Assignment 2 Big Data.
Stream processing with Spark.

Team: taigan
Members: Kazybek Askarbek
Yulia Chukanova
Shamil Khastiev
Alexandr Grichshenko
GitHub: https://github.com/QazyBi/Stream_Processing_Spark

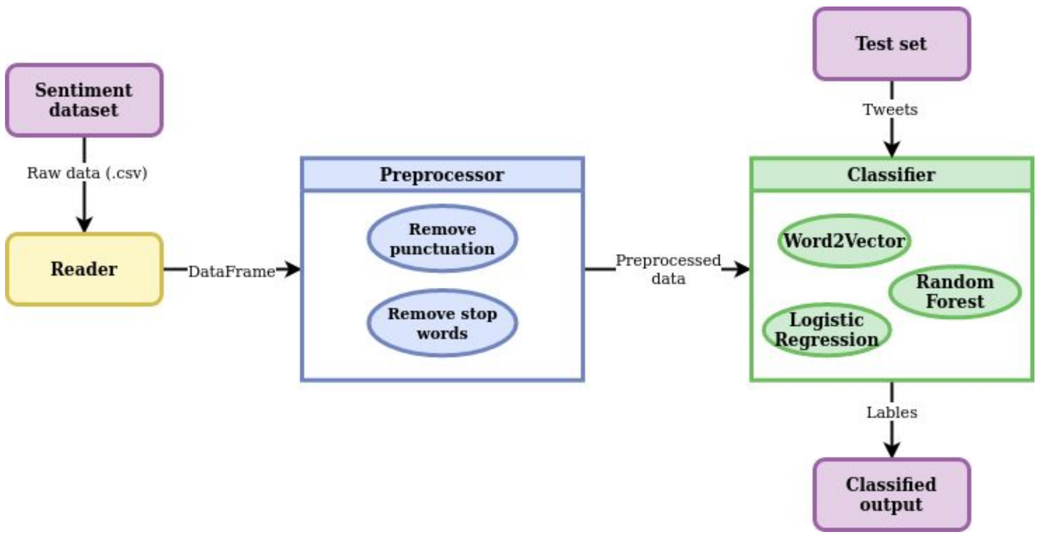
I. Introduction.

The goal of this assignment was to develop a machine learning application in Scala that will identify tweets by their sentiments (positive, neutral or negative). Moreover, it was required that the application could be utilized using Spark framework for stream processing running on Hadoop YARN resource manager.

The assignment, as the majority of machine learning tasks, was split into two stages: training phase and testing phase. We have used Stanford sentiment treebank [1] as our training set, due to the fact that it provides a large number of training examples, which are concise enough to be considered an adequate substitute for actual tweets. Also, by using this dataset it was possible to extract 3 labels as opposed to other datasets which are suited only for binary classification. We think that having a multi-label classifier enhances our system in terms and provides more useful and interpretable outputs.

The remaining part of the report is organized as follows: section II will review our design choices and describe the pipeline of our solution, section III will present and evaluate the results of the application.

II. Methodology.



The above image briefly describes the functionality of our application and in this section we will speak about each element in detail. Before proceeding to the design choices we state the contribution of team members.

Team member	**Contribution**
Kazybek Askarbek	Implementation of Random Forest and testing on the cluster
Yulia Chukanova	Implementation of dataset Reader, preprocessing steps and Word2Vector model
Shamil Khastiev	Implementation of Logistic Regression
Alexandr Grichshenko	Implementation of the driver code (Main.scala) and composition of the report

- **Reader**

The first stage in the pipeline is to load the dataset and present it to the next layers as a spark DataFrame. The source files containing sentiment data are *dictionary.txt* containing mapping of the phrase to its id and *sentiment_labels.txt* where each phrase id is associated with a class. The data in both documents is joined into a single DataFrame using phrase id as key.

- **Preprocessor**

In this stage we apply transformations to the raw data in order to eliminate redundancies that might hinder the performance of the ML models. We have chosen two sufficient transformations:

- removal of punctuation marks as they will interfere with vectorization of the word and do not carry any useful insight into sentiment (f.e. exclamation mark could both mean delight and outrage)
- removal of “stop words” such as pronouns and prepositions as by themselves they provide limited context into the tone of a sentence they appear in

We reckon that these transformations are sufficient as they prevent a classifier of dealing with obvious redundant information while still preserving the majority of original data.

- **Word2Vector**

As a method for numerical feature extraction we decided to use the Word2Vector model provided by Spark that encodes any given line of text into a unique vector based on the average occurrence of every word in the entire dataset. We opted for this particular method as its nature (similar words are situated near each other in a vector space) entails robustness and better generalization for the ML models. One parameter of this model that is worth mentioning is the vector size, which regulates how many words in the overall dictionary will be taken into account when building a vector. We have decided against using hyperparameter optimization in this case in order to divert more computational resources and time to optimizing ML models as parameters there tend to have a larger impact on performance. The agreed value for vector size was 50 as the statements in the training set and the target domain are rather short.

- **Logistic regression**

Since we are facing a classification task, Logistic Regression naturally comes as an initial choice for a model. We were interested to see how a relatively simple model will perform in comparison to a more sophisticated one (Random Forest) and whether we can draw any conclusion on the complexity of the task itself based on the results.

In the training of logistic regression three major hyperparameters were considered:
- *maxIter:* stopping condition on number of iterations
- *elasticNetParam:* parameter corresponding to L1 regularization (lasso)
- *regParam:* parameter corresponding to L2 regularization (ridge)

Out of these three we decided to perform grid search on only two of them: *maxIter* and *regParam.* The decision to exclude L1 regularization completely was made due to an aggressive nature of this type of regularization. When training logistic regression, especially with relatively few features we do not expect the weights to blow up to an extent that would warrant the application of L1. During the hyperparameter optimization we searched over 4 values for *regParam* (0, 0.01, 0.1). We wanted to check the corner case of applying no regression (0) and gradual increase up until 0.1. As for *maxIter* we considered values (50, 100). It is worth mentioning that we had to limit the number of values for hyperparameters due to the persistent out of memory exception on the cluster, the same is true regarding hyperparameters of the next model. The results of the testing will be provided in the later sections of the report.

- **Random forest**

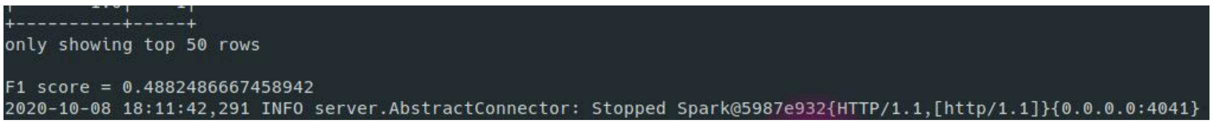
As a more advanced model to tackle this task we have chosen Random forest. There were two hyperparameters that we needed to search over:
- *numTrees:* amount of trees in the forest
- *maxDepth:* maximum depth of the trees

Each of the above parameters was given a value in the set (5, 10, 15, 20) as such values are commonly used in optimization of computationally non-intensive tasks.

III. Results.

- **Random forest**

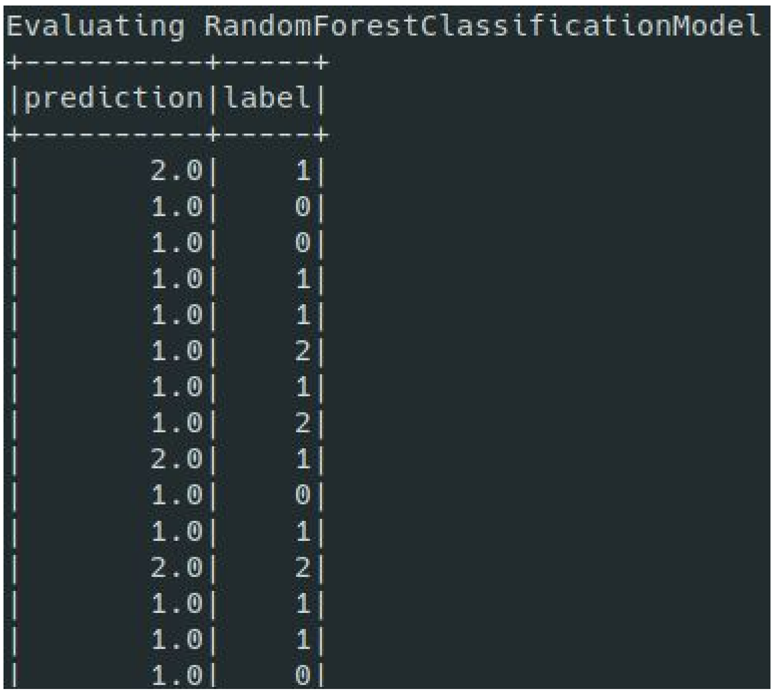
The screenshot below captures the F1 score of the model as evaluated on a test set that constituted 10% of the original dataset.



```
+----------+-----+
only showing top 50 rows

F1 score = 0.4882486667458942
2020-10-08 18:11:42,291 INFO server.AbstractConnector: Stopped Spark@5987e932{HTTP/1.1,[http/1.1]}{0.0.0.0:4041}
```

It is evident that the performance can hardly be considered satisfactory with such a score. One of the possible explanations for it could be the poor variety of hyperparameter choices inflicted by the exception described above. Another point, although not as quantifiable as the previous one, is the quality of the training data and the inherent subjectivity of labelling sentiment. In the screenshot below you may see a classification example using random forest:

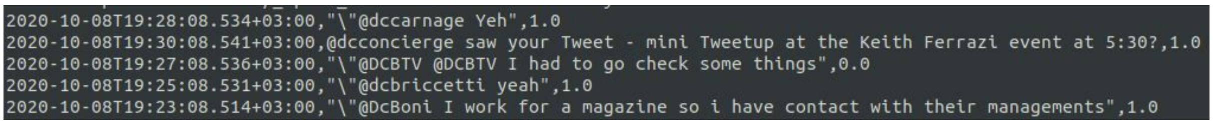


```
Evaluating RandomForestClassificationModel
+----------+-----+
|prediction|label|
+----------+-----+
|       2.0|    1|
|       1.0|    0|
|       1.0|    0|
|       1.0|    1|
|       1.0|    1|
|       1.0|    2|
|       1.0|    1|
|       1.0|    2|
|       2.0|    1|
|       1.0|    0|
|       1.0|    1|
|       2.0|    2|
|       1.0|    1|
|       1.0|    1|
|       1.0|    0|
```

- **Logistic regression**

The results of logistic regression are approximately similar to those of random forest. The F1 score was 0.4646224246890406, which is slightly worse than the results of random forest. We reckon that the reasoning for such performance is similar across both models.

Below you may see some examples of tweet classification:



```
2020-10-08T19:28:08.534+03:00,"\"@dccarnage Yeh",1.0
2020-10-08T19:30:08.541+03:00,@dcconcierge saw your Tweet - mini Tweetup at the Keith Ferrazi event at 5:30?,1.0
2020-10-08T19:27:08.536+03:00,"\"@DCBTV @DCBTV I had to go check some things",0.0
2020-10-08T19:25:08.531+03:00,"\"@dcbriccetti yeah",1.0
2020-10-08T19:23:08.514+03:00,"\"@DcBoni I work for a magazine so i have contact with their managements",1.0
```

Apparently, our classifier turned out to be in a foul mood, as it labeled the majority of the tweets as negative (1). Perhaps, the usage of the word "yeah/yeh" was perceived as passive-aggressive, and the last tweet as a hidden threat of a disgruntled customer.

IV. References.

[1] https://nlp.stanford.edu/sentiment/code.html