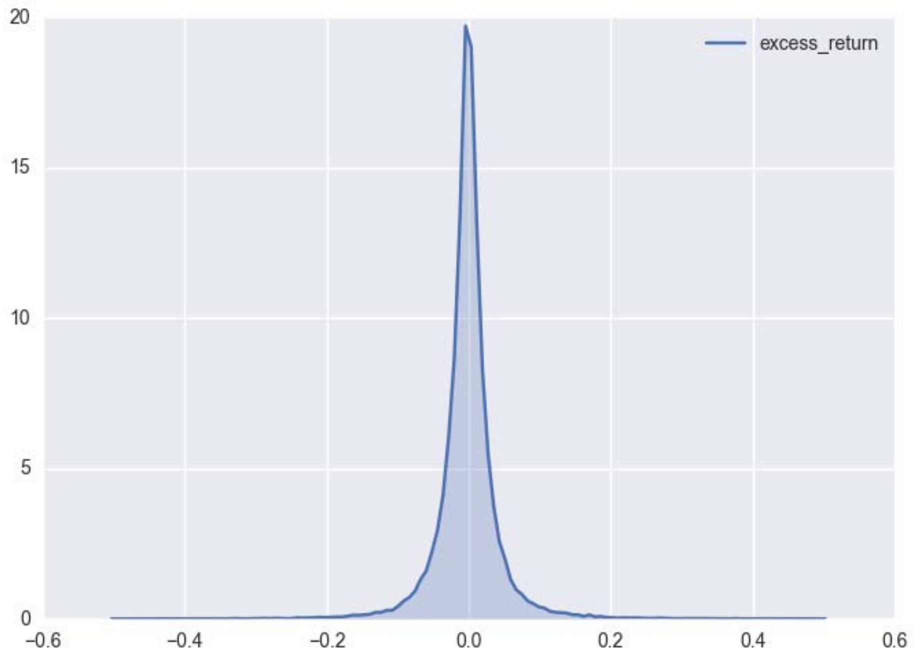


Figure 1 - Excess Return distribution
*Extreme values of more than 50% were excluded

It is interesting to note that the tails of the distribution are different depending on the day of the week in which the report was released:

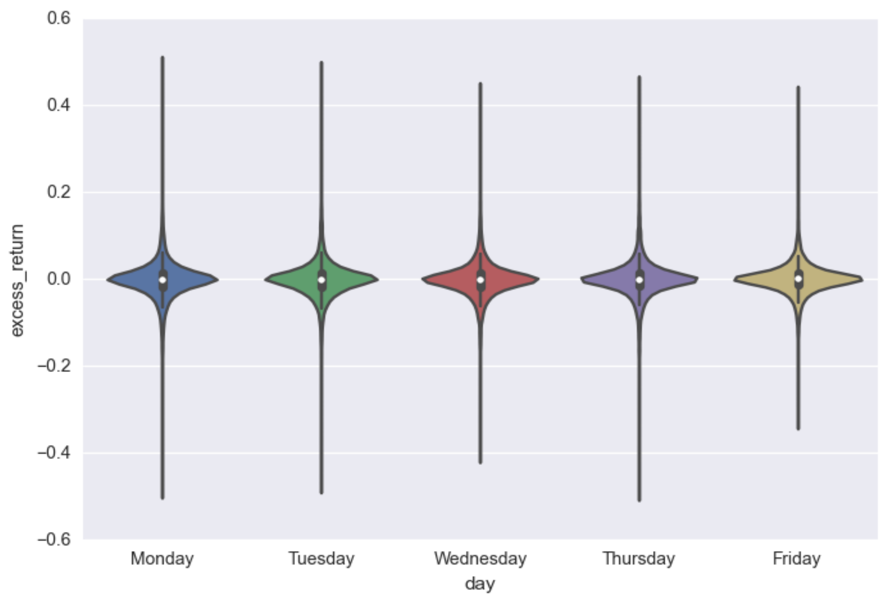
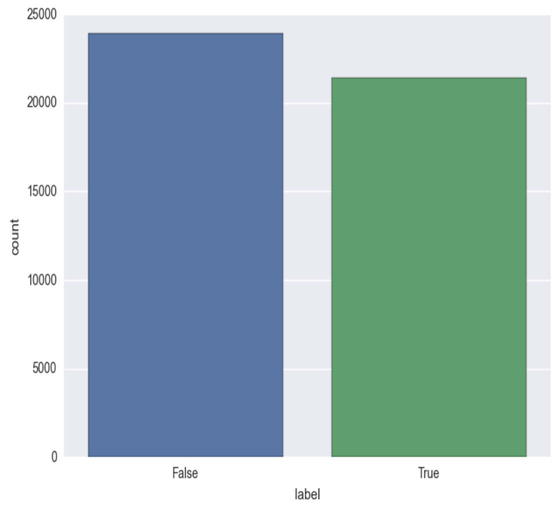


Figure 2 - Excess Return Distribution by Day

It seems that it is less likely for companies to release negative news on Fridays, as the left tail on that day is much shorter than the right tail. I next consider an indicator variable set to one (true) if the excess return was positive and zero (false) otherwise. This results in a somewhat skewed target variable, as there are more negative returns rather than positive (approximately 52% of the corpus). That distribution is consistent no matter what day of the week the earnings report is released:



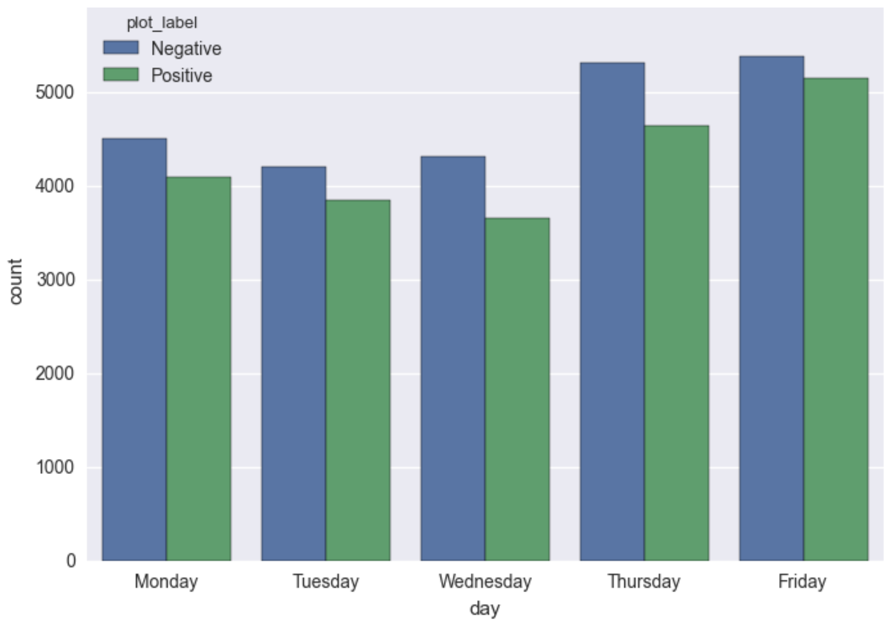


Figure 3 - Excess Return Label Distribution
Top - Total Distribution, Bottom - Grouped by Day of the Week

For the purpose of predicting whether the excess return will be positive or negative, it is a good practice to start the analysis by focusing only on the edges. The cases where the excess return is significantly positive or negative are more interesting from an investor or trader point of view. Focusing on the significant excess returns weakens the possibility that they are due to confounding events. In other words, cases in which the excess return is not significantly positive or negative are less interesting from a practical standpoint, and we should care less whether our model can classify them correctly. Those cases in which the excess return is more than 2%-3% (or less than negative 2%-3%) are more interesting as the possible outcome of buy/sell is more significant. Therefore, for the vast majority of the research I focus on the top and bottom 10% of the excess return. These are cases in which the excess return is more than approximately 3.5%, in absolute value:

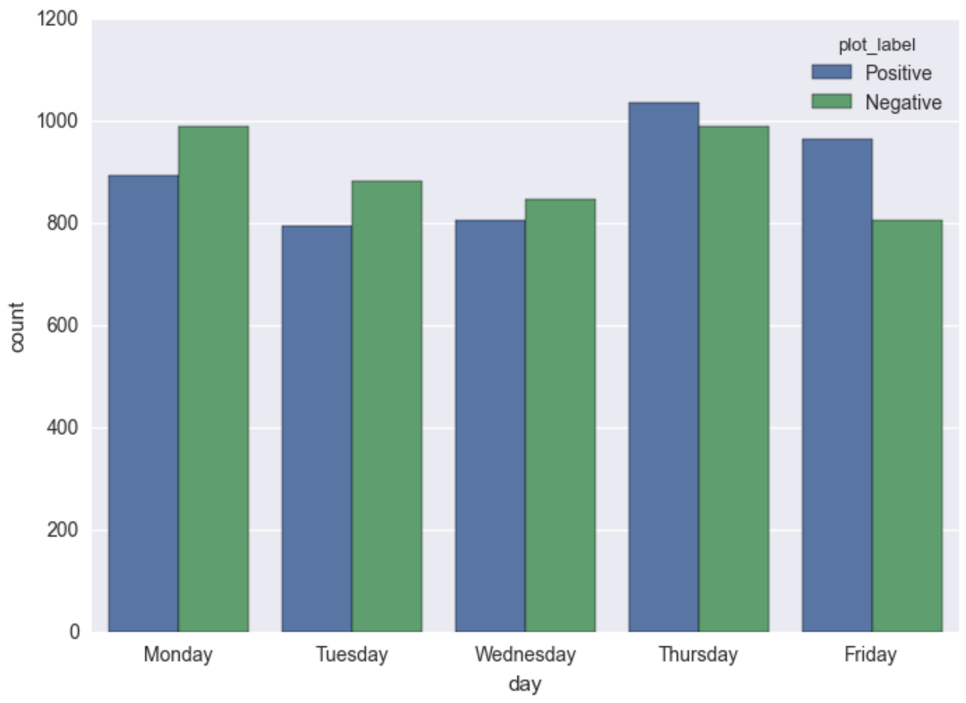


Figure 4 - Excess Return Label for Top/Bottom 10 percentile, by Day of Week

In this subset of reports it seems that it is somewhat more likely to announce positive earnings towards the end of week, while reports announced in the first half of the week are more likely to follow with negative excess return. This insight is important since the day of the week might be a good addition to the textual information and can be added as a feature to the machine learning algorithms.

IV. Results

In order to test the hypothesis, I implement all the techniques described in the previous section. I start with the simpler methods and then proceed to the more complex, in the hope of better results.

IV. 1 Dictionary based modeling:

At this stage I try using the L&M and the Harvard/GI dictionaries, separately and in combination. At first I try to see whether there's a positive correlation between the number of negative and positive terms to the class of the document. A scatter plot, where the x-axis represents the number of negative terms, the y-axis represents the number of positive terms, and the color visualizes the class of the document is an easy way to visualize and test this approach:

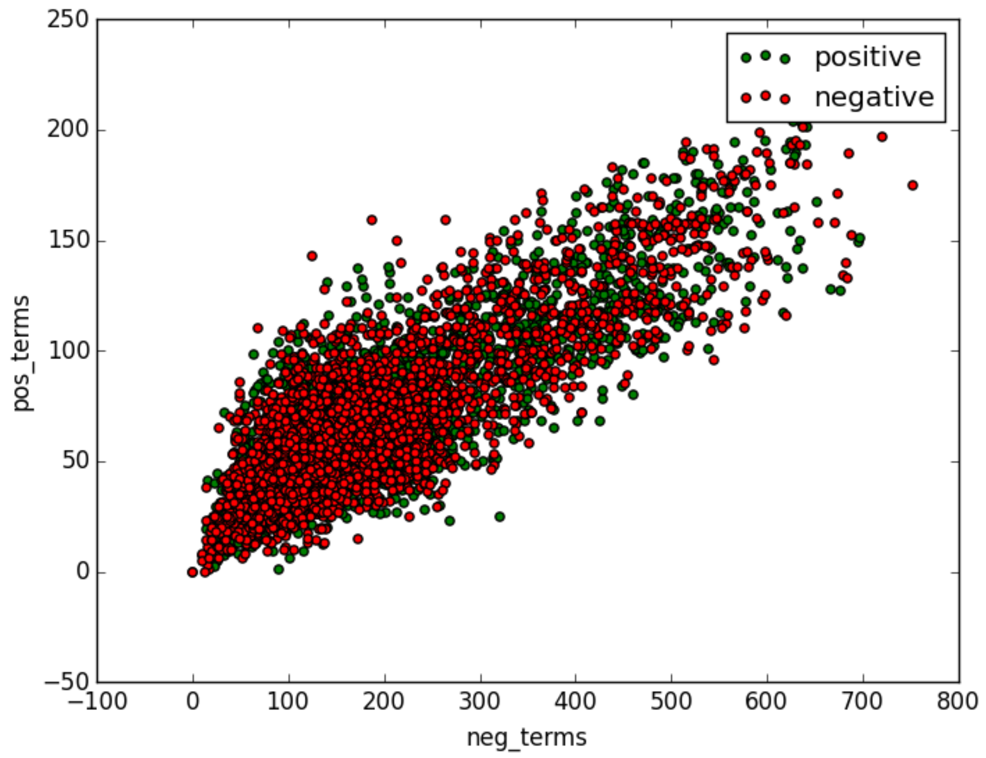


Figure 5 - Scatter Plot of the Positive and Negative Terms

Ideally, we would like to see a cluster of negative documents in the lower-right end of the plot (many negative terms, few positive terms), and a cluster of positive documents in the upper-left (many positive, few negative). However, that is not the case, there’s no apparent difference between the negative and positive documents. We can also see that there’s a strong correlation between the number of positive terms and the number of negative terms. That is not surprising since documents that are longer simply have higher probability to include more negative and positive terms. Therefore, my next step is to control for document length. I do that by looking at the percentage of positive and negative terms out of the document length, which is estimated based on the number of terms in the document:

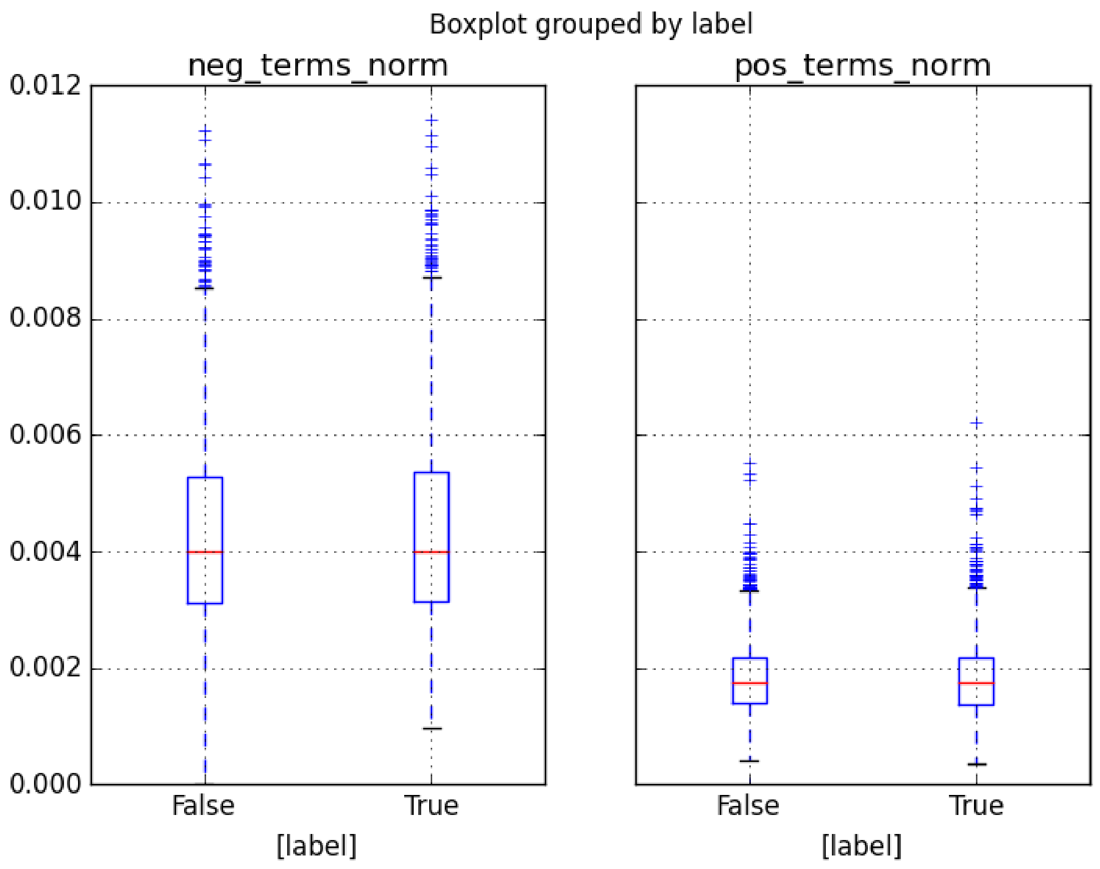


Figure 6 - Boxplot of Positive and Negative Terms, Normalized to Document Length

Despite the normalization, we cannot spot any relation between the number of negative and positive terms to the sentiment of the document. Similar results for both the controlled and non-controlled analysis are achieved for all types of dictionaries.

As a final step for the dictionary based analysis I try using the polarity metric. Each document is given a polarity score, as described in the methodology section, and then I look on the metric versus both the binary class (positive/negative) as well as against the underlying excess return, which is a continues variable. However, as the two figures below illustrate, the polarity cannot indicate a clear signal to differentiate between positive and negative reports.

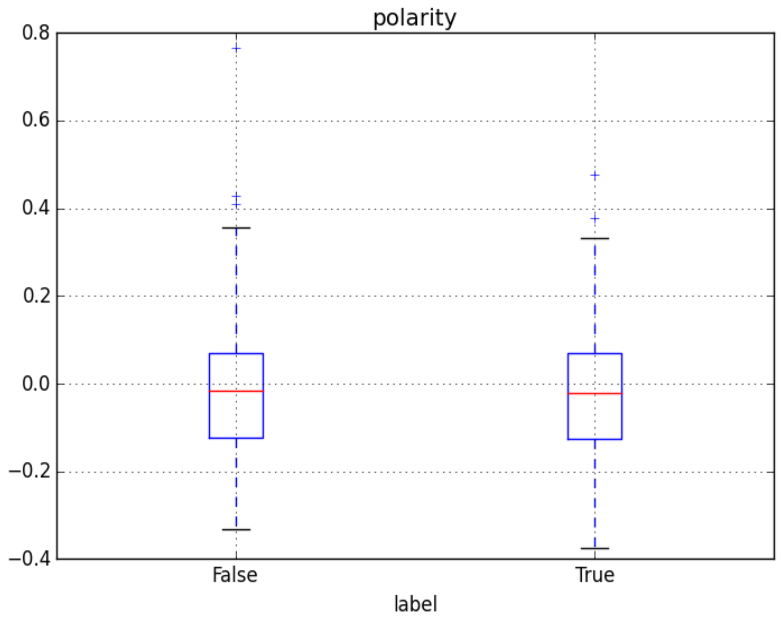


Figure 7 - Polarity Boxplot

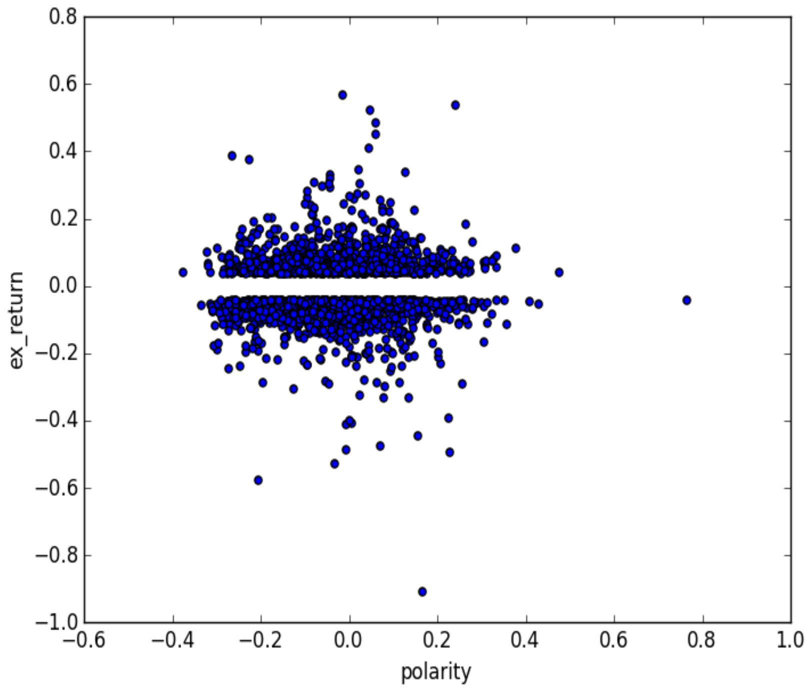


Figure 8 - Polarity vs. Excess Return

IV.2 Machine Learning Based Modeling

Since the dictionary based techniques were not useful in differentiating between positive and negative reports, I proceed to try machine learning models on the 10th percentile dataset. I implement all the preprocessing stages as discussed in the methodology section, including removing punctuation, stop words, numbers etc. I then create the term-document matrix which represents each document as a bag of its words. From this stage I try using both regular term-frequency counts and TFIDF preprocessing. For each of these two data representation methods I try three well-known classification models: Naive-Bayes classifier, Random Forest, and Support Vector Machines. For each of these models I test few different configuration hyper-parameters which control for overfitting the train data. To evaluate the performance of the models I use a cross validation approach. Cross validation is a common technique in machine learning to evaluate results accurately without leaving any part of the available data unused. In this approach we train a model on most of the train set, but we leave a small portion (fold) to evaluate the results. We repeat this process several times, and in each iteration we use a different fold to test the results while the other folds are used to train the model. To consolidate the different results from all the iterations we average the accuracy. In this study I use a 5-fold cross validation when evaluating models' performance. In each iteration I train a model on 80% of the data, and evaluate the performance on the remaining 20%.

Data Representation	**Model**	**Train Accuracy**	**5-Fold CV Accuracy**	**5-Fold CV Precision**	**5-Fold CV NPV**
Term Frequency	Naive Bayes, Alpha=1	53.42%	52.7%	54%	52%
Term Frequency	Naive Bayes, Alpha=0.1	53.41%	52.16%	51%	55%
Term Frequency	Random Forest, 50 Trees	51.59%	53.99%	53%	55%
Term Frequency	Random Forest, 500 Trees	52.65%	51.88	52%	52%
Term Frequency	SVM, Gaussian kernel, C=1	99.58%	49.44%	49%	52%

Term Frequency	SVM, Linear kernel, C=1	75.54%	49.49%	48%	51%
TFIDF	Naive Bayes, Alpha=1	55.73%	51.55%	52%	51%
TFIDF	Naive Bayes, Alpha=0.1	55.17%	51.38%	51%	52%
TFIDF	Random Forest, 50 Trees	50.91%	51.33%	51%	51%
IFIDF	Random Forest, 500 Trees	52.8%	53.49%	53%	54%
TFIDF	SVM, Gaussian kernel, C=1	50.56%	47.28%	47%*	0%*
TFIDF	SVM, linear kernel, C=1	58.6%	53.55%	53%	54%

Table 1 – Results for Various Classification Methods

Based on these results, it seems that using best practice classification algorithms without any feature selection method cannot identify any patterns that would differentiate between positive and negative reports in a useful manner. With approximately 50% accuracy as a base rate, all the methods above reach a disappointing result for each of our metrics. It is also interesting to note that most algorithms can't even get good results on the train set. The only two models that show good results with the training data are the linear SVM and the Gaussian SVM. This is not surprising since SVM is a much more complex model than Naïve Bayes and Random Forest. Since the train set includes only about 7000 instances, it is safe to assume the SVM models create a support vector for almost each one of the instances in the train set, leading to a very good result in the learning phase, but an awful result in the cross validation phase, even worse than a random guess. This means that those models couldn't generalize beyond the train set, which is the famous overfitting problem.

The fact that most of these models weren't able to get good results with the train set implies that we incur the problem of high dimensionality - the ratio between the number of features and number of instances is too high. Therefore, my next step is to try using the L&M dictionary and the Fisher score as a feature selection process before the classification algorithms.

IV.3 Reducing Dimensionality with Feature Selection

The Fisher score is a supervised way to implement feature selection, since the selection is dependent on the target variable. I start by implementing the metric for each feature (term), and then execute the classification models again with few iterations, where in each iteration I increase the number of features included in the model (500, 2000, 10000). The results prove a very minor improvement over the 50% baseline.

Number of Features	**Model**	**Train Accuracy**	**5-fold CV Accuracy**	**5-fold CV Precision**	**5-fold CV NPV**
500	Naive Bayes, Alpha=0.1	57.31%	52.35%	54%	52%
2000	Naive Bayes, Alpha=0.1	62.78%	53.98%	54%	54%
10000	Naive Bayes, Alpha=0.1	68.91%	53.31%	53%	54%

Table 2 – Results for Classification with Fischer Based Feature Selection

Another way to focus on the most important features is to use the predefined dictionaries that were discussed in the dictionary based methods section. While using simple linear weights as a discriminating function didn't yield good results, it is possible that the machine learning algorithms will be able to yield more value by using only the terms from the dictionaries. Therefore, I try using the L&M dictionary terms as a feature selection step, when the machine learning classifiers would only use them and ignore all other terms. As the table below shows, this methodology yields the best results for the cross validation accuracy metric – 55.42%, and 57% precision, when using all the terms in the dictionary (about 7200 terms). It is also important to note that we no longer overfit the training data, as the cross validation results are mostly consistent with the train results.

Number of Features	**Model**	**Train Accuracy**	**5-fold CV Accuracy**	**5-fold CV Precision**	**5-fold CV NPV**
100	Naive Bayes, Alpha=0.1	60.63%	53.04%	53%	53%
1000	Naive Bayes, Alpha=0.1	60.41%	51.65%	50%	53%
Entire Dictionary	Naive Bayes, Alpha=0.1	60.35%	**55.42%**	57%	54%

Table 3 – Results for Classification with Dictionary Based Feature Selection

As a next stage, I try to combine the Fisher score with the L&M dictionary, so that the models will only learn from the terms in the dictionary that have the highest Fischer score. As can be seen below, this method doesn’t yield better results than using the entire L&M dictionary’s terms.

Number of Features	**Model**	**Train Accuracy**	**5-fold CV Accuracy**	**5-fold CV Precision**	**5-fold CV NPV**
4000	Naive Bayes, Alpha=0.75	60.61%	53.35%	53%	54%
2000	Naive Bayes, Alpha=0.1	59.41%	53.61%	54%	53%
1000	Naive Bayes, Alpha=0.1	58.42%	53.65%	53%	55%
500	Naive Bayes, Alpha=0.1	57.69%	53.31%	53%	54$

Table 4 – Results for Classification with combined Fischer and Dictionary Feature Selection

V. Summary

Out of the methods I tried to model the problem, the best approach is using the entire L&M dictionary without any additional feature selection. The model’s accuracy with a 5-fold cross validation is 55.42%, the model's precision is 57%, and the NPV is 54%. To illustrate what terms are of highest value we can look on the list of most important 10 terms. These are the terms that the model found as the best differentiators:

Positive Terms	**Negative Terms**
Attainment	Loss
Beautiful/Beautifully	Effective
Booming	Impairment
Courteous	Claims
Creativeness	Against
Delightfully	Benefit
Distinctively	Gain
Enjoyably	Adverse
Excelling	Litigation
Exclusiveness	Capital

Table 5 – List of Most Important Positive and Negative Terms

It is important to remember that using only these 10 terms is not enough, as the models get their true power by using all the terms in the dictionary, as shown in the comparisons before. While the result is better than a random guess by about 5.5% percent, it is still not the result I had hoped for.

From a trading perspective, the model's accuracy isn't useful, and it would be more important to estimate the performance of using our best model as a trading strategy. Based on our definition and modeling of the problem, this strategy aims for a one-day trade – when the predictive model predicts 'buy' the trader will buy the stock and short the market, and will exit that position at the end of the next trading day. Similarly, when the model predicts 'sell', the trader will short the stock and buy the market and exit the next day. Therefore, we can estimate the quality of our model as a trading strategy by looking on the average excess return one would have made by following the model's prediction. More formally, our strategy's quality is measured as follows:

$$\frac{1}{N}\sum_{i=1}^{N}|Ex.Return_i| * P_i \text{ , } Where \;\; P_i = \begin{cases} 1, & Prediction\ is\ correct \\ -1, & Prediction\ is\ wrong \end{cases}$$

Note that we only care about the absolute value of the excess return. If it is negative and our model predicted it would be negative, the trader will short the stock and buy the market, thereby making positive return by following the strategy. When evaluating our model as a trading strategy it has a very minor positive return of 0.2% over the market return. In other words, if we were to follow the model's prediction for our 40,000 quarterly reports dataset, we would have earned only 0.2% better return than simply buying the market instead. This is a very small excess return, and isn't likely statistically significant. This analysis doesn't take into account those repots the model has learned from, and only looks on 'new' instances which the model has never seen before.

From a practical standpoint, using an approach that is correct only 55% percent of the time and can only yields minor return over the market will not necessarily be a good investment strategy, and therefore my conclusion is to reject the hypothesis that the impact of quarterly earnings reports on next day stock performance can be automatically mined.

VI. Future Research

As a follow-up research to this project, I suggest few additional or alternative methods to consider:

- Get more data for the learning phase - with only about 10 thousand instances to learn from (after filtering the top and bottom 10 percent), it might be useful to obtain more 10-Qs as training data. This is considered "expensive" as it might take a lot of time to obtain more data.

- Natural Language Processing (NLP) – Another common practice in text analytics is to use NLP. These techniques can tag each term in a sentence to a part of speech. For example, tagging nouns, verbs and adjectives. Since we are looking for a sentiment or signal in the reports, it'll be interesting to use only adjectives as a feature selection step, and then continue to the machine learning step.
- Deep Learning - this more computationally complex machine learning approach has shown impressive results in recent studies for problems such as image recognition, voice recognition, and sentiment analysis. Therefore, it'll be interesting to explore whether those results can be replicated with our dataset. However, deep learning also requires an extremely big dataset to learn from, and it is not necessarily possible to obtain that many 10-Qs.
- Volatility prediction – while I couldn't predict the impact of reports on stock performance, it is valuable to research whether text analytics techniques can be used to predict extreme movement in either direction. This kind of signal is valuable in options trading.

VII. Bibliography

1) Colm Kearney and Sha Liu, Textual sentiment in finance: A survey of methods and models, April 2013.
2) C.D. Manning and H. Schütze. Foundations of Statistical Natural Language Processing. MIT Press, 1999.
3) T. Loughran and W. McDonald. When is a liability not a liability. Journal of Finance, 66:35–65, 2011

4) Stone, P. J., D. C. Dunphy, M. S. Smith, and D. M. Ogilvie (1966). The General Inquirer: A Computer Approach to Content Analysis. Cambridge: MIT Press

5) Sanjiv Ranjan Das, Text and Context: Language Analytics in Finance. Foundations and Trends ® in Finance, Volume 8, Issue 3, 2014.

6) Li, F., (2010). The information content of forward-looking statements in corporate filings - A Naive Bayesian machine learning algorithm approach. Journal of Accounting Research, 48, 1049-1102.

7) R. A. Fisher. The use of multiple measurements in taxonomic problems. Annals of Eugenics, 7(2):179–188, 1936.