Christopher Lang, Jingqiao Huang, Fang Ye

May 4, 2017

Abstract

Our team analyzed two-year Twitter data using PySpark parallel programming. We deployed simple linear regression model to predict stock prices and further conducted portfolio optimization using CAMP model and Markowitz Mean-Variance Optimization based on the following five stocks: IBM, MSFT, YHOO, SBUX, and NVDA

Using twitter to forcast stock prices

Big Data Project C

**EXECUTIVE SUMMARY**

In this executive summary, we will explain in detail how we utilized parallel programming to read the files, clean the data, allocate the assets, and optimize our portfolio.

**File Reading** We decided that the initial data ingestion should start with RDD instead of Spark’s DataFrames, because RDD is as close to Python’s basic data structure as possible. Twitter data are not simple key-value combinations, they have a seemingly simple but deep hierarchical layout that is not suited for storage in a DataFrame. Hence, for ingestion, we utilized Spark textFile function to ingest tweets as strings and to take advantage of RDD’s distributed nature. The ingestion process was slow as Spark had to decompress the files and read the values. To ensure that each executor had enough data to work on at a time, we re-partitioned the data so that each executor had a reasonably sized chunk. In our case the partition size was selected to be the number of executors multiplied by 4 (based on our experience, 3 or 4 was commonly used).

Furthermore, each bz2 compressed file contained roughly 2,000 JSON documents in a line-delimited format. Each of these JSON documents are pushed through Python’s built-in JSON module to convert it to a dictionary of values using a map function.

**Data Cleaning**  To clean the data, we first ran preliminary filter functions. Several JSON documents suggested that they were deleted, hence we assumed that documents with the “delete” key were to be removed. Another filter function specifically looked for the “lang” key. Several JSON documents didn’t have this tag and was simply removed. With those two filters, we filtered out English tweets for further analysis. A separate feature generator function was used to create features suitable for DataFrame. This included term frequency counts, “@” mentions, hashtag counts, various punctuation counts, and even mentions of money.

All that has been discussed was developed on a Hortonwork’s virtual machine using Spark 2.1.0 and Anaconda’s Python 3.5.3, with 7 cores and 12GB of RAM. Ingestion was a painfully slow process. Therefore, to minimize wait times – for instance, ingestion took 4 to 5 hours on a 45GB set of compressed JSON files – we ensured that we persisted the data after the RDD ingestion. Running count, we retrieved approximately 37 million JSON documents from archive 09, despite that our archive 10 was corrupted.

**Asset Allocation** Due to time constraints and constant crashes, we decided to simplify our strategy to one that tests a statistical fit on a simple Linear Regression model and allocates assets based on term frequencies of each stock being mentioned in tweets. We’ve found that doubling of tweets about a particular company over a 60 second period is associated with an increase in the company's trading volume and an increase in its price. The goal was to associate the number of frequency counts to adjusted stock closing prices, and further, to predict closing prices.   Some other ideas include that we would normalize the data into a sliding window, e.g., term frequency for the past 1 hour, but it was not suitable as we only had closing prices for a day or just a partial count.

We calculate the term frequencies of the five stocks mentioned in tweets for each trading date and allocate the capital according to the rank of the term frequencies. If a stock is mentioned the most times in the tweets, we invest 30% of our capital on this stock. Then we allocate 20% of our capital to the stock for which the mentioned times rank second and third respectively. 10% of the capital goes to the stock for which the mentioned times rank fourth. We keep the 10% of capital in cash.

**Sentiment Analysis** We used TextBlob to conduct a simple sentiment analysis on the ‘text’ part and translated the texts into “Positive”, “Negative”, and “Neutral” categories. Along with the frequency counts, we pay particular attention to the stocks which have not just the greater popularity, but also greater amount of either positive or negative tweets. We may put a long position on the positive popular one(s) and a short or hold position on the negative popular one(s).

**Portfolio Optimization** The primary packages we deployed include numpy, pandas, and scipy. We started out by obtaining the daily adjusted closing prices for the following 5 stocks, IBM, MSFT, YHOO, SBUX, and NVDA, respectively, which dated from April 27, 2010 until April 27, 2017, through *Yahoo! Finance* using the Remoted Data Access function in pandas\_datareader module. Dow Jones Industrial Average (DJIA) daily adjusted closing prices within the same timeframe were collected as well to serve as a benchmark to regressed upon for regression analysis. We referred to DJIA instead of other indices such as SP 500 or Russell 2000 was because all 5 stocks are large companies having relatively stable cash inflows and similar impact on the market as blue chips do.

The next step was to use the frequency counts of each stock being mentioned in the tweets as a popularity indicator and rank the 5 stocks in the SBUX, YHOO, MSFN, IBM, and VNDA . Because short-selling is prohibited, we decided to consider two scenarios, which are (a) fully distributing all the funds to the five stocks, and (b) saving a portion of cash as a risk-free instrument with a given risk-free rate of 5%. With a target return of 25%, we then optimized each model using Markowitz Mean-Variance Optimization as well as comparing Minimum Variance Portfolio (MVP), Efficient Frontier (EF), and Sharpe Ratios to find the better strategy. We defined functions to get each of those values. Again, without short-selling, we need to add constraints and upper boundaries to the optimization models, such that all parameters are within [0,1] interval, i.e., the weights are nonzero numbers and WIBM + WMSFT + WYHOO + WSBUX + WNVDA = 1, and that the optimal model comes from those under the EF. To include a risk-free asset, which could be equivalent to cash savings in the bank, if buying additional assets such as Treasury bond is not an option, we also looked at Capital Market Line (CML) to find the tangency portfolio with the best variance-risk combination.

The optimal fully funded portfolio is the one with zero investment in IBM, 9.54% in MSFT, 1.91% in YHOO, 65.89% in SBUX, and 22.66% in NVDA. Its associated Sharpe Ratio is 0.0653. The optimal portfolio with risk-free component is constructed as 1.55% in IBM, 21.34% in MSFT, 7.00% IN YHOO, 51.96% in SBUX, and 15.14% in NVD, with 2.04% saved as cash. The Sharpe Ratio is 1.0200. As is seen here, the second strategy performs better with a risk-free instrument, as its Sharpe Ratio is much higher, indicating that with the same amount of risk exposure, the second portfolio enjoys higher return.