Language Abstractions for Concurrent and Parallel Programming Uppsala University – Autumn 2017 Project Report

Nam Nguyen, Nevine Gouda

15th January 2018

Contents

1 Introduction

Upon airing the newest *Fifty Shades Freed* movie on this Valentine's day, Universal Studios has posted multiple short trailer on social media and received a great interaction from the audience. Given more than 60 thousands re-tweets on twitter, the production team now wants to know whether the audience's reaction was positive, negative or neutral. This tweet labeling task is called sentiment analysis. It is well employed to study the customers feedback and respond accordingly, or to study public opinion regarding current events (MemeTracker) or that a politician (e.g. Donald Trump).

The challenge Sentiment analysis could be done quickly by a human skimming through a short paragraph of a specific topic and deciding complex emotions that the text conveys such as satisfaction, frustration, anger, etc. However, for an algorithm that cannot understand the real meaning that words convey, automatic sentiment analysis becomes a challenging task. Instead of looking at all the words, a sentiment analysis algorithm recognizes and bases on the keywords in the text to decide the overall emotional status of the input.

In this project, we aim to implement a twitter sentiment analysis application using pySpark. This application analyzes the sentiment of a tweet that user inputted and the task is broken down to the following sub-tasks which will be elaborated later on in this report. Given the scope of this project we keep the implementation of the application on a local machine once and another time while utilizing Hadoop HDFS. This will be elaborated more in section 3 and section 4.

- Data collecting
- Data pre-processing
- Algorithms implementation
- Performance evaluation

The rest of this report is organized as follow. The use cases of how to compile and run the program, including key examples are presented in section 2. Section 3 is a description of important data structures, algorithms and functions. The program's performance and the benefits and drawbacks of our chosen concurrency abstraction are discussed in section 4 and section 5 respectively. Lastly, section 6 discusses the shortcomings as well as future works and the report is concluded with a description of which parts of the project each individual did in section 7 our experiments along with computational and comparative results.

2 Use cases

Our solution is presented in a directory with 1 data folder which is acting a local directory, 1 user folder which is acting as a local HDFS directory, 1 scripts folder which holds all our logic and 1 main setup.py file. Therefore in order to be able to setup any prerequisites to run the project please do the following:

1. Open up the terminal and cd (change directory) to the Project's root directory.
2. run: `sudo python setup.py install`.

Then in order to run the script using local files do the following:

1. Again to run the script go to the terminal and run
   `python scripts/Twitter_Sentiment_Analysis.py`.

2. It will then prompt a message whether you have a dataset that you want to test or just test with our test data. Thus enter "d" if you want our default dataset.
   `N.B.` Please note that if you want to test your own data set then please insert the data in the data folder and make sure that your file contains as a header the words "Sentiment" for the tweet's label column and "SentimentText" for the tweets columns, where each row should represent a tweet. The file can contain other columns with other names like; Tweet ID, but these two columns are the most important. While if you want to test our dataset please note that the data was obtained from Kaggle [1].

3. Then leave it to finish. Just note that any log that has "INFO:__main__:__TSA__:" is our logging, and anything else is pyspark's logging dump, and you can just ignore these.

4. When you see "Logistic Regression correctly classified the tweets with an accuracy of x%" where x is the accuracy then the first algorithm is done. And The same thing will happen with the two other algorithms.

5. When all the models are done training, the program will give you the option to insert a new tweet as a test tweet where you see how the program will predict its class with each classifier. For instance you can test "happy world" where it should classify it as a positive tweet.

6. Then you are also prompted with an option to see how many times a word exits in tweets classified as positive relative to tweets classified as negative. For instance if you write "happy" it will print:
   `The word happy occurred 146 times in Sentiments of label 1.`
   `The word happy occurred 66 times in Sentiments of label 0.`

While in order to run the script using HDFS do the following:

1. Both Hadoop and HDFS needs to be installed and setup in order for this code to work. Where we followed these links [2], [3] to set them up next to everything else we already installed. And given that pySpark, Hadoop and HDFS are properly installed, then you can continue.

2. Again to run the script go to the terminal and run
   `python scripts/Twitter_Sentiment_Analysis_hdfs.py`.

3. It will then prompt a message to enter the link for the dataset you want to test with. Please make sure that you enter the path properly. For example this a similar example to an hdfs path:
   `"hdfs://localhost:9000/user/small.csv"` where you would simply create a directory in hdfs called user with small.csv located inside.

4. Then leave it to finish. Just note that any log that has "INFO:__main__:__TSA__:" is our logging, and anything else is pyspark's logging dump , and you can just ignore these.

5. When you see "Logistic Regression correctly classified the tweets with an accuracy of x%" where x is the accuracy then the first algorithm is done. And the same thing will happen with the two other algorithms.

6. When all the models are done training, the program will give you the option to insert a new tweet as a test tweet where you see how the program will predict its class with each classifier.

7. Then you are also prompted with an option to see how many times a word exits in tweets classified as positive relative to tweets classified as negative. For instance if you write "happy" it will print:
   ```
   The word happy occurred 146 times in Sentiments of label 1.
   The word happy occurred 66 times in Sentiments of label 0.
   ```

3 Program documentation

3.1 The mathematics idea

The idea behind the application is simple. First we need to collect data of label positive/negative/neutral. From that point, the problem becomes a supervised classification problem. Since pySpark already supports some common algorithms used in sentiment analysis, we only concern which algorithm to use and how to pre-process data.

Data pre-processing We studied the related works to see what are the important features associating with sentiment analysis. A simple sentiment analysis algorithm using Naive Bayes solely concerns the keywords in a tweet [4]. However, other researches suggest that there are other interesting textual features. The following are two of the features we want to incorporate to the classifier. They were both promoted upon the limit of 140 characters for tweets on Twitter.

- Word lengthening: People tends to strengthen words to emphasize the emotion (e.g. *"thank youuuu"* or *"OMGGGGG"*). According to a research work detecting sentiment in microblogs, the amount of tweets having lengthened words is 17% of the total amount of tweets, which illustrates how common the feature is [5]. In order to do filter these words, we utilizes *pytypo* library to shorten the repeated character so that only three repeated characters are kept. According to that, *"happyyyyy"* and *"happyyyyyyyyy"* are both contracted to *"happyyy"*.

- Emoticons: Today, young people make a great use of emoticons to convey their emotion through texts [6]. Therefore, we decided to keep the emoticons when we eliminate the punctuations in the texts using *emot* library.

Choice of algorithms As discussed earlier, this is a supervised classification. Therefore, we employed common algorithms to solve this problem, including the following.

- Naive Bayes
- Support Vector Machine (SVM) with Stochastic Gradient Descent (SGD)
- Linear regression with Limited-memory BFGS (LBFGS)

Naive Bayes and SVM are used commonly for sentiment analysis as of supervised learning [7, 8] since they were reported to have a good accuracy. The last choice of algorithm is arbitrary. We added one more algorithm because we want to have a more general comparison. All these algorithms are implemented as a part of the pySpark sentiment analysis library.

3.2 Steps of the algorithms

The following elaborates four steps of the

Step 1 Data collection. To save time and effort put on data cleaning, we find datasets from *kaggle.com*, which are clean, in English and without noise (tweets are fully labeled). This will be elaborated more in section 4.

Step 2 Data pre-processing is done in several smaller steps. These come with an assumption that the data taken from step 1

1. Trimming and tokenizing text into words by space character
2. Lowercasing words
3. Extracting emoticons (as explained in subsection 3.1)
4. Removing punctuations
5. Stemming words (using library *nltk*)
6. Removing stop words (using library *nltk*)
7. Shortening lengthened words (as explained in subsection 3.1)

Step 3 Algorithms implementation are done accordingly to the tutorials to use the pySpark sentiment analysis library. Three algorithms were all implemented as explained in subsection 3.1 and they could be tested in the program accordingly to the instruction in section 2.

Step 4 Performance evaluation - this will be elaborated in section 4

3.3 Integrating Hadoop HDFS

First we implemented everything using local files as the data input. But at the end we tried and successfully installed *Hadoop* and HDFS locally as a standalone cluster. It was not possible to try the HDFS remotely as it requires to pay for the service for a service provider like *Cloudera*. Thus our work around was to trick the HDFS to be located locally instead of remote and then access the data from there. Thus each worker node will then be able to work on its partition given the spark job passed to the node. Putting in mind that a worker node in our case is just a processor. However we don't expect to see any improvement in the performance due to the *Hadoop* overhead, and that the files are still stored locally on one machine. However if the same exact code ran on supercomputer as a standalone cluster, or a cluster filled with commodity hardware then it is expected to notice performance enhancements.

4 Performance evaluation

4.1 Testing environment

Experiments were run on a MacOS High Sierra machine with a 3.5 GHz Intel Core i7 with 8 hyper-threading cores.

4.2 Dataset and adjustments

Two dataset used for the project are taken from *kaggle.com*, respectively denoted $D1$ and $D2$. Each dataset is described as follows.

The first dataset This very big data contained more than 150MB of data, thus we extracted only the first 2.56MB of the file (corresponding to 12900 first lines) to get a suitable input file [9].

The first four lines of $D1$ is as shown below.

```
ItemID,Sentiment,SentimentSource,SentimentText
1,0,Sentiment140, is so sad for my APL friend.............
2,0,Sentiment140, I missed the New Moon trailer...
3,1,Sentiment140, omg its already 7:30 :O
```

As shown above, the first line shows the meaning of each column in the table such that are *"Sentiment"* and *"SentimentText"* corresponding to the sentiment value and the according text are in the 2^{nd} and the 4^{th} lines.

This dataset does not separate between training and testing data so we split randomly the extracted file (of size 2.56MB) to two parts for training and testing of ratio 3:7 respectively. At this point, $D1$ is ready to use.

The second dataset This data is published by *Kaggle* for a Twitter sentiment analysis competition [1], which already include a training and a testing datasets of size 8.26MB and 24.92MB respectively. Therefore, $D2$ is ready to use without adjustments.

4.3 Experimental results

Runtime with different sizes of data

The performance evaluation in term of accuracy of three learning algorithms Naive Bayes, SVM with SGD and Linear regression with LBFGS on two datasets $D1$ and $D2$ are recorded in table 1. The accuracy varies in about 10%, and in fact, we have encountered inputs that were classified into contradictory classes by different models. Figure 1 illustrates such a case, where an inputted tweet *"what a life"* is concluded to be a negative by the model trained with Naive Bayes and positive by the one trained with SVM.

According to the table, it is also seen that the model using Naive Bayes is the most accurate one. To the best of our knowledge, the accuracy of sentiment classification could get up to 80%, thus we feel that the accuracy we achieved with Naive Bayes was good enough.

Figure 1: Contradictory results received from two different models using Naive Bayes and SVM with SGD

```
Enter a tweet to classify or just Q to quit:
what a life
INFO:__main__:We predict that you entered a NEGATIVE tweet using NAive Bayes!
INFO:__main__:We predict that you entered a POSITIVE tweet using SVM with Stochastic Gradient Descent!
```

Table 1: Performance evaluation of three learning algorithms Naive Bayes, SVM with SGD and Linear regression with LBFGS on two datasets $D1$ and $D2$

Work \ Participant	$D1$	$D2$
Linear regression with LBFGS	64.8157894737%	63.4024109441%
SVM with SGD	69.9473684211%	64.1879994582%
Naive Bayes	73.9210526316%	70.0121901666%

4.4 Hadoop HDFS in use

At Figures 2 & 3 you can see that *Hadoop* is actually working with one datafile in the HDFS and it is called small.csv which is obtained from *Kaggle* [1]. However due to lack of time and meaning we didn't go further with Hadoop and HDFS. As we only wanted a proof of concept that we can successfully connect pySpark with Hadoop and HDFS. Yet it was useless to keep going on with a standalone local cluster because it won't make any improvement in the performance with respect to time and speed. In fact it resulted in a worse performance due to the overhead.

Figure 2: Terminal Screen-shot that successfully lists the files in the HDFS directory

```
nl119-149-217:Project nevinemgouda$ hdfs dfs -ls /user/nevinemgouda
18/01/15 18:23:01 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Found 1 items
-rw-r--r--   1 nevinemgouda supergroup    1261623 2018-01-15 18:13 /user/nevinem
gouda/small.csv
```

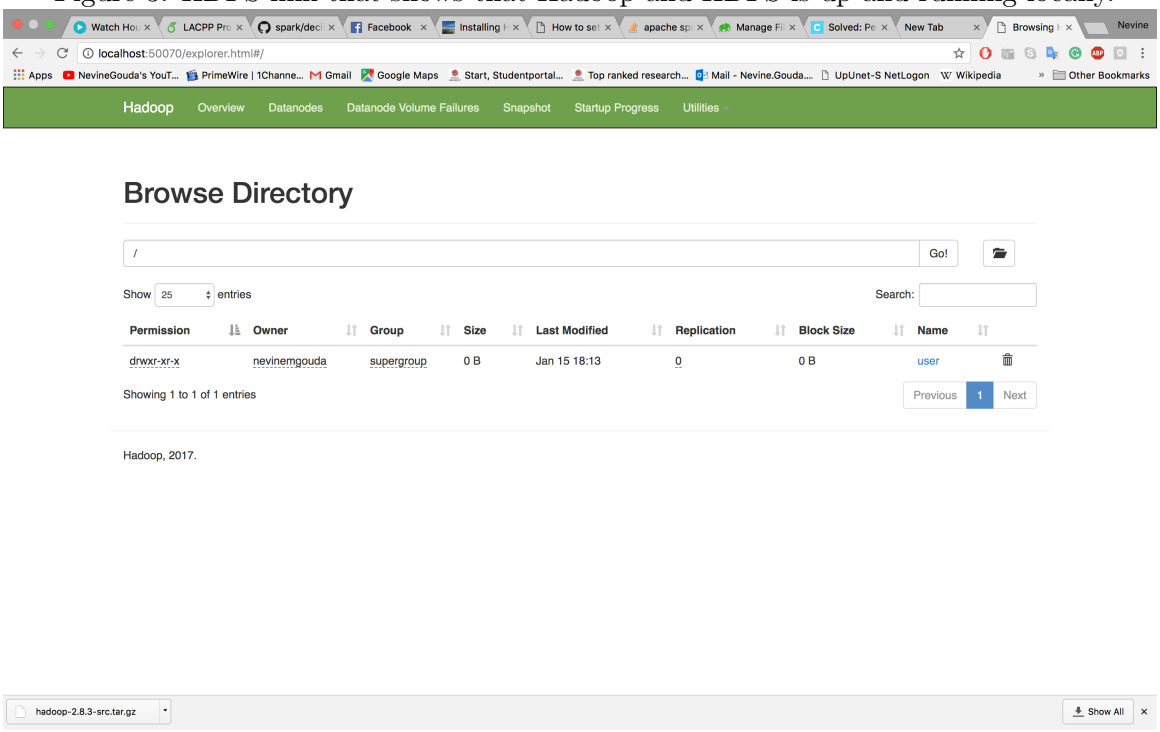
5 Concurrency abstractions

In this part, we discuss benefits and drawbacks of our chosen concurrency abstraction in the context of this project. The discussion will be elaborated individually by each student in the group in a separate paragraph. Overlapping is expected as two students elaborate on the same matter.

5.1 Elaborated by Nam Nguyen

To start with the advantages, I think using spark is a great choice for this project. The abstraction is not too low that we have to deal with data partitioning, data distribution, load balancing and node management (restarting when failed). Yet the abstraction is not too high that hides away the bottleneck of the calculation (storing and loading data to/from disk between every map-reduce operator) since spark tries to keep data in memory rather than writing to disk. Spark also provides highly flexible operators that free us from forcing everything to using map and reduce functions (compared to when using the MapReduce framework in python in assignment 3). This leaves the abstraction at a just-right level where the implementation is readable and clean. The set up of the project on Hadoop HDFS without changing much the code shows the great potential to speedup the program in distributed environment with little refactoring cost. If this sentiment analysis project was to use a lower level abstraction level programming, it would have been more struggling to handle the concurrency even right in the data cleaning part.

Figure 3: HDFS link that shows that Hadoop and HDFS is up and running locally.



As for the disadvantages, spark will not be unleashed to the full potential without being used in a distributed system where computation is done in different machines with their local data. With only one local machine that we implemented and ran the program on, the speedup is not seen while overhead costs are introduced, slowing down the execution.

5.2 Elaborated by Nevine Gouda

I believe that using *pySpark*, *MapReduce*, *Hadoop* and *HDFS* frameworks has many perks which I will be discussing in the next couple of paragraphs. Where for *pySpark* just encapsulates everything and saves the developer from a lot of hassle about handling threads, distributed systems, or even exception handling like if a worker node is not available. This is all like a gray box from the developer. And if the developer wants more flexibility for instance change the number of partitions or threads, worker nodes, reducers, scheduling, then all what one has to do it set up some flags and its ready to go. Thus the level of abstraction is really high and extremely easy to use. Another advantage for using pySpark is the amount of support for python AND spark is tremendous. We found many tutorials, examples and even sample codes that is provided by pySpark themselves.

Another advantage is while using *MapReduce* framework. Where all the tasks can be simply be broken down to map and reduce functions, with some tweaks or some extra Spark functionalities like *wholeTextFiles*, *saveAsTextFile* or *parallelize*. Thus it was very easy to read all the files using *wholeTextFiles*, pre-process the data using repeated transform(map,group_by, and reduce) functions to lower casing all the words, removing punctuations, keeping emoticons, stemming, removing stop words and even handle the elongated words. As well as creating a feature vector for each tweet. Which will simply be the input to train the model and finally test it with the test data.

However a drawback for working in this eco-system, is that if we don't have access to either a super computer or an actual cluster, not just one commodity hardware, then doing the tasks using MapReduce and Hadoop will actually be much slower that sequentially. That is because the overhead will be too much. Another drawback is that we weren't able to test the results with a really big data to compare the accuracy with bigger data. That is because testing the code with a big data on a commodity hardware would take a lot of time that it is considered infeasible. Simply because when we tried to run a file of size 100MB, it resulted in an error where the heap exceeded its maximum size. And even if we can handle such error, it will be really slow.

6 Known shortcomings

In this part, we reflect to our own implementation and discuss its major weaknesses. The shortcomings are listed as follows, and will be elaborated individually by each student in the group in a separate paragraph (covering different shortcomings). Less overlapping is expected compared to section 5 as two students elaborate on the different major shortcomings with deeper insights.

- More algorithms
- Running on local machine

6.1 Elaborated by Nam Nguyen

One of the big concern with the algorithms implementation we encountered was whether to implement the algorithm from scratch so that we can see how helpful spark/mapreduce is instead of using the library algorithms. But also then from the goal of the project itself, implementing an algorithm that is ready to use is completely redundant. Eventually we decided to just use the library algorithms. We could have used more algorithms to compared the results if there were more time.

It is also worth noting that the algorithms are totally based on the dictionary of tokenized words and without taking the context into consideration. This does not work well with input of complex sentiments, such as *"the food was terrible but the service was amazing"*. This will need more well-designed algorithm to cope with.

6.2 Elaborated by Nevine Gouda

One of the shortcomings is that we didn't explore *Hadoop* as much, thus we weren't able to test the performance in respect to time and speed. Meaning that we weren't able to play around with the resource managers, node managers, data nodes or even the number of partitioning or threads. That is due to the lack of a cluster and actually lack of time as we left the *HDFS* part at the very end in case we had time for it. In addition we wanted to try a big file and check whether the accuracy improves as the size of the input data increases but we weren't able to as stated previously in the report. An improvement that can be done in our code though is instead of having a map function for each stage, such as one map function for lower case, another one for stemming, yet another one for handling elongated words, and another one to remove stop words and so forth, it would increase the performance in terms of speed and time if we tried to reduce the number of maps and maybe do the lowercasing,stemming and removing stop words all in one map stage. We believe that this will improvement the speed. We only left it the way it is for sake of code readability.

Table 2: Breakdown of the task and level of participant of each group member

Work \ Contributor	Nam Nguyen	Nevine Gouda
1. Data collection	X	X
2. Data pre-processing	X	X
3. Algorithms	X	X
4. Performance evaluation	X	X
5. Using Hadoop HDFS	X	X
6. Report	X	X

7 Individual workload

Table 2 shows a breakdown of the workload and who contributed to which parts of the project.

References

[1] Kaggle. Twitter sentiment analysis. [Online]. Available: https://www.kaggle.com/c/twitter-sentiment-analysis2/data

[2] MAREK. Installing hadoop on mac part 1. [Online]. Available: https://hortonworks.com/hadoop-tutorial/using-commandline-manage-files-hdfs/

[3] Hortonworks. Manage files on hdfs via cli/ambari files view. [Online]. Available: https://hortonworks.com/hadoop-tutorial/using-commandline-manage-files-hdfs/

[4] nisarg64. Sentiment-analysis-pyspark. [Online]. Available: https://github.com/nisarg64/Sentiment-Analysis-Pyspark/blob/master/text_analysis.py

[5] S. Brody and N. Diakopoulos, "Cooooooooooooooollllllllllll!!!!!!!!!!!!!!: using word lengthening to detect sentiment in microblogs," in *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics, 2011, pp. 562–570.

[6] K.-L. Liu, W.-J. Li, and M. Guo, "Emoticon smoothed language models for twitter sentiment analysis." in *Aaai*, 2012.

[7] B. Pang and L. Lee, "A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts," in *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2004, p. 271.

[8] A. Go, L. Huang, and R. Bhayani, "Twitter sentiment analysis," *Entropy*, vol. 17, p. 252, 2009.

[9] David. twitter_sentiment. [Online]. Available: https://www.kaggle.com/ywang311/twitter-sentiment/data