ORIE 4741 Mid Term Report

Ziming Zeng & Tiansheng Liu

1 . Why do we do this project

The most popular hypothesis about financial market is the efficient market hypothesis, which states that the current price of an asset has reflected all of the available information. So, under this assumption, what drives the price? The answer is news. In other words, price of an asset reacts to new information. Therefore, analyzing news is essentially the core of modern finance. There are 2 approaches to analyze the news – monitoring news 24 hours a day, or design an algorithm to analyze the sentiment of news automatically. The problem with the first approach is that market nowadays react to news so rapidly that human cannot possibly catch up with its speed. For example, the VIX index skyrocketed in a few seconds after Trump's tweet about "Fire and fury". That leaves us with only the second approach and we believe this is the future of finance.

2 . Data we use

We use Thompson Reuters historical real-time news headlines data from 2007 to 2017. This gives us 1 GB size data, which we believe is definitely enough for training our models like 1-gram and 2-gram. But if you want to train like a 15-gram model, the news data alone will not be enough. The reason we choose Thompson Reuters is that they are famous for the speed of reporting news and the industry is actually using Thompson Reuters' service. Comparing to Thompson Reuters, other sources like Bloomberg, Wall Street Journal and Financial Times are too slow. For our project, we are interested in implementing trading strategies at a relatively high frequency. Therefore, speed means everything for us.

3 . Answers to possible questions

(1) Is your data corrupted: No. This is the advantage of using commercial news data from Thompson Reuters.
(2) How many features do you use: In our most basic model, we use a dictionary (we will explain in great details below) that has 2337 negative words and 353 positive words. Therefore, the total number of features we use is 2690. In our Model 2 and Model 3, we generate our lexicons and weights ourselves, so the initial total number of features are in millions. But we will cut the number of features according to their weights in the end.
(3) How do you solve overfitting: the weights we design will handle overfitting. This is illustrated in section 6. Most of trivial words will have weights close to 0 assigned to them.
(4) How will you test the effectiveness of your model: Cross validation.
(5) Why not include more sources of data: Including other sources may cause duplicities. For example: Thompson Reuters reported fire and fury in 12:00 am and WSJ reported it again at 12:05 am. In our Model 2 and 3, we need to tag return for the news, and if we don't have consistent time for the same news we will be in trouble.
(6) Why do you think market reacts to sentiment of news: First, market reacts to new information. This is a conclusion of efficient market hypothesis (EMH) and is also the foundation of finance. Second, it has been proven by enormous amount of papers.
(7) How do you trade assets that are mentioned in the news: We only trade liquid stocks that are mentioned in the news to avoid market impact. And we do this using a concept called Fuzzy Match. You can find a brief explanation in section 8.

4 . Our idea

Our idea can be basically divided into the following 2 steps. Details about each step can be found in following sections.

1. Given a news, analyze the sentiment of this news
2. Trade the company that is mentioned in the news according to the sentiment of the news and hedge the trade using S&P 500 futures.

5 . Step 1 Model 1

For the first step, there are a lot of ways to analyze the sentiment of the news. We take a step-by-step approach to develop a basic model first and extend our model to increase the prediction accuracy.

In our most basic model, we used a famous financial dictionary created by Loughran and McDonald. The reason we use this dictionary is that the sentiments of words in financial news can be quite different from our usual language. For example, "liability" may have a negative sentiment in our normal conversation, but in finance it is just a concept in

financial report with no particular sentiment. Loughran and McDonald created this list of positive and negative words using 10-K and 10-Q reports from SEC. It has been proven in the Loughran and McDonald (2009) paper that this dictionary also has good prediction power for financial news.

Given a sentence of news headlines, we tokenize the headlines into single word and match each word in the sentence with the Loughran and McDonald dictionary. After that, we generate our sentiment score of the sentence using the following equation:

$$Sentiment_i = \frac{\#\ of\ positive\ words\ in\ headline_i - \#\ of\ negative\ words\ in\ headline_i}{total\ \#\ of\ words\ in\ headline_i}$$

The intuition for the above equation is straightforward: if a sentence has more positive words than negative words, then the sentence will have a positive sentiment.

6 . Step 1 Model 2

The problem with the first model is that it assumes that every positive or negative word has the same weight. This is obviously not true. For example, 'crash' has a stronger sentiment comparing to 'drop'. Therefore, in this model, we will assign weights to different words.

We used a method put forward by Jegadeesh and Wu (2013) to generate weights of words. The intuition behind this is we assign weights for each word according to the market reaction to each headline that contains those words. Mathematically, the above idea can be expressed as follows:

$$Sentiment_i = \sum_{j=1}^{J} (w_j F_{i,j}) \frac{1}{a_i}$$

where J is the total number of positive and negative words in our lexicon, w_j is the weight of word j, $F_{i,j}$ is the number of occurrences of word j in headline i, and a_i is the total number of words in headline i. The only unknown is the above equation is w_j and we can estimate w_j using the following regression model:

$$r_i = a + b \sum_{j=1}^{J} (w_j F_{i,j}) \frac{1}{a_i} + \varepsilon_i$$

Since b is a constant, combining b and w_j together and write it as B_j gives us the regression model:

$$r_i = a + \sum_{j=1}^{J} (B_j F_{i,j}) \frac{1}{a_i} + \varepsilon_i$$

where r_i is the abnormal return of stock market caused by headline i. After getting $B_j^{\sim}$, we can get $w_j^{\sim}$ using standardization.

Why do we choose regression instead of classification? We have 10 years news data. It is impossible for us to tag them manually as 'positive' or 'negative'. Therefore, we have to tag them with returns. But if we only tag them as positive or negative according to returns, we fail to capture the magnitude of returns. Therefore, we choose to use multivariate regression to capture this magnitude.

7 . Step 1 Model 3

The problem with Model 2 is that single word cannot capture the effect of expression. The easiest example is 'not good'. The above 2 models will count 'not' as neutral and 'good' as positive. However, we know that this is a negative expression. Therefore, we introduced a concept called bigram to deal with this situation. If have time, we will probably test the trigram.

It is better to illustrate bigram using an example. Given a sentence 'Sam loves to eat hotdog', we can generate a list of bigram as follows: ['Same loves', 'loves to', 'to eat', 'eat hotdog']. Basically, bigram is a combination of the nearest 2 words in a sentence.
With bigram, we can use the same method discussed in Model 2 to generate sentiment scores.

8 . Step 2

We cannot simulate it in real-time because we cannot afford Thompson Reuters' real-time news service and getting real-time bid and ask quote from our brokers. If we cannot get this real-time data, simulating trading using lagged news data has no meaning. Therefore, in this step, we separate our data into training and testing data and back test of our strategy. So how do we trade the particular stocks that are mentioned in the news? The answer is a fuzzy match algorithm. We have collected all the company names from NYSE, NASDAQ, AMEX from NASDAQ and we make a Python dictionary with keys equal to the company names and values equal to its symbol. The problem here is that companies may not be mentioned in its full names in the news. For example, Apple Inc. may be called Apple in the news. That is why we need a fuzzy match algorithm here. Due to the 3 pages limit, we will not expand this concept here.

Another reason we want to discuss Step 2 here is that the market reacts to news so fast that even if we use algorithm to automatically generate sentiment and place orders, we are still too slow. When a news comes out, competitors like Citadel or Renaissance will all buy it. If we are too slow, their buying orders will lift the price and we will lose money. So the question is, how can we be faster?

Our answer is Google N-Gram. N-Gram is a corpus created by Google to do autocomplete. The idea behind N-Gram can be easily illustrated using an example. Given a sentence: 'The market is doing extremely well because FED has decided to lower the interest rate'. Denote the words in this sentence as $w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}, w_{13}, w_{14}, w_{15}$. Then the probability of this sentence can be computed as follows according to the chain rule.

$$P(w_1w_2w_3w_4w_5w_6w_7w_8w_9w_{10}w_{11}w_{12}w_{13}w_{14}w_{15}) = P(w_1)P(w_2|w_1)P(w_3|w_1w_2)P(w_4|w_1w_2w_3)P(w_5|w_1w_2w_3w_4)\dots$$

So if we are considering a 2-Gram case, we will assume that the probability distribution of the next word is determined by the probability distribution of the last 2 words. Therefore, the above equation can be rewritten as follows:

$$P(w_1w_2w_3w_4w_5w_6w_7w_8w_9w_{10}w_{11}w_{12}w_{13}w_{14}w_{15}) = P(w_1)P(w_2|w_1)P(w_3|w_1w_2)P(w_4|w_2w_3)P(w_5|w_3w_4)P(w_6|w_4w_5)\dots$$

The beauty of N-Gram is that I can predict the probability of the next word using the past N words. And, for example in the 2-Gram case, if we know that: $P(well|\text{doing extremely})$ is really high like 95%. Then we don't need to wait to read through this news to place orders. Right after we have finished reading 'doing extremely', we are in. This will give us an edge in speed. For the above example, we can save $\frac{2}{3}$ of reading time. Due to the limit of computation power, we will only try 2-Gram case.

9 . What we have done

By locating xpath in source code from Thompson Reuters website, we use Python and Scrapy to collect the historical real-time news headlines data from 2007 to 2017.

We have also finished the data cleaning of our news data and the generation of sentiment scores in our basic Model 1. The graph below shows a normalized distribution of our sentiment scores across different years. About 80% of our news have weak sentiment scores close to 0 according to Model 1. The mean of sentiment scores is above 0 overall. However, we observe a fatter tail on the negative sentiment part than the positive sentiment part and this is true for all time intervals.