NLP Analysis of Company Earnings Releases

Final Report

Thomas Ulrich, Chaz Pratt, Philipp Thun-Hohenstein

Abstract: The semantic information contained in the transcripts of American companies' earnings releases is analysed and used to predict stock price changes. Significant improvements on the best current publicly available solutions are presented. We compare results from the Naïve Bayes algorithm and a framework leveraging recent NLP advances such as the doc2vec toolkit.

Introduction

In this project, our goal is to use machine learning to dissect and leverage the rich information contained in company earnings transcripts – i.e. the written transcripts of the quarterly earnings calls held by the management of American and most international publicly held companies – to predict whether the companies' stock prices will rise or fall. Although this problem has been on the radar of the machine learning community for some time, there is not yet a conclusive, publicly available solution. One of us can also tell from personal experience that it represents a functionality gap at even the most informed fundamentally-oriented hedge funds, and has likely only been solved by specialized quantitative trading strategies or service providers.

The problem is important because 1) it may harbour significant profit opportunities if markets do not efficiently price in the full semantic information contained in company earnings releases, and 2) in case inefficiencies are unearthed, it provides a blueprint to address them, leading to better price discovery and more "complete" market prices vs. the status quo today.

The input for our project is a series of earnings transcripts for large American companies over a period of 15 years (see details in the dataset section). After pre-processing the data to eliminate noise, stopwords and punctuation, and splitting it into individual sentences, we use a variety of procedures to generate features for several of the machine learning algorithms we discussed in class, most successfully the Naïve Bayes algorithm. This ranges from simple generation of word frequency matrices, as in the spam filter problem from class, all the way up to recent advances in NLP word- and phrase-vectorization. We have avoided any sort of grammar-based parsing as we perceived the benefit to be dubious in the context of the more sentiment-driven analysis we wanted to achieve.

Related Work

In 2010, a team from Stanford's CS224N natural language processing course used unsupervised content selection to generate 10-sentence summaries of the earnings report transcripts and tried to predict stock volatility (rather than prices) based on the results. They argued based on the efficient market hypothesis that predicting stock prices using publicly available data would be impossible. However, they didn't present data to back up their assertion.

An example project from MIT's Sloan School of Management machine learning course [MIT OpenCourseWare] seems to have been the first publicly available demonstration that quarterly earnings reports could be used to predict stock prices. The team used a random forest scheme based on linear regression. In keeping with the course's focus on business aspects rather than mathematics, the paper was not as technically ambitious as others we read; however, it provides a useful introduction to the topic.

Later in 2012, a team from the University of Michigan [Gierad] reported a consistent 1% increase in performance, compared to the standard Capital Asset Pricing Model. They used a set of six different kinds of text features generated from sets of words in the earnings reports. Then, a complex algorithm

combined six models, including linear and RBF SVMs, logistic regression, Bayesian networks, a decision tree, and a perceptron, using boosting and bagging meta-methods, and successfully used the results to model future stock prices. After this result, several startups including Lucena Research, InvestieComp, and PredictionValley began working on the problem and further public work seems to have stopped.

We also conducted some research into the inclusion of context in bulk NLP learning projects, since we perceived the lack of context preservation to be one of the key shortcomings of existing contributions. Ignoring a significant chunk of historical advances including [Miller], [Koo] and [Brown] (due to their difficulties in achieving scale beyond a few thousand samples, compared to our 500k sentences, see section3), we ultimately found two very interesting papers by Google researchers ([Lin, Wu], [Le, Mikolov]) that propose a context-preserving solution for generating features for discriminative learning algorithms – see more detailed description in section 3 under feature generation.

Dataset and Features

Data Availability and Collection. Since 2002, transcripts of company earnings release conference calls have been made public under Regulation Fair Disclosure, which was aimed at providing small retail investors with access to the same type of information enjoyed by large financial institutions. Transcripts are made available through several services including Factiva, Thomson Reuters, Factset, Bloomberg, as well as sites like Seekingalpha.com. We acquired a dataset of all the available earnings releases available on the Factiva database for the 20 largest American companies by market capitalization, which equates to about 50 releases per company, or roughly 1,000 documents total.[1]

Preprocessing. Our preprocessing consisted of standard cleaning (lower case, punctuation etc) followed by eliminating very short stub sentences and stopwords. We also classified the sentences from each earnings transcript into 3 separate buckets to be able to run more fine-grained analysis: i) main body of the earnings transcripts (usually pre-scripted), ii) questions asked by the large banks' financial analysts, and iii) answers by management to these questions.

Feature Generation. We applied three major methods of feature generation:

- Raw frequency matrix: This is the approach familiar from the spam classifier in problem set 2. We generate a list of all words in the dataset and count how many times each word appears in each document.
- Paragraph vectors: One of our starting assumptions for this project was that existing solutions fall short because they do not sufficiently capture *context*, even in cases where bigrams or multigrams are used. Two recent papers published by Google scientists [Lin, Wu], [Le, Mikolov] provide inspiration as to how best to integrate such context. The basic idea is to generate semantics-driven vector-representations of sentences in multiple steps. First, the words in all phrases in the dataset are evaluated in a context window of up to 3 words forward and backwards. Words which frequently appear in the same context are assigned vectors that are close in the word vector-space. Then, for each sentence a paragraph vector is derived which, given the existing word vectors and several words in a phrase, maximizes the probability of predicting subsequent words in that phrase. This allows preservation of context, since words that on their own can be ambiguous (e.g. "great") derive specific meaning through the paragraph vectors. Sentences that express similar sentiment then end up close together in paragraph-vector-space. Clustering such paragraph-vectors via a k-means algorithm allows us to capture how many sentences in a given earnings release correspond to preset clusters, which we can suppose stand for certain qualities such as "sensational", "honest" etc.[2] We used the Python gensim module's doc2vec toolkit to automate this process.

Other Notes on Dataset / Features. We did not try using labelled datasets in our analysis since most of these databases stem e.g. from movie reviews and differ quite heavily from our application space in the

[1] The dates ranged from 2002-2015. We did consider the fact that the current 20 largest companies were more likely than others to have rising stock prices; however, scoring our results by comparing them to a control strategy eliminates this source of potential bias. See details below.

[2] However, we cannot expect to identify specifically which cluster corresponds to which group of sentiments.

language used and the sentiments expressed. We also avoided the use of any grammar parsing algorithms, chiefly because we eliminated common grammar prepositions as stopwords and because we expected limited value-add from explicit grammar parsing. As an experiment, we tried using variance threshold feature reduction to select the most important features by ignoring those that are the same in more than 80% of samples. The results were generally good (see below).

Methods

The convenient pipeline interface offered by Python's Scikit-Learn module let us experiment with a range of different learning algorithms, but by far the most important and successful was the Naïve Bayes algorithm. This algorithm is built on a very mathematically simple idea, Bayes' rule, which says that if you know the probability of event X, the probability of event Y, and the probability of event X given event Y, you can calculate the probability of event Y given event X from this formula:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

Why is this important? Well, if we are trying to classify a set of text documents into two sets, Bayes' rule lets us estimate the probability that a document falls into set 1 if we have estimates of three things:

1. p(x), the likelihood of the document itself (i.e. the probability of the document existing – very unusual documents like the Lewis Carroll poem "Jabberwocky" would have low likelihood functions, while typical corporate earnings calls have a lot of boring, everyday features and have high likelihoods);
2. p(y), the probability that any document belongs in set 1;
3. and p(x|y), the likelihood of a document similar to ours occurring in set 1.

All of these probabilities can be estimated from a training set. The best estimate for p(y = 1) is obvious: just count the number of documents in set 1 and divide by the total number of documents. Understanding the likelihood p(x) is a bit more difficult. Ignoring sentence structure for a moment, let us represent a document as a set of words that occur with particular frequencies. In the feature vector at right, for example, the words "a" and "buy" occur exactly once in the document, while "aardvark," "aardwolf," and "zygmurgy" do not appear at all.[3]

$$x = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \text{a} \\ \text{aardvark} \\ \text{aardwolf} \\ \vdots \\ \text{buy} \\ \vdots \\ \text{zygmurgy} \end{matrix}$$

Let us pretend for a moment that word counts are completely independent of one another. That is, the number of times the word "a" appears is independent of the number of times the word "the" appears, and so on. Let us also further simplify the problem by converting the feature vector above into binary. That is, we don't care how many times a word appears but only whether or not it appears at all. This will simplify the math that follows. Then if $x_1 \ldots x_{50,000}$ represent the presence or absence of 50,000 different words in the document, we have

$$\begin{aligned} & p(x_1, \ldots, x_{50000}|y) \\ &= p(x_1|y)p(x_2|y, x_1)p(x_3|y, x_1, x_2) \cdots p(x_{50000}|y, x_1, \ldots, x_{49999}) \\ &= p(x_1|y)p(x_2|y)p(x_3|y) \cdots p(x_{50000}|y) \\ &= \prod_{i=1}^{n} p(x_i|y) \end{aligned}$$

where we can get the $p(x_i|y)$ from the training set. Furthermore p(x) can be found easily enough as

$$\left(\prod_{i=1}^{n} p(x_i|y = 1)\right) p(y = 1) + \left(\prod_{i=1}^{n} p(x_i|y = 0)\right) p(y = 0)$$

[3] No, "zygmurgy" is not a real word.

And that's all we need to start making predictions. How reasonable were our assumptions? Assuming zero correlation between word counts is arguably unrealistic, but practice has shown that making this assumption often produces useful results. Making the feature vector binary made the equations shorter and simpler, but it wasn't necessary. One can easily generalize our approach to a multinomial distribution. Thus, we have a reasonable algorithm. Indeed, this Naïve Bayes algorithm is often used in the real world for text classification tasks like finding spam e-mails.

Results and Discussion

To measure the effectiveness of our strategies, 6-fold cross-validation was run on a random permutation of the data. For the test set of each fold, an average return was computed from two different strategies: buying an equal (monetary) amount of each of the stocks recommended by our machine learning algorithm (the 'ML strategy') and buying an equal amount of every stock in the test set (the 'control' or 'index fund' strategy). All stocks were then held for one quarter after the release of the earnings call transcript and sold. Then the APY = $(1 + \text{quarterly yield})^4 - 1$. We repeated this process three times and averaged the results. Thus, each ML strategy was tested against the control strategy a total of 18 times.

Features	Feature selection	Classifier	ML APY	Control	Diff
Word counts	none	Naïve Bayes	11.69%	8.71%	2.98%
Word counts	Variance threshold	Naïve Bayes	12.52%	8.78%	3.73%
Word counts	none	SVM[4]	7.42%	8.67%	-1.25%
Word counts	none	Logistic regression[5]	12.20%	8.73%	3.47%
Word counts	Variance threshold	Logistic regression	12.77%	8.74%	4.03%
Word counts	none	Random forest	6.18%	8.72%	-2.54%
Word counts	none	Bagging using Naïve Bayes	10.66%	8.75%	1.90%
Word counts	none	Gradient boost	9.93%	8.71%	1.22%
Bigrams	Variance threshold	Naïve Bayes	12.79%	8.80%	4.00%
Bigrams+trigrams	Variance threshold	Naïve Bayes	12.85%	8.83%	4.03%
Bigrams	Variance threshold	Logistic regression	*program execution failed*[6]		
Bigrams	Variance threshold	Random forest	10.43%	8.69%	1.74%
Bigrams	Variance threshold	Gradient boost	9.51%	8.78%	0.73%
Cluster distances	none	Random forest	8.71%	9.41%[7]	-0.71%
Cluster distances	none	Gradient boost	8.91%	9.45%	-0.54%

As you can see, the most important (and somewhat surprising) result is that we are indeed able to beat the market: most machine learning strategies based on the earnings call transcripts do at least a little better than the control strategy and many of the strategies perform 3% or more better. These are outstanding results that fully justify our interest in making investment decisions based on analysing earnings call transcripts.

[4] Our SVM results should be taken with a grain of salt – probably due to the relatively small data set, the SVM would often recommend buying all stocks or buying no stocks at all, and the returns had an extremely high variance from run to run.

[5] Implemented as a linear classifier with stochastic gradient descent learning. We stuck with Scikit-Learn's default hyperparameters; some experimentation here and elsewhere only worsened our results.

[6] We got a division by zero error, which we think is due to insufficient data.

[7] The control strategy return is a bit higher in the NLP examples than in the others because a slightly different data set was used for compatibility.

The second result that jumps out is that while our choice of algorithm is certainly very important, choosing the right input features also has a major impact on the quality of the predictions. Using bigrams offers a noticeable improvement – apparently, considering pairs of words together is very useful for analysing text. That makes sense to us in view of the complex grammatical structures of the English language. Moving up to trigrams, on the other hand, made the program run much more slowly without significantly improving the return. Worst of all were the sentiment analysis-based cluster distances – no algorithm could produce useful predictions based on these features.

Thirdly, it seems to be a general trend that more "complicated" classifier algorithms tend to do worse here. Most glaringly obvious is the fact that despite spending many hours of work developing our NLP methods, we were unable even to match the control strategy. But there were several other examples as well; the naïve Bayes-based bagging classifier does less well than the naïve Bayes algorithm by itself, and the random forest and gradient boost algorithms don't do well at all.

Looking at the table of results above, it is clear that naïve Bayes and variations perform remarkably well on this data set. It is interesting to think about why this might be the case. Considering a key assumption inherent to the naïve Bayes algorithm, perhaps the reason is that word counts are genuinely largely uncorrelated for this data set (possibly because of the large number of different speakers). But on the other hand, the fact that logistic regression also does quite well is evidence against this theory. (Logistic regression tends to work better with correlated features.) Of course, we may just be ascribing too much meaning to these results. It's quite possible that they may be nothing more than a curiosity specific to the particular set of stocks we are working with or a consequence of insufficient data set size.

We didn't expect our classification accuracies to be especially high. Initial results suggested we might be able to reach about 70-75%, but a more serious later analysis puts our true scores somewhere in the 60-65% range for most of the algorithms. Of course, increasing classification accuracy wasn't the project goal – rather, we were looking for high returns on the stock market, which are definitely achievable with 65% accuracy in predicting which stock prices will rise. At right is a typical confusion matrix from the bigram naïve Bayes algorithm.

164	158
199	224

Conclusion

At the start of this project, we aimed to find out whether it is possible to successfully pick stocks by using machine learning techniques to analyse earnings call transcripts. We can now answer that question affirmatively. Empirically, the stock market seems not to be quite as efficient as economists tend to assume, and that in itself is an interesting result to us.

Of course, the next logical question is whether or not we can use machine learning to make millions of dollars by picking stocks. The answer to this question is far from straightforward: in view of the continued rise of quantitative analysis in the last few years, it is possible that the rest of the market has already discovered this approach and now adjusts prices appropriately. If that's true, then while this approach would have worked in the past, it may not continue to succeed in the future.

From a pure machine learning perspective, it would be interesting to test our algorithms on a larger data set. (We were limited by the fact that Stanford does not have API access to large databases of earnings call transcripts, so we were forced to download all of our data by hand.) More data would provide better opportunities to check for overfitting (successful cross validation suggests that we did not overfit for most algorithms, but more investigation is needed in some cases); let us test whether our approach is truly generalizable; and help get a better picture of why some algorithms work better than others.

Finally, we were surprised by the utter lack of positive results from our natural language processing algorithms. One possible reason is that the data may need some preprocessing (similar to the kernel methods discussed in class) before it can be used successfully. More experimentation with the parameters of the NLP algorithm might also yield interesting results.

References

S. Miller, J. Guinness, and A. Zamanian. 2004. Name Tagging with Word Clusters and Discriminative Training. In Proceedings of HLT-NAACL, pages 337–342 (quoted from [Lin, Wu])

T. Koo, X. Carreras, and M. Collins. Simple Semisupervised Dependency Parsing. Proceedings of ACL, 2008. (quoted from [Lin, Wu])

P.F. Brown, V.J. Della Pietra, P.V. de Souza, J.C. Lai, and R.L. Mercer. 1992. Class-based n-gram models of natural language. Computational Linguistics, 18(4):467–479. (quoted from [Lin, Wu])

Dekang Lin, Xiaoyun Wu. 2009. Phrase Clustering for Discriminative Learning, https://www.aclweb.org/anthology/P/P09/P09-1116.pdf. Accessed 16 December 2016

Quoc Le, Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. https://cs.stanford.edu/~quocle/paragraph_vector.pdf. Accessed 16 December 2016

Programming Libraries Used

- Scikit-Learn
- Gensim
- NLTK
- Pandas
- Yahoo! Finance
- Numpy