**LAB3: DATA ANALYTICS PIPELINE USING APACHE SPARK**

**Submitted BY:**

**Abhinav Kumar (Person No: 50247660)**

**Ravi Malik (Person No: 50247302)**

In this lab, we will explore processing graph data using Spark [1]. Here is a chance to show case your knowledge in Apache Spark data pipelines and big-data analytics. You can apply the model you build here, to numerous applications. Apache Spark is fantastically suited for this application we will describe here. And the application has uses in many vertical domains

**The Data**

Our task was to collect news articles for known categories in a directory. The four categories were Business, Sports, Politics and Comics. Given an unknown input dataset i.e. an article ,we want classify it into one of the categories. For this purpose we train our classifier on known articles (labels given) and test it on an unknown set of articles. The classifier makes the assumption that each new crime description is assigned to one and only one category. This is multi-class text classification problem.

* **Input**: Descript (Column name: comment\_text)

Example: “ PREMIERE LEAGUE”

* **Output**: Category (Column name: id)

Example: SPORTS

We combined all the articles from the four categories into one file named train4.csv and assigned labels to each article i.e category name. We used Pyspark to do the classification.

**Data Ingestion and Extraction**

We loaded the csv file with Spark csv packages using sqlContext

**BLOCK DIAGRAM OF DATA PIPELINE:**

Our pipeline includes three steps (**Feature Engineering**):

1. **regexTokenizer**: Tokenization (with Regular Expression)
2. **stopwordsRemover**: Remove Stop Words (Cleaning of data)
3. **countVectors**: Count vectors (“document-term vectors”)

**regexTokenizer**

A regex based tokenizer that extracts tokens either by using the provided regex pattern to split the text (default) or repeatedly matching the regex (if gaps is false).

**countVectors**

**In this step ,we form the document term vectors:  
It is a database of words that appear in a set of documents.The document-term matrix contains rows corresponding to the documents and columns corresponding to the terms. For instance if one has the following two (short) documents:**

**D1=”America calls soccer as football.”**

**D2=”Soccer is famous in America as football.”**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Soccer** | **is** | **famous** | **in** | **America** | **as** | **football** | **calls** |  |
| **D1** | **1** | **0** | **0** | **0** | **1** | **1** | **1** | **1** |  |
| **D2** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **0** |  |

Now, a column can also be understood as word vector for the corresponding word in the matrix

**StringIndexer**

StringIndexer encodes a string column of labels to a column of label indices. The indices are in [0, numLabels), ordered by label frequencies, so the most frequent label gets index 0. In our case, the label column (id) will be encoded to label indices, from 0 to 3; the most frequent label (LARCENY/THEFT) will be indexed as 0.

**Partioning Training and Test sets.**

The dataset is split into training and test data for classifying purpose.

**Model Training and Evaluation**

**Logistic Regression using Count Vector Features**

Our model will make predictions and score on the test set; we then look at the top 10 predictions from the highest probability.

We get an accuracy of : 98.17%

**Random Forest**

Our model will make predictions and score on the test set; we then look at the top 10 predictions from the highest probability.

We get an accuracy of : 94.52%

As we see the accuracy of test data in case of logistic regression is higher as compared to RandomForest classifier because RandomForest is not the best choice for high-dimensional sparse data.

**Unknown Set**

For an unknown set of articles compiled into a csv file name train6.csv, we get the following accuracies from the two classifiers:

Logistic Regression : 95.26%

RandomForest : 94.48%