Predicting Reddit Post Popularity

using Apache Hadoop and Spark

Narmin Orujova
University of Stavanger, Norway
n.orujova.uis.no

Nikisadat Abbasian
University of Stavanger, Norway
n.abbasian.uis.no

ABSTRACT

The identification of current trends and prediction of future trends using the socially generated big data is a new paradigm in computational science. This can naturally be applied in predictive marketing, which is technique involving usage of data analytics for determination of marketing strategies and actions with high probability of succeeding for businesses [4]. However, resulting from the massive size of the Big Data it is becoming computationally expensive to perform the analytics on a single machine. This paper introduces the implementation of Decision Tree and Random Forest algorithms using both Hadoop and Spark with preprocessing in Hadoop on large datasets with computational efficiency resulting from usage of cluster of nodes and high prediction accuracy due to the usage of Big Data.

Code: https://github.com/Narmin97/Group$_1$*5−Predictiong−Reddit−Post − Popularity.git*

KEYWORDS

Reddit Post Popularity, Hadoop, Spark, Decision Tree, Random Forest

1 INTRODUCTION

Reddit is a popular social media website where users submit posts containing either text or links, which other users can either "upvote" or "downvote" [16]. In the past years it has become increasingly popular with much of its traffic being driven by its voting system. The vote or score a post gets is derived by subtracting the number of "downvotes" from the number of "upvotes". Interestingly, the popularity of the posts is usually determined not by the topic, but by the discussion in the comments of the post [3]. Hence, by analyzing the features of the comments along with the features of the posts one can attempt to predict the score the post will get. To predict the popularity of the Reddit posts we can use Decision Tree and Random Forest. These algorithms are among of the best performing ones in regression tasks, however they also are computationally expensive when tested on large datasets. Solution for improved performance for these algorithms is the distribution of datasets and computation to the cluster of machines. The two widely used frameworks for a distributed development are Apache Hadoop and Spark, each being complement of each other [2]. Using these frameworks for preprocessing, training, and testing the massive datasets we can efficiently scale our project to get better performance both in the computational cost and accuracy of predictions.

Supervised by Tomasz Wiktorski and Jayachander Surbiryala.

Project in Computer Science (DAT620), IDE, UiS
2018.

To predict the Reddit post popularity using regression algorithms we want to use 3 different datasets. The dataset we have found on the Kaggle contains the features of Reddit Post till the 2017, which can result in irrelevant predictions. Therefore, apart from using this dataset we want to extract a new one with more recent posts using Pushshift API. After getting the dataset we want to implement the Decision Tree and Random Forest algorithms using Hadoop and Spark. If in case of Hadoop we have to write the algorithms ourselves using MRJob library of python, in case of Spark we can utilize the MLlib library. Both of these frameworks are using the MapReduce, the functional programming model. Therefore, implementation using these frameworks is different from standard scripting language implementation and requires extensive study and work.

The key contributions of this project are:

- To create a new dataset of Reddit Posts/Comments
- To apply EDA and correlation analyses to extract the features for regression algorithms
- To use Hadoop Streaming for Preprocessing
- To implement the Decision Tree using MLlib library of Spark
- To implement the Random Forest in 2 different way, using MRJob library of python and MLlib library of Spark
- To compare the calculational performance and accuracy of both implementations

2 BACKGROUND AND MOTIVATION

This section gives the background information on the main constituents of the project. First, we will give here the brief introduction to use cases of the project along with introduction to the Reddit website and its posts. Then introduce the related work done before to predict the popularity of Reddit posts. After we will describe the Decision Tree and Random Forest algorithms and why we chose these algorithms. We will conclude with presentation of used accuracy metrics and by giving the insights of the used datasets.

2.1 Use Case

Today's digital world is saturated with web content in the form of business opportunities for marketing companies and entertainment to internet users, hence not every item becomes popular. As a result, to boost the revenues companies invest a lot of money to identify the digital content which will become popular. Since the data generated on the web has massive dimensions using traditional single machine computation takes a lot of resources and result in deficient performance. Using the algorithms capable to process the data in parallelised manner can help to analyze huge amounts of data and get valuable insights. Motivated by this and considering that Decision Tree and Random Forest are good at analysing the categorical variables [13] we are going to use them for training the

model on the datasets and try to make predictions of the popularity for Reddit Posts.

2.2 Reddit Posts

Reddit is a enormous collection of forums for sharing the news or any other web content, which can be voted or commented by other users [15]. According to Alexa, it is the 18th most popular site worldwide [1]. Every post in the Reddit is under a category, known as *Subreddit*, where different topics are discussed. For example, in the */r/boardgames* subreddit people talk about board games, or */r/nba* subreddit is for discussion of National Basketball Association. Next to each post there are up and down arrows together with the number. Reddit users by clicking one of these arrows either upvote or downvote the post, hence increasing/decreasing the visibility of the post and the number reflects the sum of the votes. There are different attributes of the posts, such as gildings or in other term the awards the post has received and using these features together with the sentiment of post/comments and the engagement of the subreddit we are going to make the predictions of the score the post will get using Decision Tree and Random Forest algorithms.

2.3 Related Work

There have been several works, attempting to predict popularity of posts by using linear and non-linear machine learning approaches. Some of these works are listed in the following paragraphs.

"Reddit recommendation system" by Daniel POON, Yu Wu and David Zhang in 2011, can be considered as an early work in this field. They tried to predict number of upvote which a post can received in its lifetime by using multiclass classification with Naïve bays and SVM. The employed evaluation metrics were Classification error and RMSE scores applied on the classifier separately to compare their results [18].

In 2012 Jordan Segall and Alex Zamoshchin proposed a "predicting reddit posts popularity" using Machine learning methods such as KNN, SVM, BPMF-MCMC, and SGD to find best fit model. Then they implemented a model by linear regression combination of KNN, SVM, BPMF-MCMC, and SGD with a RMSE score equal to 0.49 which had lowest RMSE score among other algorithms. Moreover, by using this system each post could be ranked with its actual score [6].

Noah Makow, Matt Millican, Chenyao Yu delved virality prediction on reddit data to develop a "A Network Model for Reddit Post Virality Prediction" by using Random Forest and Boost Tree in 2017. They also focused on how predicting post score can be affected by Network structural features and temporal features. Original poster features and linguistic features were also included in the model. The obtained scores by using evaluation metric for all model were too high [9].

In 2018, a year after a huge reddit dataset became available on Kaggle, Prasanna Chandramouli, Radhakrishnam Moni and Varun Elgano implemented sentiment analysis on mentioned data by using MapReduce and Hive approaches to find meaningful correlation between various features of a post. They employed MapReduce for pre-processing and profiling data and Hive for retrieving knowledge from data by using powerful python semantic analysis tool such as NLTK(Natural Language Toolkit) and VADER(Valence Aware Dictionary Reasoner) [10].

Another work of multiclass classification on Reddit data is "Predicting Popularity of Reddit Posts" accomplished by Dylan Mendonca which was implemented by machine learning methods namely Random Forest and Logistic Regression as well as feature engineering approach. In addition, they utilized accuracy and F1 score as evaluation metrics which resulted 70% of score for Logistic Regression and even better 71% for multi class model [8].

Some of these works predicted the popularity of posts by analyzing only the metadata of the post and not considering the content and the commentary of the post. Others had a very small dataset; therefore, their results were not convincing enough if to consider the omitted correlation factors. In this project we have attempted to predict the popularity of posts using massive datasets of comments and posts combined together.

2.4 Decision Tree

Decision tree is a greedy search algorithm which recursively partitions the feature space. For each leaf node, which is the bottom-most partition it predicts the same label. To choose the partition point the algorithm greedily searches for the best splitting condition out of the possible ones by maximizing the information gain at the tree interior node. The diagram of Decision Tree algorithm is shown in Figure 1.

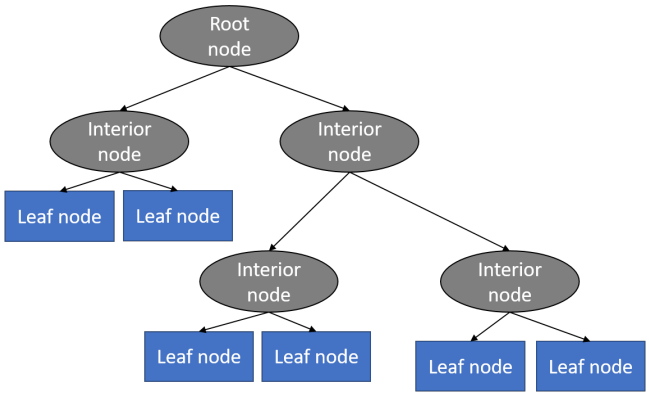

Figure 1: The Decision Tree Algorithm

At each tree node the split s is chosen from the set of split *(D,s)* in the dataset D, when:

$$argmaxsIG(D, s), \tag{1}$$

where IG(D,s) is the information gain of the node.

The information gain is computed from the node impurity measure, which is a measure of node values' homogeneity. Since in our case we conduct the regression analyses the impurity measure used is the variance of values on the node of the tree at each possible split. The information gain is calculated by getting the difference between the impurity of the parent node and weighted sum of its child nodes' impurities. If we assume a binary split, where the dataset D of size N is split into 2 child nodes D_{left} and D_{right} with

their sizes N_{left} and N_{right} correspondingly, the information gain is calculated as follows [13]:

$$IG(D,s) = Impurity(D) - \frac{N_{left}}{N} Impurity(D_{left}) - \frac{N_{right}}{N} Impurity(D_{right}) \quad (2)$$

Split candidates

In the case of continuous features the split candidates are determined by performing a quantile calculation over fraction of the data and based on that the ordered splits "bins" [13].

In the case of categorical variables for each value M of categorical variable there exists 2^{M-1} split candidates [13].

Stopping rule

The three main conditions to stop the recursive construction of tree are the followings:

- If the maximum depth of the tree is reached
- The split candidates do not lead to the increase in the information gain greater than a threshold
- The minimum number of children condition for the tree is reached

2.5 Random Forest

Random forests algorithm consists of the ensembles of decision trees. It one of the most successful machine learning algorithms for regression analysis. To reduce the overfitting they combine different decision trees. The basic structure of Random Forest is shown in Figure 2.

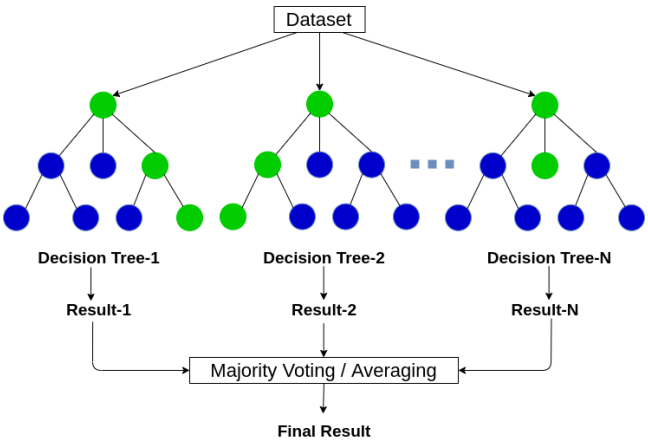

Figure 2: The Random Forest Algorithm

The algorithm behind them is as follows. In general definition the algorithm first defines the number of trees and builds corresponding number of trees based on the training data. Later, it combines the output of each tree to make the final prediction. However, rather than simply averaging the result of each tree, this algorithm uses 2 key techniques of randomness injection, which boost its performance and stands behind the notion of *Random* Forest in its name [7]. These techniques are:

- Random sampling of training data
- Random subsets of features

Random sampling of training data. During the training the algorithm constructs each tree from a random sample of given dataset. These samples can be drawn either with replacement technique known as *bootstrapping*, meaning that some samples can be used multiple times, or without replacement. The rationale behind this step is even if individual tree has high variance with respect to a particular set, since each of them are trained on different set of training data, overall, the forest will have lower variance without increasing the bias [7].

Random subsets of features. Using this technique algorithm splits the node in each decision tree using different subset of all features, hence, improving the performance. Generally for classification problems the number of selected features is equal to $\sqrt{number_{features}}$, but sometimes all the features can also be considered [7].

This scenario comes from the real life analogy of relying on multiple sources before making a final decision.

2.6 Accuracy Metrics

The accuracy metrics used for model evaluation in this project is Root Mean Squared Error (RMSE). Its formula is given by:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}, \quad (3)$$

where N is the number of records in the sample.

Its implementation both in Apache Hadoop and Spark is simple. In baseline model it is calculated using math library of python and in MLlib using the RegressionEvaluator() method of pyspark.ml.evaluation library.

2.7 Datasets

To train and test our model we have used 3 different datasets. First dataset was downloaded from the Kaggle website and can be found under the following link $https://www.kaggle.com/reddit/reddit-comments-may-2015$. In 2017 Reddit released an enormous dataset with around 1.7 billion of publicly available comments. Kaggle hosts a small portion of comments starting from May 2015 till the 2017. This dataset is a csv file with 59 218 270 rows and 22 columns describing the attributes of comments, such as subreddit, author, creation timestamp, controversiality, score and etc. In total it is 15 GB of size.

After getting this dataset, we have decided to create another one of smaller size, however with the most recent comments on the Reddit to get the up-to-date results. Additionally, we have decided to combine the information from the comments and posts dataset to obtain a wider feature space and more insights into the Reddit data.

There are 2 main ways to get the data from Reddit, one is directly using the Reddit API with Python wrapper praw and second is retrieving the data through the Pushshift API. Pushshift

(*pushshift.io/*) stores the database of historical Reddit submissions and comments which can be queried using the Pushshift API, for which Python has 2 wrappers [11]. The Reddit API is great but has the limit as its drawback. It allows to pull only limited amount of recent submissions or comments for a subreddit [11]. Hence, using Reddit API one is not able to create a large dataset. An alternative to this is Python wrapper *pmaw* for Pushift API, which allows to specify the parameters of the query to run, such as time interval and subreddit for which submission or comments should be retrieved [11].

To pull the data from Pushift API we have created a list of Reddit Subreddits, in total got 1801 distinct subreddits and with rate-limit of 60 requests per minute we have pulled the submissions on these subreddits for the last one month [12]. In total it took a whole day to get the results and code snippet used to pull the data is given in Listing 1.

```
import pandas as pd
import datetime as dt
from pmaw import PushshiftAPI

api = PushshiftAPI()

before = int(dt.datetime(2021,4,1,0,0).timestamp())
after = int(dt.datetime(2020,10,1,0,0).timestamp())

test_df = pd.DataFrame()
res = []

for sub in unique_subreddits:
    res.append(subreddit)
    print('Number of visited Subreddits is', len(res))
    submissions = api.search_submissions(subreddit=sub,
                  limit=None, before=before, after=after)

    submissions_df = pd.DataFrame(submissions)
    test_df = test_df.append(submissions_df)
```

Listing 1: Code snippet for pulling the submissions data

The resulting dataset was 5 GB of size csv file that contains 2 959 061 rows and 95 columns such. Based on this dataset we have identified the top 10 most engaging subreddits by deriving new feature, which we will talk about in details in the Section 3.1 and retrieved the Reddit comments on these subreddits for the last 6 months. The code snippet for getting the comments through Pushift API is also given in the Listing 2.

```
api = PushshiftAPI()

before = int(dt.datetime(2021,4,1,0,0).timestamp())
after = int(dt.datetime(2021,3,1,0,0).timestamp())

subreddit = 'AskReddit'

comments = api.search_comments(subreddit=subreddit, limit
      = None, before=before, after=after)
comments_df = pd.DataFrame(comments)
test_df = comments_df
```

Listing 2: Code snippet for pulling the comments data

The third dataset was a csv file 2.5 GB of size with 6 100 537 rows and 39 columns.

3 PREPROCESSING

In this section we are going to describe the steps taken to preprocess the data using Hadoop Streaming and then compare the performance of Hadoop Streaming with the pure Python implementation on the small 2.5 GB dataset. We are going to describe the Exploratory Data Analysis (EDA) conducted to identify the features for predictions and will elaborate on the Feature Generation and Cleaning using Hadoop Streaming to prepare the data for processing using Decision Tree and Random Forest algorithms.

3.1 EDA and Feature Selection

As the first step in our predictive analysis we have downloaded and loaded data to visualize the distribution and check summary statistics of feature variables. Since the Kaggle dataset was 15 GB of size and the posts dataset was 5 GB, loading the data to jupyter notebook at once was a huge challenge. As a solution to this we have used the pandas library's loading in chunks function. In the case of 2.5 GB we were able to easily load the dataset on local machine. The summary statistics of the datasets have showed that on 15 GB dataset out of 59,218,270 rows 4,713,988 and out of 22 columns 3 completely consisted of missing values. The 2.5 GB dataset in 10 out of 39 columns had only missing values. Hence, our first task of preprocessing was to clean the data from the missing values.

Next, we have converted the utc timestamp column of both datasets into separate *hour of the day, day of the week, day of the month, month of year, and year* columns and plotted the score of posts versus each of these columns to identify if there is any pattern. As can be observed from the Figure 3 the pattern was indeed present, and as one example of it is that content posted at night tends to get relatively higher score compared to day time posts. Hence we concluded that these new features can be used for future predictions.

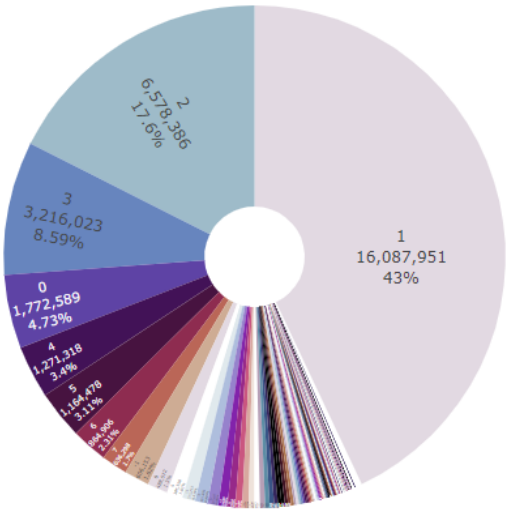

Figure 3: The Relationship between hour of day and Post Score

The next feature generated from our datasets was the engagement. As was mentioned before we have collected a 5 GB dataset of the posts, which contained the attributes like link of the post in the subreddit, number of comments under each post, and author of the posts. We have used these attributes to derive a new feature called the engagement, which was indicating how engaging the subreddit is. To calculate the engagement value of each subreddit we have calculated using MapReduce the number of links in each subreddit, number of comments under each links, along with the number of different authors participating in each subreddit.This has helped us to quantify the activities under each subreddit. This score then was added to the 2.5 GB dataset used for analyses. The code for these calculations is given in Listing 3.

```
import sys

link_comments = {}
subreddit_link = {}
subreddit_author = {}
new = {}

for line in sys.stdin:
    line = line.strip()
    line = line.split('\t')

    author = line[0]
    link = line[1]
    num_comments = line[2]
    subreddit = line[3]
    if link in link_comments:
        link_comments[link].append(int(num_comments))
    else:
        link_comments[link] = []
        link_comments[link].append(int(num_comments))

    if subreddit in subreddit_author:
        subreddit_author[subreddit].append(str(author))
    else:
        subreddit_author[subreddit] = []
        subreddit_author[subreddit].append(str(author))

    if subreddit in subreddit_link:
        subreddit_link[subreddit].append(str(link))
    else:
        subreddit_link[subreddit] = []
        subreddit_link[subreddit].append(str(link))

sum_comments = {}
for link in link_comments.keys():
    sum_comments[link] = []
    sum_comments[link].append(sum(link_comments[link]))

for key in subreddit_link.keys():
    for value in subreddit_link[key]:
        if key in new:
            new[key].append(int(str(sum_comments[value])
    [1:-1]))
        else:
            new[key] = []
            new[key].append(int(str(sum_comments[value])
    [1:-1]))

for subreddit in subreddit_author.keys():
    num_author = len(subreddit_author[subreddit])
    num_links = sum(new[subreddit])
    print ('%s,%s,%s,' % (subreddit, num_author,
    num_links))
```

Listing 3: Code snippet for Engagement

The last derived feature of the datasets was the sentiment of each comment posted under the subreddit posts. To get the sentiment of the comment we used the Natural Language Toolkit (NLTK) library, a ML based toolkit for human language data processing. NLTK is a python library for commonly user for NLP tasks, such as sentence segmentation, tokenization, and part-of-speech tagging. Then we passed the tokens to the VADER (Valence Aware Dictionary and sentiment Reasoner), which is a rule based sentiment analysis. We used VADER to classify the sentiment of users comments into 5 distinct classes - highly positive, moderately positive, neutral, moderately negative, highly negative.

All of these steps of preprocessing were tested on 2.5 GB dataset both locally in pure Python and using MapReduce. The 15 GB dataset was preprocessed only using Hadoop cluster.

3.2 Hadoop Streaming

To understand better the essence of Map Reduce jobs we have decide to go with low level Hadoop Streaming implementation of the MapReduce job for preprocessing. The code snippets of separate mapper and reducer can be observed correspondingly in the Listing 4 and 5.

```
def PreProcessingMapper(argv):
    sid = SentimentIntensityAnalyzer()

    my_iterator = iter(sys.stdin.readline, "")
    next(my_iterator)
    for row in csv.reader(iter(sys.stdin.readline, "")):
        if len(row) ==22:
            for i in range(0, 22):
                if isinstance(row[i], int):
                    continue
                elif len(row[i]) > 0:
                    continue
                else:
                    row[i]=0

            timestamp = row[0]
            utc = pd.to_datetime(timestamp, unit='s')
            hour = utc.hour
            dayofweek = utc.isoweekday()
            day = utc.day

            year = utc.year

            metrics = {}
            def remove_int(text):
                return ''.join([str(i) for i in text])
            if isinstance(row[17], int):
                sentiment_class = 10
            else:
                ss = sid.polarity_scores(remove_int(row
    [17]))
                for k in sorted(ss):
                    metrics[k] = ss[k]
                if(metrics['compound'] > 0.6):
                    sentiment_class = 1
                elif(metrics['compound'] > 0.25):
                    sentiment_class = 2
                elif(metrics['compound'] > -0.25):
                    sentiment_class = 3
                elif(metrics['compound'] > -0.6):
                    sentiment_class = 4
                else:
                    sentiment_class = 5
```

Listing 4: The Preprocessing Mapper

In the mapper function the line 4-5 read the first row of the csv file which is the column names and pass their values starting the preprocessing from the second row. The line 6-14 clean the rows and columns with the NaN values, by dropping the NaN

rows and replacing the NaN in row with 0, which is the mode for these columns. Next the line 16-22 convert the timestamp to the known format and extract the values of hour, day of week, day, month, and year. Through the line 24-42 the body of the comments are parsed and for each comment the polarity score is calculated using VADER tool. After identification of scores the sentiment class of each comment is assigned based on the threshold value. As a last step of preprocessing only highly correlated features are selected and printed by the preprocessing mapper function. This preprocessing step is mapper only job and all of the processing is done in mapper function.

```
#!/usr/bin/env python3
import sys

for line in sys.stdin:
    line = line.strip().split("\t")
    print ("%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%
     s\t%s\t%s\t%s\t%s" % (line[0], line[1], line[2],
      line[3], line[4], line[5], line[6],
          line[7], line[8], line[9], line[10], line[11],
     line[12], line[13], line[14], line[15], line[16]))
```

Listing 5: The Preprocessing Reducer

```
def Preprocessing(data):

    data = data.drop([columns], axis=1)
    data = data.dropna()

    data['created_loc_time'] = pd.to_datetime(data['
     created_utc'], unit='s')

    data['hour'] = [d.hour for d in data['
     created_loc_time']]
    data['dayofweek'] = [d.isoweekday() for d in data['
     created_loc_time']]
    data['day'] = [d.day for d in data['created_loc_time'
     ]]
    data['month'] = [d.month for d in data['
     created_loc_time']]
    data['year'] = [d.year for d in data['
     created_loc_time']]

    sid = SentimentIntensityAnalyzer()
    def sentiment_class_definition(comment):
        metrics = {}
        def remove_int(text):
            return ''.join([str(i) for i in text])
        ss = sid.polarity_scores(remove_int(comment))
        for k in sorted(ss):
            metrics[k] = ss[k]
            # Divides the Body into Sentiment Classes :
     {1: 'HP', 2: 'MP', 3: 'N', 4: 'MN', 5: 'HN'}
            if(metrics['compound'] > 0.6):
                sentiment_class = 1
            elif(metrics['compound'] > 0.25):
                sentiment_class = 2
            elif(metrics['compound'] > -0.25):
                sentiment_class = 3
            elif(metrics['compound'] > -0.6):
                sentiment_class = 4
            else:
                sentiment_class = 5
        return sentiment_class
    lis = []
    for index, row in data.iterrows():
        lis.append(sentiment_class_definition(row['body'
     ]))
        data['sentiment_class'] = pd.DataFrame(lis)
    data = data.drop('body', axis=1)
    return data
```

Listing 6: The Pure Python Code

As alternative to this all of these steps are implemented in the jupyter notebook using pandas dataframes. The code snippet for pure Python implementation is given in Listing 6.

The comparison of performance for both implementations is given in the table below.

Table 1: Performance Comparison for 2.5 GB data

	Hadoop Streaming	Pure Python
Loading	54s	146.76s
Preprocessing	9m 26s	6h 54m

.

3.2.1 Experimental Setup We ran the Hadoop Streaming preprocessing on the Hadoop Cluster with master node having 8 GB RAM and 4 CPUs, and 3 slave nodes each having 4 GB RAM and 2 CPUs. The total time for preprocessing the 15 GB data was 8 hours, and for 2.5 GB data was 9 minutes and 26 seconds. Execution of code in pure Python was done on a single machine with 8GB of RAM 7 cores and took 6 hours 54 minutes.

.

3.2.2 Analysis As it can be observed from the table, the loading and execution time on Hadoop cluster is significantly reduced in comparison with pure Python implementation. Even a simple loading operation took almost 3 times longer in pure Python, hence proving the point of time efficiency in deployment of clusters for computation.

4 BASELINE MODEL IMPLEMENTATION AND RESULTS

In this section we will talk about the implementation steps for baseline model construction and its results.

As the first step of our prediction task we have created a baseline model to use for comparison of the used models' performance. The code for the baseline model is given in the Listing 7. We first split the loaded spark dataframe into training and test sets. Next on the line 1 we create a basic model predicting the average value of training set's score values as the output and on the line 2 we obtain our predictions for test data. The lines 4-5 calculate the RMSE score of the predictions and print them on the line 6. After running this code we obtained the baseline value of 50 used to compare against the Decision Tree and Random Forest models' performance.

```
mean_score = float(trainingData.select('score').groupBy()
     .sum().first()[0]) / trainingData.count()
testData_withPreds = testData.withColumn('prediction',
     lit(mean_score))

def calcRMSE(data, labelCol='score'):
    return sqrt(float(data.rdd.map(lambda row: (row[
     labelCol]-row['prediction'])**2).reduce(lambda a,b:
     a+b)) / data.count())
print ('RMSE: %4.2f' % (calcRMSE(testData_withPreds,
     labelCol='score')))
```

Listing 7: Code snippet for Baseline Model

The table of performance summary for time and accuracy in case of both datasets is given below.

Table 2: Performance Comparison

	15GB Dataset	2.5GB Dataset
Time	12m 40s	1m 21s
Accuracy	50.4 RMSE	80.03 RMSE

In the following sections these numbers will be kept as reference for performance evaluation of implemented models.

5 DECISION TREE ALGORITHM AND IMPLEMENTATION

In this section we will describe the implementation steps of Decision Tree Algorithm using Spark MLlib library.

5.1 Spark

The general algorithm for implementing Decision Tree is displayed in the Figure 4. The same logic was used while implementing Decision Tree for Random Forest in MRJob, discussed later in Section 6.1.

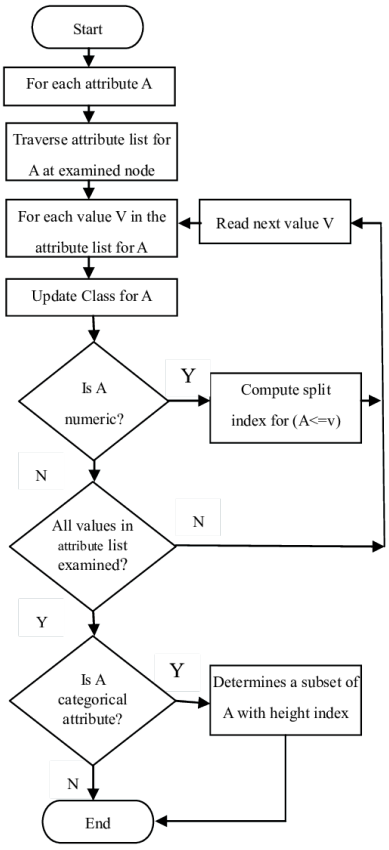

Figure 4: The Decision Tree Algorithm

To implement the decision tree algorithm in MLlib we have used decision tree library for regression. Before building the model we have prepared the training and test datasets by converting them to the LabeledPoint datatype, as required by the DecisionTreeRegressor() method. The codes of the data preparation step is given in the Listing 8. First we load the preprocessed data into the spark dataframe to apply the pipeline of transformations to the input data. The line 1-12 display the application of stringIndexer method for changing the categorical values of variables to numerical for easy interpretation by the model, since the model requires numerical inputs. For this we first define the columns with categorical variables (line 1) and after pass the values of those columns to StringIndexer method on the line 3-12. In the StringIndexer method the string values are encoded to indices sorted by their corresponding frequencies. The next step in data preparation is application of VectorAssembler on the line 14-19. This method transforms the data to a combination of list of feature columns in one single vector column. The last step in data preparation is transformation by VectorIndexer applied in the line 21-23 for automatic identification of categorical features and indexing them. After preparing the data we split it into train and test sets on line 24 and fit to the model on line 25-27. The tuning parameters of the Decision Tree are:

- maxBins - the number of bins which are used for discretization of continuous features [13].
- maxDepth - the maximum depth of the tree [13].
- impurity - homogeneity measure of the labels at the tree node. In case of classification this measure can be Gini impurity or entropy, whereas for regression the only measure is variance [13].

The predictions of the model are obtained on the line 28 and lastly the accuracy of predictions are obtained on the line 30-32. At the end the code prints the calculated RMSE score (line 34) and the summary of the tree constructed (line 37).

After getting the initial results we have added the lines given in Listing 9 to the code to tune the parameters. The parameters to tune were maxBins and maxDepth, since our task was regression, which has only variance as its impurity measure.

.

5.1.1 Experimental Setup To run the Decision Tree algorithm in Spark we have set up the cluster with driver node with 8 GB RAM and 4 CPUs, and 3 worker nodes each having 4 GB RAM and 2 CPUs. The total time for running the algorithm on the 15 GB data was 2 hours and 50 minutes, and for 2.5 GB data was 14 minutes and 11 seconds.

.

5.1.2 Analysis Before running the code on the Spark cluster we have tested the results on the small sample of the original dataset using the pyspark shell. After fixing the error and getting the working code we have obtained our initial results with the default parameters. Even though the RMSE score was lower compared to the baseline model it was still relatively high. Hence we have decided to tune the parameters with GridSearch for obtaining the optimal results. The total time to run the tuning of parameters for 15GB dataset was 10 hours 35 minutes, whereas for 2.5GB dataset was 3 hours 15 minutes. As a result, we were able to reduce the score

from 48.5 RMSE to 43.23 RMSE on 15GB dataset and from 80.4 to 78.3 RMSE on small dataset.

```
categoricalColumns = [item[0] for item in data.dtypes if
    item[1].startswith('string') ]

    stages = []
    for categoricalCol in categoricalColumns:

        stringIndexer = StringIndexer(inputCol =
    categoricalCol, outputCol = categoricalCol + 'Index'
    )
        stages += [stringIndexer]

    pipeline = Pipeline(stages = stages)
    pipelineModel = pipeline.fit(data)
    data = pipelineModel.transform(data)
    data = data.drop(*categoricalColumns)

    assembler = VectorAssembler( inputCols=['
    subredditIndex', 'sentimentIndex', 'HoDIndex', '
    DoWIndex', 'monthIndex'],
                                outputCol="features")

    output = assembler.transform(data)
    output = output.withColumn('label', output['score'])
    data = output.select("features", "label")

    featureIndexer =\
        VectorIndexer(inputCol="features", outputCol="
    indexedFeatures", maxCategories=25).fit(data)

    (trainingData, testData) = data.randomSplit([0.7,
    0.3])
    dt = DecisionTreeRegressor(featuresCol="
    indexedFeatures")
    pipeline = Pipeline(stages=[featureIndexer, dt])
    model = pipeline.fit(trainingData)
    predictions = model.transform(testData)

    evaluator = RegressionEvaluator(
        labelCol="label", predictionCol="prediction",
    metricName="rmse")
    rmse = evaluator.evaluate(predictions)

    print("Root Mean Squared Error (RMSE) on test data =
    %g" % rmse)

    treeModel = model.stages[1]
    print(treeModel)
```

Listing 8: Code snippet for DT in MLlib

```
dtparamGrid = (ParamGridBuilder().addGrid(dt.maxDepth,
    [5, 10, 15,  20, 25, 30]).addGrid(dt.maxBins, [25,
    35, 45, 55, 65]).build())

dtcv = CrossValidator(estimator = pipeline,
                      estimatorParamMaps = dtparamGrid,
                      evaluator = evaluator,
                      numFolds = 5)

dtcvModel = dtcv.fit(trainingData)
dtpredictions = dtcvModel.transform(testData)
```

Listing 9: The Tuning of DT

Table 3: Performance Comparison

5GB Dataset		
	Time	Accuracy
Without Tuning	2h 50m	48.8 RMSE
With Tuning	10h 35m	43.23 RMSE
2.5GB Dataset		
	Time	Accuracy
Without Tuning	14m 11s	80.4 RMSE
With Tuning	3h 15m	78.3 RMSE

The table above summarizes the performance for time and accuracy in case of both datasets.

6 RANDOM FOREST ALGORITHM AND IMPLEMENTATION

This section will give the overview of steps taken to implement Random Forest algorithm first using MRJob library of Python and then using MLlib library of Spark.

The codes in listings were based on the instructions in the course book [17].

6.1 MRJob

Considering that Random Forest algorithm can be implemented in parallel easily by distributing the training of trees across the nodes, we have decided to implement this algorithm both using MRJob library of Python and MLlib library of Spark.

As mentioned in the section 2 Random forest is an ensemble learning method which constructs multiple decision trees and outputs the mean of prediction of individual trees as its prediction. In our case the predicted score of the given post will correspond to the mean of the predictions of each constructed tree.

The overall structure of the implementation in MRJob consists of 2 steps. The first step in turn consists of 1 mapper and 1 reducer functions, whereas the second step consists of 1 mapper, 1 combiner, and 1 reducer functions. In the step 1 the training data is partitioned to train the trees on different subset of the original dataset. In step 2 the trees are trained and predictions are made. To implement the Random Forest algorithm we need to take the user defined options such as:

- maxSplitNumber- the maximum number of splits created from the input data.
- sampleNumber- the number of subsample for single split sampling without replacement.
- sampleSize- the number of records in subsample.
- testFile- path to the testfile.

To get the inputs from the command line we have defined configure options as shown in Listing 10.

Next we defined the steps as shown in Listing 11. Each of this steps will be described separately in the following part of this section.

```
def configure_options(self):
    super(MRPredictDelay, self).configure_options()
    self.add_passthrough_option('--maxSplitNumber', type=
     'int', default=10)
    self.add_passthrough_option('--sampleNumber', type='
     int', default=1000)
    self.add_passthrough_option('--sampleSize', type='int
     ', default=1000000)
    self.add_file_option('--testData')
```

Listing 10: Configuration Function

```
def steps(self):
    return [
        self.mr(mapper = self.step_1_mapper,
                reducer = self.step_1_reducer),
        self.mr(mapper_init = self.step_2_test_mapper,
                mapper = self.step_2_mapper,
                combiner = self.step_2_combiner,
                reducer = self.step_2_reducer)]
```

Listing 11: Steps

The Step 1 Mapper function in Listing 12 splits the input data into the maximum number of splits defined by option maxSplitNumber and generates the (key, value) pair, where key is a random number in the range of 0 and maxSplitNumber, and value is the parsed input line.

```
def step_1_mapper(self,_,line):
    key = random.randint(0, self.options.maxSplitNumber)
    line = line.replace('\t', '').split(',')
    yield key, line
```

Listing 12: Step 1: Mapper

The result of Step 1 Mapper function is passed to the Reducer function given in Listing 13. Here for each split we randomly collect without replacement a subsample using the sampleNumber option with the number records defined by sampleSize option. The output of this function is (key, value) pair, where key is the combination of the split number and the subsample number, and the value is the subsample itself.

```
def step_1_reducer(self, key, values):
    values_list = np.array(list(values))
    l = values_list.shape[0]
    if l < self.options.sampleSize:
        yield (key, list(values))
    else:
        for i in range(0, self.options.sampleNumber):
            idx = np.random.choice(l, size=self.options.
     sampleSize, replace=False)
            yield "{}_{}".format(key, i), values_list[idx
     , :].tolist()
```

Listing 13: Step 1: Reducer

After getting the samples of data we start building the trees in the second step of our implementation. The first function of Step 2 Mapper is given in the Listing 14. Since all the decision trees will be tested against same test data in this function we load the test data through the testData option and store it into the class attribute defined as self.testset. Next we run the Decision tree on the training data and make predictions. The mapper function yields a (key, value) pair with key 1 and value being the list of predictions.

```
def step_2_mapper(self, key, trainingData):
    DT = DecisionTree(training_data = trainingData,
     field_types = self.field_types)
    model = DT.build_model()
    predictions = DT.predict(model, self.test_set)
    yield 'predictions',(1,predictions)
```

Listing 14: Step 2: Mapper

In Step 2 Combiner function given in the Listing 15 we calculate the number of predictions and sum all the predictions received by this combiner to yield as (key, value) pair to reducer. This function is written to speed up the performance.

```
def step_2_combiner(self, key, predictions):
    predictions = list(predictions)
    combined_prediction = []
    predictions_number = len(predictions)
    for i in range(0, predictions_number):
        if i == 0:
            combined_prediction = predictions[i][1]
        else:
            combined_prediction = np.add(
     combined_prediction, predictions[i][1])
    yield key, (predictions_number, combined_prediction.
     tolist())
```

Listing 15: Step 2: Combiner

The last function of Step 2 given in listing 16 is Reducer which calculates the total number of predictions and their sum to yield the average value of prediction corresponding to each record in the test set.

```
def step_2_reducer(self, key, predictions):
    predictions = list(predictions)
    final_prediction = []
    predictions_number = 0
    for i in range(0, len(predictions)):
        predictions_number += predictions[i][0]
        if i == 0:
            final_prediction = predictions[i][1]
        else:
            final_prediction = np.add(final_prediction,
     predictions[i][1])

    final_prediction = np.divide(final_prediction,
     predictions_number)
    yield key, final_prediction.tolist()
```

Listing 16: Step 2: Reducer

.

6.1.1 Experimental Setup To run the Random Forest algorithm in Hadoop we have set up the cluster with master node with 8 GB RAM and 4 CPUs, and 3 slave nodes each having 4 GB RAM and 2 CPUs. The total time for running the algorithm on the 15 GB data was 14 hours 25 minutes, and for 2.5 GB data was 1 hour and 30 minutes.

Table 4: Performance Comparison: MRJob

	Time
5GB Dataset	14h 25m
2.5GB Dataset	1h 30m

6.2 Spark

The Spark implementation steps of Random Forest Algorithm were the same as steps for Decision Tree implementation. It was built using the MLlib library of Spark, which constructs the model on top of Decision Tree implementation. In this algorithm each tree is trained independently, in parallel, with subsamples of data and subsets of features. As in case of MRJob implementation the sampling and choice of features is done randomly. The data preparation for Random Forest also involved StringIndexer (line 1-10), VectorAssembler (line 12-13), and VectorIndexer (line 19-20). After preparing the data and splitting it into train and test sets (line 21) it was fit to the RandomForestRegressor() method. The predictions were obtained in line 26 and printed in line 32 together with model summary printed in line 35. Like in case of Decision Tree the parameters to tune in case of Random Forest were:

- numTrees - the number of trees [14].
- featureSubsetStrategy - the number of features for using while splitting [14].
- maxDepth - the maximum depth of the tree [14].
- impurity - homogeneity measure of the labels at the tree node. In case of classification this measure can be Gini impurity or entropy, whereas for regression the only measure is variance [14].

.

6.2.1 Experimental Setup and Results To run the Decision Tree algorithm in Spark we have set up the cluster with driver node with 8 GB RAM and 4 CPUs, and 3 worker nodes each having 4 GB RAM and 2 CPUs. The total time for running the algorithm on the 15 GB data was 4 hours 35minutes, and for 2.5 GB data was 14 minutes and 43 seconds.

.

6.2.2 Analysis Before running the code on the Spark cluster we have tested the results on the small sample of the original dataset using the pyspark shell. After fixing the error and getting the working code we have obtained our initial results with the default parameters. Even though the RMSE score was lower compared to the baseline model in case of large dataset it was still relatively high. In case of small dataset it was even slightly higher. Hence we have decided to tune the parameters with GridSearch like in case of Decision Tree for obtaining the optimal results. As a result, we were able to reduce the score for 15GB dataset from 48.5 RMSE to 42.10 RMSE and for small dataset from 81.3 RMSE to 78.5 RMSE. As a result, by tuning of parameters we were able to obtain performing better than a baseline model. The total amount of time for tuning in case of large dataset was 14hours minutes, for small it was 3 hours 58 minutes. As we can observe from the comparison of MRJob and and Spark implementations, even with tuning Spark implementation is faster for large dataset.

The Listing 17 displays the code snippet for Random Forest implementation in Spark.

The table below summarizes the performance for time and accuracy in case of both datasets.

Table 5: Performance Comparison: MLlib

5GB Dataset		
	Time	Accuracy
Without Tuning	4h 35m	48.5 RMSE
With Tuning	14h	42.10 RMSE
2.5GB Dataset		
	Time	Accuracy
Without Tuning	14m 43s	81.3 RMSE
With Tuning	3h 58m	78.5 RMSE

```
1  categoricalColumns = [item[0] for item in data.dtypes if
       item[1].startswith('string') ]
2      stages = []
3      for categoricalCol in categoricalColumns:
4          stringIndexer = StringIndexer(inputCol =
       categoricalCol, outputCol = categoricalCol + 'Index'
       )
5          stages += [stringIndexer]
6
7      pipeline = Pipeline(stages = stages)
8      pipelineModel = pipeline.fit(data)
9      data = pipelineModel.transform(data)
10     data = data.drop(*categoricalColumns)
11
12     assembler = VectorAssembler( inputCols=['
       subredditIndex', 'sentimentIndex', 'HoDIndex', '
       DoWIndex', 'monthIndex'],
13                                  outputCol="features")
14
15     output = assembler.transform(data)
16     output = output.withColumn('label', output['score'])
17     data = output.select("features", "label")
18
19     featureIndexer = VectorIndexer(inputCol="features",
       outputCol="indexedFeatures", maxCategories=25).fit(
       data)
20
21     (trainingData, testData) = data.randomSplit([0.7,
       0.3])
22
23     rf = RandomForestRegressor(featuresCol="
       indexedFeatures")
24     pipeline = Pipeline(stages=[featureIndexer, rf])
25     model = pipeline.fit(trainingData)
26     predictions = model.transform(testData)
27
28
29     evaluator = RegressionEvaluator(
30         labelCol="label", predictionCol="prediction",
       metricName="rmse")
31     rmse = evaluator.evaluate(predictions)
32     print("Root Mean Squared Error (RMSE) on test data =
       %g" % rmse)
33
34     treeModel = model.stages[1]
35     print(treeModel)
```

Listing 17: Code Snippe for RF in MLlib

7 RESULT AND CHALLENGES

In this section we are going to list the main challenges we have had during the project and will sum up the results of the project.

The main challenges during the implementation were long run time and memory errors due to the huge size of datasets. One of

such challenges was faced while running MRJob, when the "maximum recursion depth exceeded error" was raised. This issue was handled with manually setting recursion limit with setrecursionlimit() function [5]. Moreover, since the datasets were huge their visualization on local machine was not possible.

Nevertheless, with step by step analysis in chunks we were able to preprocess, process, and postprocess the data. The results we have obtained are summarized in the following plots.

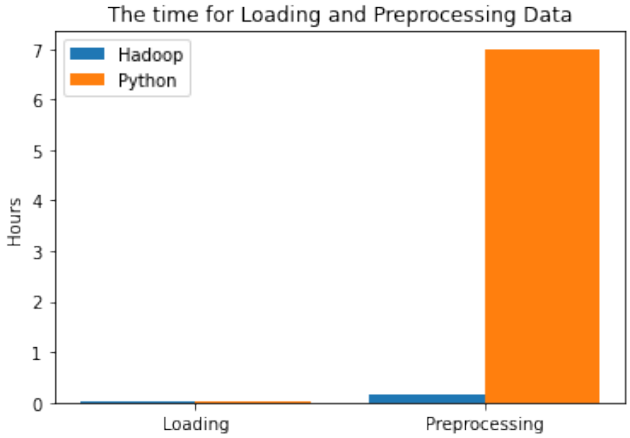

Figure 5

From the Figure 5 the gap between the loading and preprocessing in Python and Hadoop can be easily observed. Even as simple task as loading a small dataset was taking up to 2 minutes, which proves why Hadoop is more efficient with Big Data.

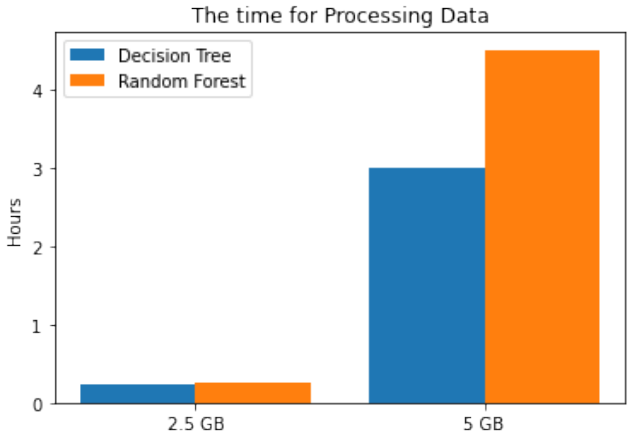

Figure 6

Figure 6 and 7 compare the performance of two Regression Algorithms. It is evident that Decision Tree is faster due to the complexity of Random Forest, which constructs multiple of such trees. It is also clear from second plot that Random Forest performs better due to the averaging of predictions from multiple trees, hence reducing the variance. However, in case of small dataset which is almost 30 times smaller it performs poorly, which is indicator of how accurate the predictions on Big Data can be.

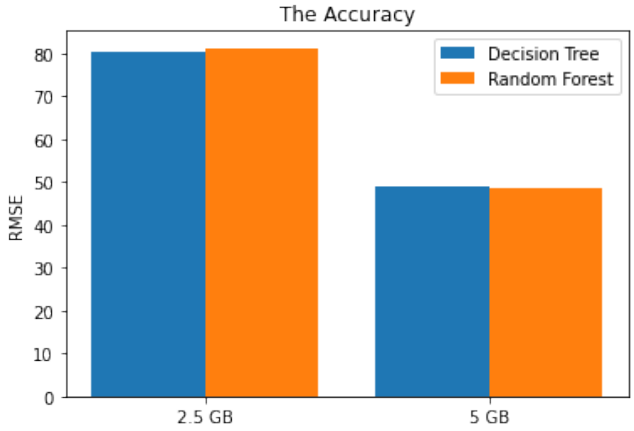

Figure 7

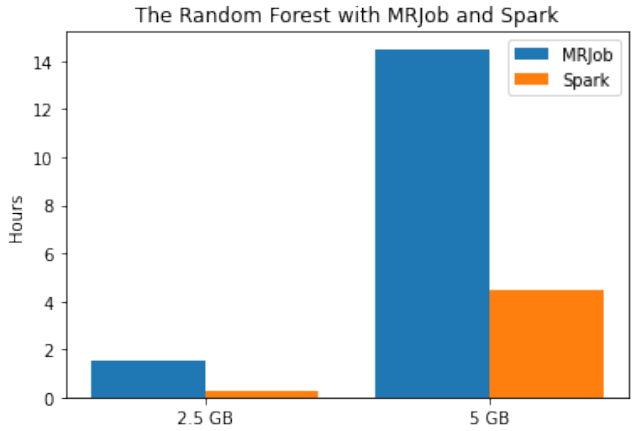

Figure 8

The last plot displays the comparison between MRJob and Spark implementations for Random Forest. The difference in execution time clearly proves the efficiency of in memory calculations of Spark opposite of heavy calculations with Hadoop. These statistics are the proof of declared 100 times fastness of Spark over Hadoop.

8 LIMITATIONS

This section enumerates the limitations faced during the project work.

The limitations of this project mainly result from time constraints imposed by the the deadlines and huge amount of time taken for the implementation steps due to the massive datasets.

First of the limitations is the Hadoop Streaming implementation of the preprocessing. Even through we have deliberately chosen to preprocess the data using Hadoop Streaming, due to the lack of

time we were not able to preprocess the data using MRJob library of Python.

Secondly, the data acquisition process took a lot of time, in total the whole first week have been spent collecting the new dataset from the API, and if to consider that the next weeks we had to study lots of new concepts while implementing the algorithms, we have lost the significant 7 days of our project. Moreover, the resulting dataset was only containing the records for last 6 months, which means with bigger dataset we could have obtained better predictions.

9 FURTHER WORK

In this section we are going to list the future work for our project.

As the amount of data has significant influence in the analysis, and our employed data was limited to 6 months, it is recommended to expand data by 2 years to enhance the accuracy and robustness of the results.

In this study, we have established our prediction model based on Decision Tree and Random Forest algorithms. For future work it is suggested to apply other machine learning approaches such as parallelized Nave Bayes, SVM and RNN architecture on the Reddit data in this context.

10 CONCLUSION

In this study, we have reviewed the implementation of Decision Tree and Random Forest algorithms on Reddit Big Data using Hadoop and Spark frameworks.

The implementation environment consists of 4-node Hadoop cluster with one master node (8GB RAM, 4 CPUs) and 3 slave nodes each having 4 GB RAM and 2 CPUs. Similar to this, we have set up 4-node Spark Cluster with driver node (8GB of RAM, 4 CPUs) and 3 workers having 4GB of RAM and 2 CPUs.

In this research two different Reddit datasets were used to implement the method as a case study. The first set was a csv file from Kaggle.com, which contained 59 million records in 22 features. Since these records expanded the time duration from May 2015 to 2017, we chose to create another dataset with more recent data of post and comments by using Pushshift API. The created dataset was 2.5 GB in size, which contained 39 features. The preprocessing phase was conducted by Hadoop MapReduce for larger dataset with the computation time of 9 minutes for small dataset and 8 hours for large dataset. In order to make better comparison among selected algorithms, the baseline prediction model was applied on train and test data.

In the next step, 2 regression algorithms of Decision Tree and Random Forest were implemented by utilizing MRjob python library and MLlib as Spark library. In the postprocessing step, the results of models were evaluated in term of model accuracy by using RMSE metric, and time complexity. The result shows that Random Forest is slightly more accurate model with 42.10 RMSE in comparison with Decision Tree with 43.23 RMSE for big dataset. In case of small dataset, due to the less amount of records with RMSE score 78.5 to 78.3 Decision Tree had slightly better performance. Even though the Random Forest took almost twice more time for building and predicting results, that time was still less than the implementation time with MRJob.

To conclude, even though we have extensively studied the structure and implementation in Apache Hadoop and Spark due to the time limitations we were not able to analyses the data expanding longer time duration and were restricted only to implementations with Decision Tree and Random Forest. These limitations are going to be addressed as future work of our project.

ACKNOWLEDGEMENT

We would like to thank our course instructors Mr Tomasz Wiktorski and Mr Jayachander Surbiryala for the materials we have learned throughout the project. It has been a very instructive and comprehensive course.

REFERENCES

[1] Alexa. 2021. The top 500 sites on the web. https://www.reddit.com/r/datasets/comments/ldozc6/how_to_create_large_datasets_from_reddit/
[2] Brad Anderson. 2019. Spark vs Hadoop frameworks: the main differences. https://medium.com/@bradanderson.contacts/spark-vs-hadoop-mapreduce-c3b998285578#:~:text=Hadoop%20indexes%20and%20tracks%20the,not%20provide%20their%20distributed%20storage.
[3] Alanna Tempest Andrei Terentiev. 2017. Predicting Reddit Post Popularity Via Initial Commentary. *IEEE Trans. Dependable Sec. Comput.* 14, 2 (2017), 172–184. https://doi.org/10.1109/TDSC.2015.2443781
[4] Molly Galetto. 2017. What is Predictive Marketing? The Rise of Predictive Marketing, Challenges, Benefits, and More. https://www.ngdata.com/what-is-predictive-marketing/
[5] GeeksforGeeks. 2019. Python | Handling recursion limit. https://www.geeksforgeeks.org/python-handling-recursion-limit/
[6] Alex Zamoshchin Jordan Segall. 2012. predicting reddit posts popularity. *Stanford University* (2012).
[7] Will Koehrsen. 2018. An Implementation and Explanation of the Random Forest in Python. https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76
[8] Dylan Mendonca. 2020. Predicting Popularity of Reddit Posts. (2020).
[9] Chenyao Yu Noah Makow, Matt Millican. 2017. A Network Model for Reddit Post Virality Prediction. *Stanford University* (2017).
[10] Radhakrishnam Moni Prasanna Chandramouli and journal = volume = number = pages = year = 2017 url = doi = timestamp = 2017 biburl = bibsource = Varun Elgano, title = predicting reddit posts popularity. [n. d.]. ([n. d.]).
[11] Reddit. 2021. How to Create Large Datasets from Reddit Submissions and Comments. https://www.reddit.com/r/datasets/comments/ldozc6/how_to_create_large_datasets_from_reddit/
[12] reddit. 2021. List Of Subreddits. https://www.reddit.com/r/ListOfSubreddits/wiki/listofsubreddits#wiki_ask_____
[13] Spark. 2021. Decision Tree. https://spark.apache.org/docs/2.2.0/mllib-decision-tree.html
[14] Spark. 2021. Decision Tree. https://spark.apache.org/docs/latest/mllib-ensembles.html#random-forests
[15] Jake Widman. 2021. What is Reddit? https://www.digitaltrends.com/web/what-is-reddit/
[16] Wikipedia. 2021. Reddit. https://en.wikipedia.org/wiki/Reddit
[17] Tomasz Wiktorski. 2019. *Data-intensive Systems Principles and Fundamentals using Hadoop and Spark*. Springer. https://doi.org/10.1007/978-3-030-04603-3
[18] David (Qifan) Zhang Yu Wu. 2011. Reddit Recommendation System Daniel Poon. *CS229,Stanford University* (2011).