Sentiment Analysis using PySpark on an EC2 Instance in the Cloud

By:
Ryan Nichols 29787739
Arshad Saidoo 27763441
Michael Scalera 27079834

For:
Prof Omar Abdul Wahab

Due:
Nov 29th 2018

Twitter Streaming Sentiment Analysis using Spark

Ryan Nichols
lConcordia University
2978773

Arshad Saidoo
Concordia University
27763441

Michael Scalera
Concordia University
27079834

***Abstract*—This project is a demonstration on how sentiment can be determined from live streamed twitter data in a cloud application. To accomplish this an AWS EC2 instance is provisioned with the required libraries to run PySpark jobs. A python script uses the tweepy python library to pipe tweets live from twitter and writes them to a port on the EC2 machine. A separate script connects a PySpark data pipeline to that port, filters out unusable terms, continuously updates a sentiment value and writes it to a text file. A third script reads the latest value written to the text file and updates a cloud hosted NodeJs server so the results can be shared via API calls.**

Keywords—PySpark, spark streaming, Twitter, Sentiment Analysis

I. Introduction

This project was the students first introduction to making a Spark application, specifically using PySpark. The learning curve for using PySpark on an EC2 instance is steep and when it came time to start coding many obstacles were overcome. From getting outdated example code working to attempting to update our cloud server with our results using multithreading in a PySpark application much was learned about the challenges of using this framework and ultimately we found success.

A. Overall Goal

The overall goal of the project was to use Twitter as a data source for a data pipeline that would give some insight in the world of crypto currency markets by using the data pipeline to determine sentiment.

B. Objectives

Our first objective was to connect to the data source, Twitter, and get access to live tweets and supply them to PySpark as a data pipeline. Our Second objective was to find a list of words that would indicate positive sentiment and a list of words for negative sentiment. Since the world of cryptocurrency has its own lingo we also needed to augment these lists with words specific to this topic that would also indicate the sentiment of the words in our pipeline.

Our third objective was to get a PySpark job connected to this data source. From there we needed to filter all tweets down to just the ones of interest, tweets containing the keywords ‘BTC’ or ‘btc’. We then needed to determine, for each word in the pipeline, if the word was relevant and may have some bearing on the overall sentiment for our chosen keywords so that a value for sentiment could be updated.

Our fourth objective was to store our results using a cloud server so that users could access them from anywhere in the world.

C. *Problem Statement*

We were tasked with making a recommendation system. We chose an uncoventional way to accomplish this task, since our reccomendation will come in the form a sentiment regarding the cryptocurrency markets. Traders can use our sentiment as one of many things to consider when making trades and therefore increase their potential profitability.

D. *Assumptions*

Our assumptions are that within tweets about a certain topic, if more words that indicate a positive sentiment than negative sentiment are written that the overall sentiment of the tweet will be positive and vice versa. We also assume the volume of the positive or negative words also reflect the overall sentiment.

E. *Methodology*

The technologies used in this project are an AWS EC2 instance for hosting the PySpark jobs and running python scripts and Heroku for hosted the NodeJs server.

The tweepy libary was used to connect to a live twitter stream and piped the data to a port on the EC2 instance. A PySpark data pipeline connected to this port. The pipeline filtered for words that matched either of our two lists of relevant words. Relevant words were stored in two text files, positive.txt and negative.txt. Originally these lists were found pre-made[1] but needed to be augmented with terms specific to crypto.

To do this we attempted to examine the best and worst days preformancewise for BTC, examine the tweets from the previous day, do a spark word count to find the most popular terms.

Task	Deadline (Week of)
1- Cloud infrastructure setup	October 8th
2- Sentiment Analysis	October 29th
3- Twitter data spark streaming	October 29th
4- API Node Server	November 19th

II. Project Description

A. Function Provided by the Service

The function our service provides is access to the sentiment of the latest tweets regarding BTC which should give the overall sentiment of the crypto markets which which correlate strongly with BTC. This should in turn allow traders to make more informed decision when making trades. They could also create their own apps that use our API to get the latest sentiment and notify them of any abrupt changes so if something positive or negative regarding these markets happens they will know more quickly and can therefore react more quickly.

B. *Architectural Design*

Our architectural design in figure 1 in the appendix shows us the inputs and outputs as well as the main components of our system. First, we focus on inputs and outputs, the main input to our system is the tweets that will be analyzed for sentiment. The users interact with the service via the browser and or smart phone, connecting to the Heroku hosted NodeJs server via an API call.

The main sentiment calculations will be conducted on the EC2 instance described in the design. Here our Spark pipeline will conduct the sentiment calculations

and will update a global text file called score, this will hold the sentiment score based on passed, and current calculations. The score is sent to the server using the requests libary in a python script. The server stores it in MongoDB (mLab) to facilitate the fetching of the score by the server to be served as a response in API calls..

C. *Technical implementation*

Every tutorial we could find that uses live streaming twitter data used exactly the same code for connecting using tweepy and writing the data to a port so we didn't have different examples to compare and come up with our own solution so we used that part of the code as is[5] and the file is TweetRead.py. This writes the tweets to a port so we need a script to connect to the port, filter and map the datastream which is output.py.

In that script we read two text files positive and negative, the words from those files are added to lists. We then define our filter function is_useful() that checks if the word in our data pipeline is in either list. We then define our map function update_score() which checks if the word is in the positive list, if so we increment the score for sentiment if not the word must be in the negative list therefore we decrement the score. In this function we also output the latest score to the score.txt file. In make_request.py we just need read the values in this text file, take the last value from this file and make an api call to our server using the requests library.

The NodeJs server uses the express framework to expose GET and PUT requests so we can update the server with the latest sentiment and our users can access this value using the GET request. This is hosted on Heroku.

To make our sentiment as accurate as possible we added to the dictionary (positive and negative text files) using actual tweets for a specific crypto currency (in this case BTC). First step was to retrieve tweets using hashtag and dates using the Tweepy library. We had a few problems here, mainly due to the limitations of the twitter API. We are only able to retrieve 100 tweets at a time, and only for the last 7 days. We had another python script that would look at spikes and drops in BTC price over the last 7 days, and if there was a spike we would update positive text file and drops would be for negative. In our case we only had drops (in the 7 day span for when we ran the script), therefore the end result would only update the negative text file. The next step was to update the date in the twitter script that would retrieve tweets with the date of the price drop. The script would place all the tweets in a csv file.

Now comes PySpark, our saved tweets in the csv file would be fed to our PySpark pipeline and our batch jobs would be to create tuples and count the number of recurring words, the script would then output all the tuples to another csv file.

Once the csv file was created by our PySpark pipeline, we use a final script that would sort the list of tuples in incrementing order and take any word that recurred 4 or more times and add it to the negative text file if that word didn't already exist in positive or negative file. If there was a price spike, this script would do the exact same but update the positive text file instead.

D. Project Source Code

https://github.com/Scabandari/Sentiment_Analysis

III. DISCUSSION

Before starting the sentiment analysis for this project we wanted to take a sample project using PySpark, the twitter API and the python library tweepy and get that running on an EC2 instance just to know that we had all the correct libraries installed so we could start working on the project. This caused a lot of problems because we were using on old tutorial from Udemy [2] and it didn't work. We struggled with a py4j error for about 2 weeks. We found a question regarding our exact problem on stackexchange.com but it had no answers. It wasn't until finally looking for tutorials on how PySpark works [3] that we realized all we needed to do was filter the data stream using the filter function and use the map function to determine how each word in the filtered stream affected the sentiment. Either the word would be in our positive list and we would increment our sentiment by one or the word would be in the negative list and we could decrement the sentiment by one.

We attempted to use multi-threading in the same script that was using PySpark and from the new thread we attempted to sent an API call using the python library requests. It didn't work and we think there are issues around using multi-threading with PySpark since it already has parallelized operations.

Our solution was instead of sending an API call from that script to update the server with the latest sentiment, we output the sentiment to a text file and wrote another script to read all the sentiments in and take the latest value ie the last line and then make the API call to our server.

The security of the app should be very sound since we're implementing two kinds of API calls for our server. One put request that our app uses to update the latest sentiment and a get request that our users can use. If someone got the endpoint for making put requests they could update a false sentiment but since we're the only ones that know the endpoint it shouldn't be a problem. The app is scalable since anyone can access our sentiment via API call and millions of users should be no problem.

Our quality of service we would like to improve in the future since we're only looking at individual words when calculating sentiment. This is the unigram but ultimately we'd like to look at groups of words together since there is more information to be gleaned from the data when words are examined in context.

IV. CONTRIBUTIONS

Ryan: Setting up EC2 instance, installing libraries, getting Twitter stream working, connecting the PySpark to that stream, filter, map functions to get relevant data only and update sentiment and output the sentiment to a txt file. The part of the script that reads the sentiment from the text file.

Arshad: The part of the script that reads the sentiment from the text file and makes the API call to our server. Wrote the code for the NodeJs server[4] and hosted it on Heroku. Wrote a basic Android app that makes an API call to our server to get the latest sentiment.

Mike: The script that would retrieve texts by date and hashtag. Another script to count the recurring words using PySpark. And the final script to merge those recurring words to either negative or positive text files.

ReferenceS

[1] *https://github.com/endiliey/cs50/tree/master/pset6/sentiments*

[2] Spark and Python for Big Data with PySpark, https://www.udemy.com/spark-and-python-for-big-data-with-pyspark/learn/v4/content

[3] https://annefou.github.io/pyspark/02-mapreduce/

[4] https://scotch.io/tutorials/build-a-restful-api-using-node-and-express-4

[5] http://www.awesomestats.in/spark-twitter-stream/

APPENDIX:

Figure 1