StockTwits Sentiment Classifier

Retrieving Messages Using StockTwits API

StockTwits[1] is a social media platform, similar to Twitter but designed exclusively for investors and traders to share opinions about financial assets such as stocks, currencies, commodities. A large number of recent studies has utilized messages posted on this platform to measure sentiment in the financial markets, such as stock market[2] and cryptocurrency market[3]. The advantages of StockTwits in such a task includes:

- It is a popular platform sharing only about financial assets.
- It supports a wide range of financial assets (currently around 2500 tickers are being supported, among those are 532 cryptocurrencies)
- It has a public API that allows automatical retrieval of users' historical messages.
- Most importantly, StockTwits has an optional feature for its users to tag some of their posts as "*Bullish"* or *"Bearish*". Conceptually, this feature makes applying supervise learning algorithms very appealing because we naturally have a large labeled dataset. Oliveira et al., (2016); Renault, (2017) have exploited this to build lexicons that are specific for the stock market. Chen et al. (2019) also follows those studies to build a crypto-specific lexicon. All of the above authors have proved that their lexicons significantly outperform most of standard lexicons.

[1] https://stocktwits.com/

[2] See Oh & Sheng (2011), Oliveira et al. (2013), and Renault (2017)

[3] See Chen et al. (2019)

In this application of Random Forest, I use the StockTwits API to collect messages exclusively about cryptocurrencies between 2014/11/28 - 2020/07/25. The dataset contains a total of 2,045,322 messages (~287MB) from 48,648 distinct users and related to 519 cryptocurrencies. The source code for collecting messages is included in the code folder (*get_stocktwits_message.py).* Because the StockTwits API imposes call limits, it would take several days to run the code and achieve every messages inside the timeframe of our analysis. Thus, I also attach the data inside the project folder.

The next step is to create the labeled dataset for our Random Forest model later. Thus, the messages with sentiment tagged as *"Bullish"* or *"Bearish"* are separated out from the original dataset. Thus, there remains a total of 999,720 messages, in which we have 818,452 *"Bullish"* and 181,268 *"Bearish"* messages. In other words, our dataset is labeled in an unbalanced way. This imbalance implies that investors are often optimistic when they share opinions on social media, which has been documented empirically by studies such as Avery et al. (2011) or Kim & Kim (2014).

Text Pre-Process

Next, some cleanings are applied on the collected messages so that only relevant information remains. This procedure, often called *text pre-process*, takes place in this following order:

- First, all messages are lowercased.
- For some reason, messages collected from StockTwits API have all punctuations represented by their HTML codes instead of Unicode (human-readable) characters (for example, "!" was expressed by "!"). All of these HTML codes are converted back to Unicode format.

- All words with more than 3 repeated letters (such as “hellooooo”), which is very common among expressions on microblogs, are shrunk to a maximum length of 3 (i.e. hellooo).
- Currencies hashtags (such as “$BTC.X”, “$ETH.X”), money values (such as “$10k”, “€200”), hyperlinks (such as “http://stocktwits.com”), numbers (such as 10,000 or 5k), usernames (such as “@username”) are replaced respectively by the word “cashtag”, “moneytag”, “linktag”, “numbertag”, and “usertag”.
- Next, all stopwords such as “I”, “he”, “she”, “it”, “herself”, “himself”, “the”, “an”, “a” etc. are also removed.
- All punctuations are removed except “!” and “?”.
- Lastly, the prefix "negtag_" is added to any word consecutive to negative words such as “no”, “nor”, “isn’t”, etc. (the complete list of 43 negative words is included in a self-created .csv file). For example, instead of “can’t fly”, we have “negtag_fly”.

This text-processing methodology is inspired mostly by Chen et al. (2019) and Renault (2017). The main idea is to remove all words that are unnecessary for classifying sentiment of a whole sentence. However, I modified their procedures by adding some steps (such as escaping HTML symbols, having a more detailed list of stop and negative words). The source code for pre-processing messages is included in the code folder (*pre_process.py)*

Examples of messages that have gone through the process are as below.

Before Processing	After Processing
$BTC.X wow this is way ahead of binance!	cashtag wow way ahead binance!
$BTC.X needs to hold above 10,200.. I really don't want to be bored by a 2 day consolidation between 10,000-10,200	cashtag needs hold numbertag really don't want bored numbertag day consolidation numbertag numbertag
$1ST.X Got fouled, this is a POS. need to hodl cos I'm fucked at the moment with this one. Will take 5 years to break even	cashtag got fouled pos need hodl cos fucked moment one take numbertag years break even
$SPY wallstreet is piling into bitcoin and crypto. Smart money leaving before the crash $BTC.X	cashtag wallstreet piling bitcoin crypto smart money leaving crash cashtag
$BTC.X to all the bitcoin bears out there.	cashtag bitcoin bears there

Text Vectorization

From this point, every step is coded in the file *rf_sent_classifier.py.*

After pre-processing, it is time to transform the text into numeric features so that we could apply Random Forest. One of the most popular methods for text vectorization, the *TF-IDF (Term Frequency - Inverse Data Frequency)* is employed at this stage of the analysis. In essence, the method can be described as follows:

- First, create a list of every distinct word inside our textual document (often known as the *bag-of-words*).

- Second, calculate the *Term Frequency* of every word in all messages. This metric measures how frequently a term occurs in a sentence. For any term t inside a sentence s, its *term frequency* can be defined as:

$$tf_{t,s} = \frac{Number\ of\ times\ term\ t\ appears\ in\ sentence\ s}{Total\ number\ of\ terms\ in\ the\ sentence\ s}$$

- Third, compute the *Inverse Document Frequency* of every word. This measures how important a term is and can be defined as:

$$idf_t = ln(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ t\ in\ it})$$

- Finally, the *TF-IDF* weight is simply the product of the *Term Frequency* and the *Inverse Document Frequency*.

A simple example of this method can be given as follows:

*Consider a sentence containing 100 words wherein the word 'cat' appears 3 times. The TF for 'cat' is then (3 / 100) = 0.03. Now, assume we have 10 million sentences and the word 'cat' appears in 1000 of these. Then, IDF is calculated as ln(10,000,000 / 1,000) = 4. Thus, the TF-IDF weight is the product of these quantities: 0.03 * 4 = 0.12.*

In our case, due to the presence of a large number of messages (999,720), the bag-of-word*s* is a huge vocabulary that contains 189,924 different words. The final dataset of the vectorized messages thus has a large dimension (999,720 x 189,924), thus it is very time-consuming to apply a Random Forest model on this raw dataset.

Dimension Reduction

As done by Ljungberg (2011), one of the feasible solutions is to use dimension reduction methods. The problem is that our final dataset is a very large sparse matrix (meaning that there are many gaps present in the matrix), and *Python* does not offer an efficient

way to apply the familiar methods such as *Principal Component Analysis (PCA)*, or *Factor Analysis,* so it doesn't cost tons of memory and result in an error. Instead, the *sklearn package* provides a better method to work with sparse data, which is called *Truncated Singular Value Decomposition (SVD)* or *Latent Semantic Analysis (LSA).* Detailed documentation on this technique is further provided on the *sklearn*'s homepage[4].

I decided to reduce the number of features from 189,924 (distinct words) into 100 components. The total explained variance ratio of 50 components is 24.87%.

Finally, we have a dataset of 999,720 messages, each numerically converted to a vector of (1x50). According to the rationales of the application case that predicts the corporate's default rate, I divide the unbalanced dataset into the training and test set as follows:

- The training set consists of 90,634 *"Bearish"* messages and 90,634 "*Bullish"* messages (all are randomly selected)
- The test set consists of the rest messages, including 90,634 *"Bearish"* messages and 818,452 "*Bullish"* messages.

Random Forest Classifier

Applying a *Random Forest Classifier* model on the training dataset results in the confusion matrices as follows. Initially, the base model has the hyperparameters as follow: the number of trees is fixed at 500, the max depth of the tree is 50, the fraction of sample used for every iteration of bootstraping is set at 75%. The time to run this model on my laptop (with all cores running) is about 4min53s. The prediction accuracy on the test set is currently at 74.75%. The OOB score is 70.95%.

[4] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

		Actual Classes	
		Bearish	Bullish
Predicted Class	Bearish	82,208	8,426
	Bullish	5,269	85,365

Table 1: Confusion Matrix (Training Set)

		Actual Classes	
		Bearish	Bullish
Predicted Class	Bearish	59888	30747
	Bullish	175937	551880

Table 2: Confusion Matrix (Test Set)

References

Avery, C., Chevalier, J. A., & Zeckhauser, R. J. (2011). The "CAPS" Prediction System and Stock Market Returns. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.1404750

Chen, C. Y., Despres, R., Guo, L., & Renault, T. (2019). What Makes Cryptocurrencies Special? Investor Sentiment and Return Predictability During the Bubble. *SSRN Electronic Journal*, 1–36. https://doi.org/10.2139/ssrn.3398423

Kim, S. H., & Kim, D. (2014). Investor Sentiment From Internet Message Postings and the Predictability of Stock Returns. *Journal of Economic Behavior and Organization*, *107*(PB), 708–729. https://doi.org/10.1016/j.jebo.2014.04.015

Ljungberg, B. F. (2011). *Dimensionality Reduction for Bag-of-words Models: PCA Vs LSA*. 1–6. http://cs229.stanford.edu/proj2017/final-reports/5163902.pdf

Oh, C., & Sheng, O. R. L. (2011). Investigating Predictive Power Of Stock Micro Blog

Sentiment In Forecasting Future Stock Price Directional Movement. *International Conference on Information Systems 2011, ICIS 2011*, *4*, 2860–2877.

Oliveira, N., Cortez, P., & Areal, N. (2013). On The Predictability Of Stock Market Behavior Using Stocktwits Sentiment And Posting Volume. In *Portuguese conference on artificial intelligence* (pp. 355–365). https://doi.org/10.1007/978-3-642-40669-0_31

Oliveira, N., Cortez, P., & Areal, N. (2016). Stock Market Sentiment Lexicon Acquisition Using Microblogging Data and Statistical Measures. *Decision Support Systems*, *85*, 62–73. https://doi.org/10.1016/j.dss.2016.02.013

Renault, T. (2017). Intraday Online Investor Sentiment and Return Patterns in the U.s. Stock Market. *Journal of Banking and Finance*, *84*, 25–40. https://doi.org/10.1016/j.jbankfin.2017.07.002