DI SS 2023
Task 2

Vladimir Panin (12238686) Besnik Cani (12243760)
Andrei Florescu (12242987)

May 2023

1 Introduction

This report focuses on the application of Spark for processing large text data, in our case the Amazon Reivew Dataset. In this assignment, the objective is to utilize Spark exploring different methods to analyze and manipulate text data. The assignment is divided into three parts.

In the first part, RDDs are employed to calculate chi-square values and extract the top terms per category. The results are then compared with the output generated in a previous assignment.

The second part involves leveraging Spark to convert the review texts into a vector space representation with TFIDF (Term Frequency-Inverse Document Frequency) weighted features. A transformation pipeline is built to prepare the data for further analysis.

In the third part, a text classification task is performed using the features extracted in the previous step. The pipeline is extended to use a Support Vector Machine (Support Vector Machine) classifier is trained on the data, and various parameter settings are explored using grid search. The performance of the trained classifiers is evaluated using F1 measure.

2 Problem Overview

This chapter provides an overview of the dataset used in the assignment and outlines the tasks to be accomplished in each subexercise. The Amazon Review Dataset from Assignment 1 is reused for this assignment. It comprises a collection of reviews on various products available on Amazon. The dataset contains text data that can be utilized for analysis and classification tasks. For the development the *reviews_devset.json* dataset can be used, however when running it on the cluster the reviewscombined.json dataset should be used. They can be accessed using the respective Hadoop Distributed File System (HDFS) path provided.

As stated in the report for Task 1, the dataset contains the following entries:

```
reviewerID - ID of the author of the review
asin - string - unique product identifier
reviewerName - string - name of the reviewer
helpful - array of two integers [a,b] - helpfulness rating of the review:
                                        a out of b customers found the review helpful
reviewText - string - the content of the review; this is the text to be processed
overall - float - rating given to product asin by reviewer reviewerID
summary - string - the title of the review
unixReviewTime - integer - timestamp of when review was created in UNIX format
reviewTime - string - date when review was created in human readable format
category - string - the category that the product belongs to
```

An examplary row is the following:

```
{"reviewerID": "A3D8QKK1Z19XMM", "asin": "B00004R9VV", "reviewerName": "Craig S Fry",
"helpful": [1, 1], "reviewText": "Good price. Good product. I had read
some poor reviews of competitive products before purchasing this one,
so I was a little concerned. But the Flowtron turned out to be exactly what I was looking for.",
"overall": 5.0, "summary": "Satisfied", "unixReviewTime": 1383696000,
"reviewTime": "11 6, 2013", "category": "Patio_Lawn_and_Garde"}
```

2.1 RDDs

In this part of the assignment, the goal is to apply RDDs (Resilient Distributed Datasets) and transformations to repeat the steps performed in Assignment 1. The specific tasks to be accomplished include:

- Tokenization
- Case folding
- Stopword filtering
- Calculation of chi-square values
- Outputting the sorted top terms per category
- Generating a joined dictionary

The output of these tasks should be written to a file named `output_rdd.txt`. The similarites and differences between`output_rdd.txt` and `output.txt` are to be discussed.

2.2 Datasets/DataFrames: Spark ML and Pipelines

Part 2 of the assignment focuses on utilizing Spark to convert the review texts into a classic vector space representation with TFIDF-weighted features. The main objective is to build a transformation pipeline that accomplishes the same set of steps as assignment 1 and then use the pipeline for further analysis and training. The tasks, which need to be carried out, need to be implemented using built-in functions and essentially be the same ones described previously in Part 1.

Furthermore, in the task description it was pointed out that results obtained from Task 1 and and Task 2 might differ.

2.3 Text Classification

The objective of Part 3 is to train a text classifier using the features extracted in Part 2. The classifier should be capable of predicting the product category based on the text of a review. The following tasks need to be completed:

- Extending the pipeline from Part 2 to incorporate a Support Vector Machine (SVM) classifier
- Implementing a strategy to address the multi-class problem by using binary classifiers
- Applying vector length normalization using the L2 norm
- Splitting the review data into training, validation, and test sets
- Ensuring reproducibility of experiments
- Performing a grid search for parameter optimization, including comparing different settings for chi-square filtering and SVM parameters
- Evaluating the performance of the trained classifiers on the test set using the F1 measure and the MulticlassClassificationEvaluator

To ensure cluster availability for all students, it is recommended to use the development set for building, optimizing, and

3 Methodology and Approach

3.1 RDDs

This solution performs chi-squared analysis on the dataset of amazon reviews that are located in a json file. The steps consist of:

- 1. Importing of libraries and initialization of a SparkSession and SparkContext for distributed processing.
- 2. Reading the dataset file and converting each line (JSON object) into a dictionary using json.loads().
- 3. Calculation the total counts of each token across all documents and storing the result in a dictionary, where the key is the token and the value is the total documents in which the token has appeared throughout all the corpus. This will then be used to calculate **B** in the chi-squared formula
- 4. Counting the tokens for each category, obtaining an RDD that groups tokens by category and calculates the total token occurrences in each category, which will be the value of **A** when we calculate the formula for chi-squared for the token.
- 5. Counting the number of documents for each category and storing the result in a dictionary, to make use of it when we need to calculate chi-squared. This is needed to calculate **C** and **D** for the chi-squared formula.
- 6. Counting of the total number of documents **(N)**, this is also needed for calculating **D**.
- 7. Defining of the function **calculateChiSquared(category)** that calculates the chi-squared value for each token in a given category. This method returns a string, consisting of the category and the top 75 tokens and their respective chi-squared values.
- 8. In the end, we iterate over each category from the variable generated from step four and call the **calculateChiSquared()** function, and store each of the lines into a list.
- 9. The list from step eight is sorted alphabetically and it prints the items to an output file and on the console.
- 10. Write all tokens obtained from step three into the output.

3.1.1 Differences from exercise 1 output

Contrary to my expectations, there is actually difference between the chi-squared values that are calculated but not so much difference in the order in which they appear. If we take for instance the category 'Apps For Android': In exercise 1, the output for the first 4 tokens was: *games:3081.14933748442926 play:2148.3501624308224 graphics:1505.5108977351497 kindle:1470.82...* In exercise 2, the output for the first 4 tokens is: *games:2562.6501245740806 graphics:1289.817 play:1220.3661920885065 kindle:1173.858784049012.* As stated above, we can see a difference in the chi-squared calculated values, where the values are smaller in exercise 2. However, the order in which the tokens appear is almost similar throughout the output, where in some categories it is identical, and for some there is swapped tokens in the order in which they appear.

3.2 Datasets/DataFrames: Spark ML and Pipelines

Data Loading and Preprocessing: In order to work with the reviews dataset we need to read the file's lines. After that we map the list of lines to json format and then we use pandas to convert the list of json objects to a Dataframe. After this step we need to transform the "category" field to a categorical variable in order to be used for further Machine Learning processing. We can achieve this by using the *pd.Categorical()*, which asigns numerical code to the values. We then save these numerical values to a new column in the dataframe called "labels". Concluding the data preprocessing step, we create the Spark Session and then we convert the pandas dataframe to a spark dataframe which uses only the columns "reviewText", "category" and "labels".

After finishing the preprocessing, the text preprocessing and feature extraction can be applied. A Tokenizer is applied to the "reviewText" column, splitting the text into individual words. Furthermore a StopWordsRemover is utilized to eliminate common stop words from the tokenized words. The next step in the Machine Learning pipeline, a HashingTF is employed to convert the tokenized words into numerical feature vectors. Additionally, an IDF (Inverse Document Frequency) is calculated to downscale the importance of frequent words. Lastly, a ChiSqSelector is utilized to select the top 2000 features based on chi-squared statistics and the "labels" column. These diffirent processing steps are put into a pipeline using *Pipeline* from *spark.ml*. The pipeline is applied to the dataframe using the *fit* function, the model is applied and finally written as a ".csv" file.

Transforming the Data: After the pipeline is fitted, it is used to transform the Spark DataFrame, generating a transformed output DataFrame.

Writing the Output to a CSV file: The final step involves writing the transformed output DataFrame to a CSV file named "test.csv". The DataFrame is first converted to a Pandas DataFrame using the toPandas() method, due to spark having issues with some characters when writing to a file, and then the Pandas DataFrame is written to the CSV file.

3.3 Text Classification

In Part 3, the code builds upon these preprocessing steps of Part 2 and incorporates machine learning classification. The pipeline is extended to include a machine learning classifier. In this case, the code employs the LinearSVC classifier and applies the "One-vs-Rest" strategy.

The code then splits the data into training, validation, and test sets using*randomSplit()*. The result of the text classificiation was a f1-score of !Input!

After splitting the data into training, validation, and test sets, a parameter grid is defined using the *ParamGridBuilder*. The parameter explored in our example are

- regParam: [0.1, 0.2, 0.3])
- standardization, [True, False])
- maxIter, [1, 2,3]

Using cross-validation and the f1-score as a metric the best model is selected through fitting the pipeline for every combination provided in the grid search. Finally, for the best model the best f1-score for comparison with the validation dataset are printed out.

For a configuration of *regPara*= 0.1, *standardization*= True and *maxIter*= 1 a f1-score of 0.4733 was obtained. Cosidering, there are 22 categories this is a fairly decent result. It is to note, that we were not able to run the grid search on the entire dataset, but when only using the hyperparameters provided above the mentioned f1-score was obtained.

The pipeline can be seen in Figure 1.

4 Conclusion

In conclusion, the implementation of chi-square calculation is easier using spark than using map reduce(mrjob). The spark pipeline was also faster than the MapReduce job and it also was more straightforward to implement because of multiple helper functions. The result was also a bit different between task 1 and task 2 especially the pipeline output. We could see that the pipeline has as output the words and the top 2000 selected features for each review, whereas in task 1 we had only one line of words and their associated chi-square values for each category.

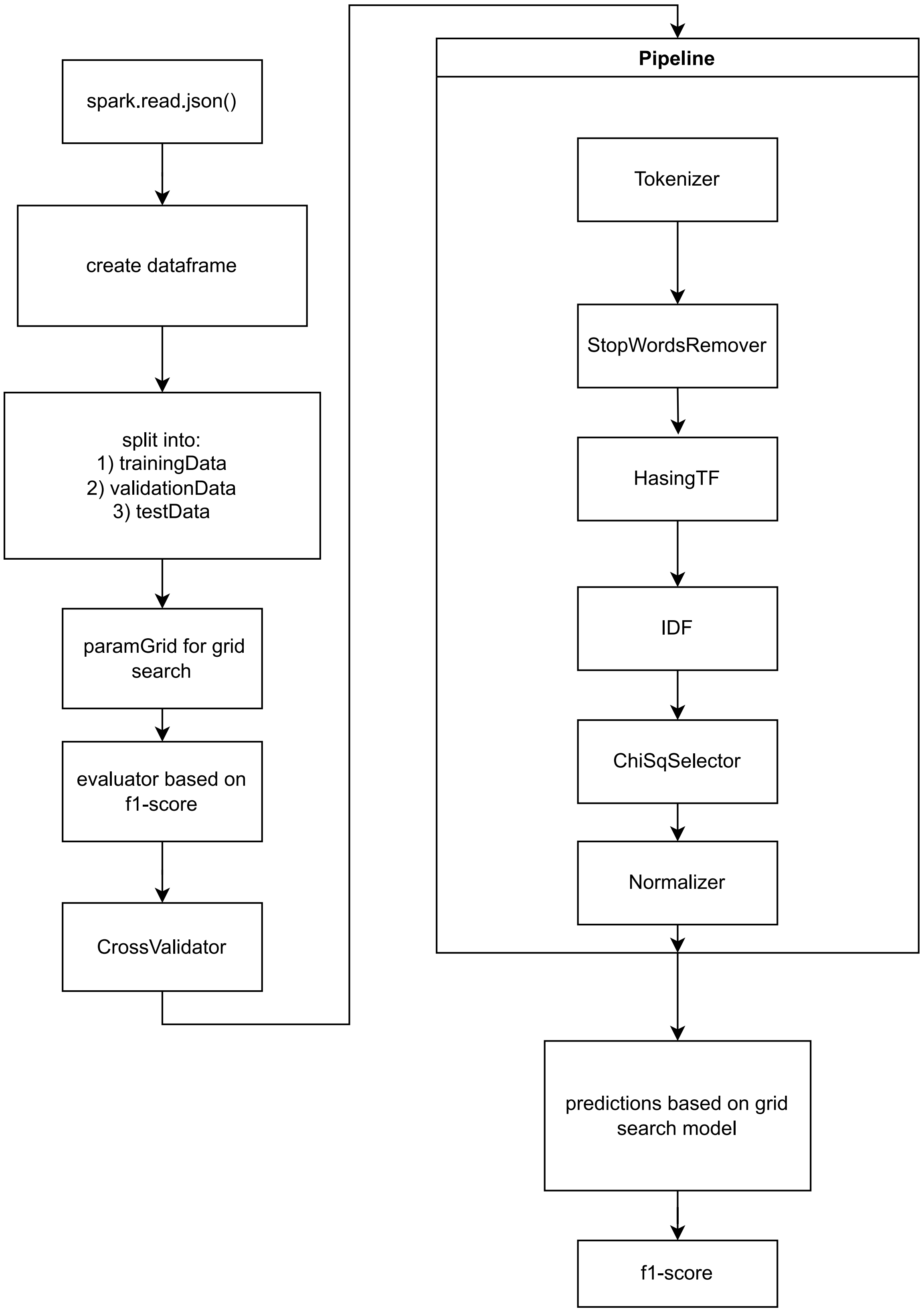

Figure 1: Pipeline

The text classification delivered as stated above a f1-score of 0.4733. It is to note that for the performance of the grid search it is important to not set the *maxIter* parameter too high, otherwise the grid search will take a much longer time to complete. Therefore, it can be stated that the implementation with the Spark pipeline and the text classification with the LinearSVC and the OneVsRest strategy can deliver decent results. Analyzing the review text in the manner proposed, can therefore be regarded as a good baseline model for assigning a product category to a review text. Further investigation with other Classifiers or adding further information like for example the review time could further enhance the f1-score obtained. Additional, exploration could be carried out in this area.