Sentiment Analysis on Commodity Forecasts using Random Forest with Bayesian Optimization

Anonymous Author(s)

Affiliation
Address
Email

Abstract

Commodity market can have significant impact on many aspects of economics and the price of commodities concern investors and market analyst. Price forecasts reports are produced at daily-basis on Internet. This paper presents a novel approach to classify sentiment of a corpus of financial report by using Random Forest. Optimized with Bayesian Optimization technique, Random Forest Model produces better result than conventional Naïve Bayes algorithm.

1. Introduction

As network connectivity and disk storage become more affordable day by day, the amount of information accessible to public is growing exponentially. The tremendous growth of information raises two questions: how to gather the information you want and how to extract intelligence out of vast amount of information. Search engines like Google can answer the first question very well, if not perfectly. However, to grasp the intelligence hidden in thousands of articles, news, and webpages is a much more challenging task. A very good solution is to use text classification empowered by Nature Language Processing and Machine Learning technology. As an application of such solution, we conducted a sentiment analysis using Random Forest classification and Naïve Bayes on a corpus of commodity forecasts and reports. A model that can automatically label a report as either positive or negative is built, in order to predict whether the price of certain commodity will be bearish or bullish. However this model is not limited to only commodity report; it can be generalized to classify any text with proper training dataset.

1.1 Commodity Market

As one type of financial markets, Commodity market focuses on raw or primary products for production, manufacturing, agriculture etc. Commodities like oil, gold, and grain are quite fundamental and the price change of which may have a great impact on many crucial business and even the macro-economic dynamics all over the world. Commodity market greatly concerns investors who want to benefit financially and business owners who need to evaluate the risk and cost. Price forecast and analysis about commodities are constantly being produced from a variety of sources and are all available online. Reports from large investment institution like Morgan Stanley and Goldman Sachs, financial news from media like Bloomberg, and financial blogs from renowned spectaculars or market analyst, those can all help market participants to make decisions. However, the creditability of different reports may vary and it is extremely difficult for someone to read all reports in order to see the big
picture of overall sentiment about certain commodity. Automating the sentiment analysis on such corpora enables market analysts to seize the sentiment at aggregated level more efficiently and therefore can make an educated decision.

1.2 Paper Overview

This paper presents a novel approach for a sentiment analysis on a corpus of financial report. Compared with Naïve Bayes, one of the most widely used technique in this field, Random Forest is proposed to classify a whether a report is positive or negative about the price of certain commodity. Moreover, on top of Random Forest, Bayesian Optimization is used to select an optimal set of parameters of Random Forest, including the number of trees, the depth of each tree, the minimum data points on leaf node, and the number of features to compute the potential splitting feature. From features extraction to cross-validation, this model is constructed, well-optimized, and evaluated to demonstrate that Random Forest outperforms the traditional Naïve Bayes in the context of sentiment analysis on commodity forecast corpora.

2 Methodology

2.1 Related studies

As stated by [1], sentiment analysis focuses on identifying positive and negative opinions, emotions and evaluations expressed in natural language. This area has drawn a wide-range of interest. Applications that automate the process vary from classification of product reviews [2] to investigation of the tone in 10-K reports of companies [3]. Some researchers utilize information from innovative online platforms like Twitter, whose streaming data are inherently easier to collect but more challenging to process [4]. [4] propose a sliding window Kappa Statistic for evaluation in time-changing data streams. Antweiler & Frank [5] study on more than 1.5 million messages posted on Yahoo!Finance and Raging Bull, classified all messages into three categories: buy, hold, and sell, and a statistically significant signal is found between the summarized opinion and stock return. Most researches mentioned above adopt Naïve Bayes as the primary method to analyze the polarity of text. On the other hand, the potential of Random Forest on text classification seems very promising. [6] evaluate the Decision Forest (similar to Random Forest) in terms of classifying documents from the Reuters and OHSUMED corpus, indicating Decision Forest always provides better accuracy than kNN classifier when documents are from different domain. Using keywords from as attributes, [7] explore web document classification and compare Random Forest with other traditional learning techniques like Naïve Bayes, RBF network and multi nominal regression model, where Random Forest is reported to give the best result.

Essentially, sentiment analysis can be considered as a binary text classification problem, where the two categories are positive and negative. Surprisingly, no one has experimented using Random Forest to conduct sentiment analysis, and most market sentiment analysis focused on stock market instead of commodity market, which makes this project a very valuable exploration. It is ideal to use Naïve Bayes as benchmark, given its wide use, proven robustness and satisfactory result. We hypothesize that Random Forest should yield a better performance than Naïve Bayes.

2.2 Data Collection and Clean-up

A corpus of 600 articles that forecast the price of various commodities is collected from multiple sources online (e.g. Reuters) and labeled manually either positive or negative. All of such documents are web documents containing HTML tags and hyperlinks to other articles or advertisements, which impose a significant noise on dataset. Such noise must be eliminated, not only because the embedded tags have no use to discriminate bearish and bullish reports, but because titles of some hyperlinks can totally mislead and confuse the classifier. For example, a negative forecast on gold price, “Goldman cuts gold price forecasts, recommends short position”, as shown in Table 1. The title of the hyperlink contains many significant positive words like “Bullish” and “Rebound”.
Table 1: HTML Internal Data Example

<table>
<thead>
<tr>
<th>Negative sentiment forecast report on Gold prices:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Title: “Goldman cuts gold price forecasts, recommends short position”</td>
</tr>
<tr>
<td>Internal HTML contents:</td>
</tr>
<tr>
<td>…</td>
</tr>
<tr>
<td>Hyperlink found within the page</td>
</tr>
<tr>
<td>…</td>
</tr>
<tr>
<td>&lt;a href=&quot;/article/idINDEE93904R20130410&quot; modid=&quot;</td>
</tr>
<tr>
<td>…</td>
</tr>
</tbody>
</table>

2.3 Feature extraction

After stripping off all noisy information using a python script we wrote it leads to the second level of data preprocessing: feature extraction. This process is quite crucial for text classification since features of text usually hide in the lexical semantics and sentence structure. One of the most straightforward and common features is the frequency of each word in the documents. However, a feature vector with all words will drastically increase the dimensionality of the data, which not only adversely affects the efficiency of training and testing process, but also induces high risk of over fitting [6]. Moreover, some of highly frequent words like “and”, “the” may just be frequent themselves and contain little information. Thus, it shouldn’t be used as features to represent a document and differentiate it from other types of documents. Those words, referred as “stop words”, are removed with a stop word remover.

2.3.1 Lemmatization

Another problem is that most words have morphological variants to a root word. For example, “different” vs. “differ”, organize vs. organizes vs. organizing. There are two ways to deal with this problem: stemming and lemmatization. Both of them can help to reduce morphological words to their root form. However, stemming only chops off the ends of some words based on heuristic rules. Lemmatization, on the other hand, is a more proper method, where it returns the base or dictionary form of a word by vocabulary or morphological analysis [8]. For instances, “raise”, “raised”, and “raising” will all be “rais” if stemming is used; lemmatization will attempt to return “raise”. Therefore, we choose lemmatization over stemming.

2.3.2 Term Frequency-Inverse Document Frequency (TF-IDF)

Even if we removed the stop words, the number of distinctive words or potential features is still not desirable. We also performed TF-IDF analysis to further extract the most predictive words. And the TF-IDF value can also be used to weigh the words.
According to [9], TF-IDF is the product of Term Frequency (TF) and and IDF(Inverse Document Frequency). Term Frequency is usually measured by number of occurrences for one term in one document. Following procedure describes the calculation of TF-IDF

Let $x \in X$ and let $d \in D$, where $|X| = M$ and $|D| = N$.

$X$ and $D$ are collections of all terms and document retained within the corpus.

We then define a function to measure the presence of terms found within the documents as follows:

$$f(x_i, d_j) = \begin{cases} 1, & \text{when } x_i \in d_j \\ 0, & \text{when } x_i \not\in d_j \end{cases}$$

$$freq(x_i, d_j) = \sum_{k=1}^{M} f(x_k, d_j) : x_k = x_i$$

$$DF(x_i) = \sum_{j=1}^{N} f(x_i, d_j)$$

$$IDF(x_i, D) = \log \frac{N}{DF(x_i)}$$

This represents the inverse document frequency

$$TF(x_i, d_j) = \frac{freq(x_i, d_j)}{\max(freq(x_k, d_j)) : \forall x, f(x, d) = 1}$$

$$TFIDF(x_i, d_j) = TF(x_i, d_j) \times IDF(x_i, D)$$

Using this TF-IDF weighting, only the words that occur frequently in one document but rarely in other documents will get high score; commonly frequent words will have fewer score.

### 2.3.3 Features

Counting the number of general negative/positive words in a document to analyze the sentiment is one of the oldest tricks [3]. It’s intuitive to add the predicate, $P(d) = \text{Document } d \text{ has more positive words, as one of the indicator. The positive and negative word lists are collected by interpreting the overall TF-IDF for bullish/bearish report and domain knowledge.}$

All of above processing is accomplished by a Python program implemented by us with NLTK package. In summary, stop words are removed first and then lemmatization is performed.

Then terms with high TF-IDF value are selected as the features, plus an additional feature that equals to 1 if target document contains more positive words than negative words, 0 otherwise.

In total, we have 207 terms plus a polarity calculation as our features. Terms can be further divided into a positive group of 116 (105 unigrams, 11 bi-grams) and a negative of 91 (83 unigrams, 8 bi-grams).

### 2.4 Random Forest

First introduced by Leo Breiman [10], random forest is a supervised statistical learning algorithm that has been proved having unsurpassable accuracy among classification techniques [7]. It is simply a collection of many decision trees where the output of each individual tree is aggregated and averaged. The final ensemble model is a Random Forest classifier.
A decision tree is a model with a tree-like structure where each node in the tree splits the input data points into two children nodes based on value of a particular feature. For a classification tree, each leaf node is usually a target class or a histogram. The histogram describes the probability of a data point belonging to each class, if the data point reaches this leaf. To train a decision tree model means to recursively construct a tree using training data. During each iteration, which feature and splitting pivot value to choose at is determined by Information Gain (IG) of each partition (defined by a feature and pivot value). Information Gain is calculated as following [11]:

\[
\text{Information Gain} = H(S) - \sum_{i \in \{1,2\}} \left| \frac{S_i}{S} \right| H(S_i)
\]

\[
H(S) = -\sum_{c \in C} p(c) \log(p(c))
\]

Above is the equation we implemented to construct a tree. As for the forest, the aggregate method we adopt to combine all trees is simply an arithmetic average of all output. The key to understand why Random Forest performs so well is the randomness achieved by bootstrapping and bagging. The source of randomness lies in two steps of building an individual tree: the random bootstrapped sample and the randomly set features selected, as described by the following algorithm [12]:

1. For \( b = 1 \) to \( B \):
   (a) Draw a bootstrap sample \( Z \) of size \( N \) from the training data.
   (b) Grow a random forest tree \( T_b \) to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size is reached:
      i. Select \( m \) variables at random from the \( p \) variables.
      ii. Pick the best variable/split-point among the \( m \).
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees \( \{T_b\}_B \)

where \( B \) is the number of trees. Bootstrapping helps to reduce the noise in data, and the random selection of features further reduces the risk of over-fit, since each tree only focus on a subset of features. Especially in our project, text analysis usually has large number of features and noisy data, Random Forest suits the most.

### 2.5 Model Tuning With Bayesian Optimization

A Random Forest model has many parameters like the number of trees, depth of each tree, the number of data points in leaf node and number of trials to pick the maximum information gain partition. Those parameters can significantly affect the performance of model, thus, need to be optimized. Accuracy on a particular set of training data with respect to the vector of parameters will be the objective function to optimize. Since we don’t have the expression of such function but can evaluate the accuracy given a set of parameters, a naïve brute-force approach is simply to try out all combinations. However, the dimension of input vector of parameters is four here, which means an extremely large number of sample need to be evaluated. Considering each evaluation requires building a new forest, this approach is undesirable for the computation power and time it costs. Bayesian optimization [13] has been proved a perfect solution for this. By using utility function to compute the most profitable location to sample next, Bayesian optimization can actively “learn” which set of parameters can give best accuracy.

Gaussian Process (GP) is used to model the accuracy objective function. As recommended by [14], we choose Gaussian process upper confidence bound (GP-UCB) as our utility function, which trades off the mean and value, as shown in following:

\[
GPUCB(x) = \mu(x) + \xi \sigma(x)
\]

where \( \mu(x) \) is the mean and \( \sigma(x) \) is variance for a given input vector \( x \), \( \xi \) is a weighting
parameter set to 0.1 as default.

Kernel function we choose for GP is expressed as following:

\[ k(x, x') = \sigma^2 e^{-0.5||x-x'||^2/l^2} \]

where \( \sigma \) and \( l \) are open parameters.

2.6 Cross Validation

After the Random Forest model is optimized, we randomly bootstrap 70% of our corpus as training data and use the rest 30% as test data to validate of our mode. This process repeats for much iteration. It is similar to k-fold cross validation. Same procedures all above are performed using Naïve Bayes as well.

3. Result

Using only 70% of the total dataset as training input, the initial configuration is 5 trees, maximum depth of 9, minimum number of data points on leaf node as 3, and 4 trials for feature sampling. This attempt yields an accuracy of 74.67% on predicting the test data (30% of total corpus). The result is not impressive and even worse than the common accuracy of 80% using Naïve Bayes. Afterwards, Bayesian Optimization is performed.

![Figure 1: Accuracy for Bayesian Optimization over Iterations](image)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of trees</td>
<td>1-30</td>
</tr>
<tr>
<td>Tree depth</td>
<td>1-40</td>
</tr>
<tr>
<td>Minimum Leaf Data</td>
<td>1-5</td>
</tr>
<tr>
<td>Number Feature Trials</td>
<td>1-20</td>
</tr>
</tbody>
</table>

Search range for number of trees is 1 to 30, for tree depth is 1-40, for minimum leaf data points is 1 to 5, and the number of features trials is 1 to 20. The grid is all of size 1. So, in total, we have 30*40*5*20= 120000 try-out parameter vectors. And the input space will be 120000*4. Note that the sample space here is mostly used to simulate the GP curve. As shown in Figure 1, the accuracy starts around 60%, which is fairly low, but improves drastically for the first 20 iterations. Then it oscillates around the 83% level. Only the first 100 iterations are shown here. The best observation occurs at the combination of 23 trees, 19 maximum tree, 1 minimum data per leaf, and 4 feature selection trials, where the accuracy peaks at 86%.
As the model tuned with optimal parameters configuration, cross-validation with 100 iterations is performed to assess the confidence of the model. For each iteration, we randomly bootstrap 70% of data and evaluate the accuracy of model on the rest 30%. Similar operations are applied to Naïve Bayes model as well. A 95% confidence interval for accuracy of both Random Forest and the benchmark model Naïve Bayes are computed. Result is displayed in Table 2.

<table>
<thead>
<tr>
<th>Table 2: Confidence Interval of Random Forest and Naïve Bayes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Random Forest 95% CI</strong></td>
</tr>
<tr>
<td>(84.81%, 86.67%)</td>
</tr>
</tbody>
</table>

The expected accuracy for random forest is 85.74% whilst the expected accuracy for Naïve Bayes is 82.85%. Although Naïve Bayes seems more robust with smaller variance and stable accuracy, Random Forest performs significantly better than Naïve Bayes. Even the lower bound of Random Forest is higher than the upper bound of the Naïve Bayes.

### 4. Project limitation and discussion

The whole dataset consists of 600 commodity forecast. It is a decent number to start with, but the size is rather small compared to other studies in sentiment analysis on text. The size of dataset also affects feature selection, since features are extracted based on term frequency analysis on entire corpus. Random Forest significantly outperforms Naïve Bayes, which conforms to our hypothesis. It is believed that the randomness lies in data bootstrapping and feature bagging when building a tree neutralizes the noise in data. Especially for text analysis where size of feature vector can be arbitrarily large and the human produced data can be very noisy, randomness becomes a very desirable characteristic. However, an un-tuned model yields very unfavorable results (recall that our first configuration attempt only gives 74.67% accuracy). The sensitivity towards model hyper-parameters makes Random Forest very tricky to use. Therefore, Optimization on parameters is essential to produce a satisfactory result and compete with traditional techniques like Naïve Bayes. Given the complexity to build a forest and optimize parameter with Bayesian Optimization in GP, computational efficiency should not be neglected. This is the reason why we spend so much effort on extracting most representative features only. Actually, in a certain sense, the calculation for Information Gain during tree construction is redundant with what was done for the preliminary feature extraction: they both aim to find the best feature that split different classes. However, optimizing the features beforehand is more computationally efficient.

Another issue involves computation efficiency is the search range of Bayesian Optimization. Even a small range like we picked may need hours to process, not to mention when dataset is large. On the other hand, by picking a smaller range, the risk of getting trapped by local minimum is fairly high.

### 5. Conclusion and Future Work

Although requiring tuning on parameters, Random Forest performs significantly better than Naïve Bayes in terms of sentiment analysis of text document. Bayesian Optimization is proved very powerful to optimize random forest parameters.

As discussed in previous section, the size of dataset should be expanded to give more confidence about the model. From an application perspective, we plan to extend our model with a web crawler that can feed time stamped daily live commodity reports. Such an integrated system will be able to collect commodity price forecast from all over the Internet, classify them, and help to gauge an overall sentiment of the market about certain commodity in real time. We believe the target system would be a very rewarding application.
Reference


