MACHINE LEARNING USING APACHE SPARK

Udayasri Sai Buddhi
Graduate Student
School of Computing and Informatics
University of North Carolina at Charlotte
Email: ubuddhi@uncc.edu

***Abstract*—Clustering and Classification are the two major problems of machine learning. Clustering is an unsupervised learning process which explores the hidden or underlying structure of the given dataset. It has its significance in the fields of statistics, machine learning, text mining biology, medicine, psychology, anthropology and many applications of engineering. Classification is a different technique than clustering. Classification is similar to clustering in that it also segments customer records into distinct segments called classes. But unlike clustering, a classification analysis requires that the end-user/analyst know ahead of time how classes are defined, so it comes under Supervised learning. Classification is the task of choosing the correct class label for a given input. In basic classification tasks, each input is considered in isolation from all other inputs, and the set of labels is defined in advance. Some examples of classification tasks are: Deciding if an email is spam or not, deciding what the topic of a news article is, from a fixed list of topic areas such as sports, technology, and politics, deciding whether a given occurrence of the word bank is used to refer to a river bank, a financial institution, the act of tilting to the side, or the act of depositing something in a financial institution etc. One of the application of classification is WordPress PoolParty Plugin – It helps to make WordPress blogs and websites more understandable by linking posts with key terms and key terms with other key terms.**

To address these problems, we need a fast and rich machine learning library which is provided by Apache Spark. Apache Spark is a general-purpose cluster computing system and provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, ML for machine learning, GraphX for graph processing, and Spark Streaming. This report aims in performing clustering and classification using machine learning library of Apache Spark on Wikipedia Dataset. Wikipedia dataset has 11 major classifications. They Are- Culture, Geography, Health, History, Mathematics, Nature, People, Philosophy, Religion, Society and Technology. So the simple wiki dataset has articles and web pages with simple English words and stored in an XML format which helps in classifying the articles with great accuracy.

Keywords—component; formatting; style; styling; insert (key words)

I. INTRODUCTION

The ability of a computer to learn without being explicitly programmed is called Machine Learning [1]. There are 3 major machine learning tasks in Machine Learning. They are - Supervised learning, Unsupervised learning and reinforcement learning.

Supervised Learning: A machine learning task in which information is extracted from the labeled training data is called a supervised learning. In this kind of learning the computer will have prior information about the dataset.

Unsupervised Learning: A machine learning algorithm that is used to retrieve information from the dataset without having any prior knowledge about the dataset is called a unsupervised learning.

Reinforcement Learning: It is a branch of Artificial Intelligence that helps the machine to automatically understand the ideal behavior of a system within a specific context which helps in maximizing the performance of system.

A. Clustering

Clustering is defined as the grouping of objects into classes with the objective of maximizing intra-class similarity and minimizing inter-class similarity (Unsupervised Learning). The application of cluster analysis to textual documents is called Document clustering (or text clustering). Applications of document clustering are automatic document organization, topic extraction and fast information retrieval or filtering. They can be categorized to two types, online and offline. Online applications are usually constrained by efficiency problems when compared to offline applications [2].

There are two main common algorithms for document clustering – Hierarchical algorithm, K-means algorithm. In Hierarchical based algorithm, through aggregation or division, documents are clustered into hierarchical structure suitable for browsing. The main drawback of this algorithm is its efficiency. K-means is a partition based algorithm in which the entire document is partitioned into k clusters in which each document belongs to the cluster with nearest mean. Hierarchical algorithms produce more in-depth information for detailed analyses, while algorithms based around variants of the K-means algorithm are more efficient and provide sufficient information for most purposes.

In general, algorithms are divided into hard and soft clustering algorithms. In Hard Clustering, each document will be member of exactly one cluster. In Soft Clustering, a document may belong to more than one cluster. Therefore, in Soft Clustering, a document has fractional membership in several clusters. Dimensionality Reduction is considered to be sub category of Soft Clustering.

B. Classification

Classification is a supervised machine learning task which classifies the objects to one or more classes. Applications of Classification Problems are text categorization (e.g., spam filtering), fraud detection, optical character recognition, machine vision (e.g., face detection), natural-language processing (e.g., spoken language understanding), market segmentation (e.g.: predict if customer will respond to promotion), bioinformatics (e.g., classify proteins according to their function) [3].

Classification of a document into one or more categories is called Document Classification. Documents can be classified based on their subjects or any other attributes such as document type, author, printing year etc. There are two ways of Document Classification: the content-based approach and the request-based approach. Content-based classification is classification in which the weight of particular subjects in a document determines the class to which the document is assigned. Request-oriented classification is classification in which the documents will be classified based on the user's expectation [4].

II. MACHINE LEARNING LIBRARY (ML)

A. *Apache Spark*

Apache Spark was developed in UC Berkeley's AMPLab and later-on it moved to Apache. It is a powerful open source processing that provides parallel data processing framework and can work with Apache Hadoop. It combines batch, streaming, and interactive analytics on data that helps in building Big Data applications with great ease and speed [6].

Speed: Apache Spark uses advanced DAG execution engine that supports cyclic data flow and in-memory computing which makes large scale data processing 100 times faster than Hadoop.

Ease of Use: Spark has APIs that supports large datasets. This includes a collection of over 80 high-level operators that make it easy to build parallel apps and can be used interactively from the Scala, Python and R shells.

A Unified Engine: Spark comes packaged with higher-level libraries, including support for SQL queries, streaming data, machine learning and graph processing. These standard libraries increase developer productivity and can be seamlessly combined to create complex workflows [7].

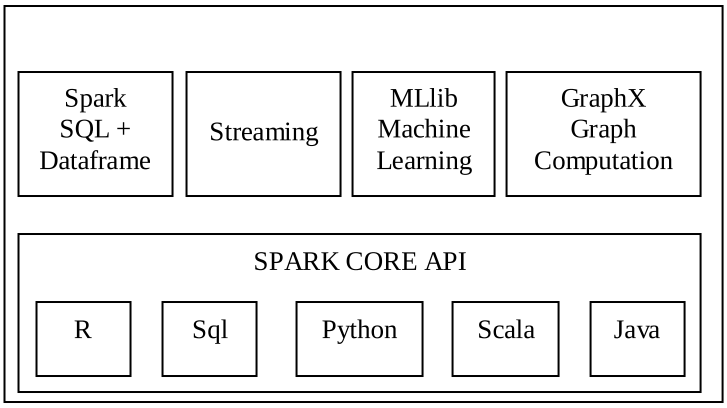


Fig. 1: Apache Spark Ecosystem [6]

B. *ML Library in Apache Spark*

ML library of Apache Spark supports nearly all Machine Learning Algorithms. It is easier to use and has the ability to save and load models. In general, Machine Learning consists of many stages such as feature extraction, transformation to model fitting and tuning. ML provides a pipeline to gives the user ease of understanding the workflow [7].

C. *Clustering*

The implementation in ML has the following parameters:

- k is the number of clusters.
- *maxIterations* is the maximum number of iterations to run.
- *initializationMode* specifies either random initialization or initialization via k-means||.
- *runs* is the number of times to run the k-means algorithm.
- *initializationSteps* determines the number of steps in the k-means algorithm.
- *epsilon* determines the distance threshold within which we consider k-means to have converged [8].

D. *Algorithm//K-Means Algorithm//[9]*

```
Input: E = {$e_1$, $e_2$…., $e_n$} (set of entities to be clustered)
      K (number of clusters)
      MaxIters (limit of iterations)
Output: C = {$c_1$, $c_2$……$c_n$} (set of cluster centroids)
      L = {l(e)\c=1, 2…., n} (set of cluster labels of E}
for each $c_i$ ϵ C do
    $c_i$ ← $e_j$ ϵ E (e.g. random selection)
end
for each $e_i$ ϵ E do
    l ($e_i$ ← argminDistance ($e_i$, $c_j$) j ϵ {1……. k}
end
changed ← false;
iter←0;
repeat
for each ci ϵ C do
      UpdateCluster(ci);
end
for each ei ϵ E do
    minDist←argminDistance($e_i$, cj) j ϵ {1……. k};
    if minDist ± l(ei); then
          l(ei) ←minDist;
          changed ← true;
    end
end
```

E. *Classification*

ML supports various methods for binary classification such as linear SVMs, logistic regression, decision trees, naive Bayes, decision trees, naive Bayes and regression analysis. [10]

F. Naive Bayes

Naive Bayes is a simple multiclass classification algorithm with the assumption of independence between every pair of features. Naive Bayes can be trained very efficiently. Within a single pass to the training data, it computes the conditional probability distribution of each feature given label, and then it applies Bayes' theorem to compute the conditional probability distribution of label given an observation and use it for prediction.

ML supports multinomial naive Bayes, which is typically used for document classification. Within that context, each observation is a document and each feature represents a term whose value is the frequency of the term. Feature values must be nonnegative to represent term frequencies. Additive smoothing can be used by setting the parameter λ. For document classification, the input feature vectors are usually sparse, and sparse vectors should be supplied as input to take advantage of sparsity. Since the training data is only used once, it is not necessary to cache it [11]

III. SYSTEM REQUIREMENTS

A. Hardware

ENVIRONMENT: Google Cloud

RAM: 17GB

SPARK MEM STORE: 5.4GB

JAVA HEAP SPACE: 10GB

HDD- 100GB SSD

B. Software Stack

OPERATING SYSTEM: Cent OS 7

TOOLS & TECHNOLOGIES:	Apache Spark 2.0.2
	Scala 2.12
	Maven 3.3.9
	Java 1.8
	Scala IDE

C. Dataset

The given dataset is a huge file of size 528MB in XML format. It is difficult to process such a huge file. Each XML file has the following structure:

```
<mediawiki >
    <page>
        <title> </title>
        <ns>    </ns>
        <id>    </id>
        <revision>
         <id>   </id>
         <parentid>        </parentid>
         <timestamp>     </timestamp>
             <contributor>
                  <username>        </username>
                   <id>      </id>
             </contributor>
                  <comment>  </comment>
                  <model>       </model>
                  <format>      </format>
                  <text >       </text>
                 <sha1>         </sha1>
            </revision>
    </page>
</mediawiki>
```

Each page attribute has revision tag in which we have text tag where the information is stored and a title tag. And thus 24077 text documents have been identified and extracted.

IV. CLUSTERING

Pre-processing phase

The preprocessing phase converts the original textual data in a data mining-ready structure, where the most significant text-features that serve to differentiate between text-categories are identified. It is the process of incorporating a new document into an information retrieval system. This phase is the most critical and complex process that leads to the representation of each document by a select set of index terms.

The preprocessing involves various steps.

Tokenization:

Tokenization in text mining is the process of converting a sentence into words, phrases, symbols, or other meaningful elements called tokens which are grouped together as a semantic unit and used as input for machine learning.

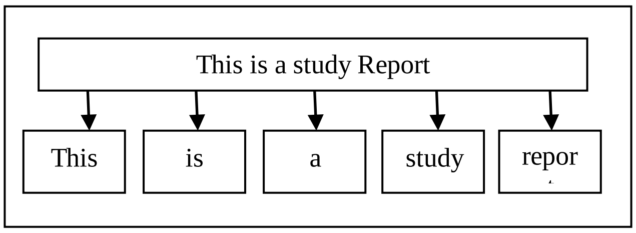


Fig. 2: Tokenization

Stop word Removal:

Words such as: a, an, the, and, are, as, at, be, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with etc. will not carry any information and has very little significance but accounts for most frequent occurrence which can be excluded. These words are called stop words and the process of removing these words is called stop word removal.

Term-Frequency Matrix

Term-Frequency matrix is the matrix which converts the words into a matrix. It collects all the words from all the documents which forms the features for the dataset. In our dataset, we have about more than 2 lakh words. Term-Frequency is a matrix with a size of number of documents as the number of rows, number of features as the number of columns. It gives the importance of a term to a document as vectors.

Inverse-Document Frequency

Inverse-Document Frequency gives the information about how important a word is on the whole document. If a term is present in all the documents, the IDF will be a zero. If a word appears in all the documents that reflects that it does not carry much information. Hence it tends to zero. The specificity of a term can be quantified as an inverse function of the number of documents in which it occurs.

Normalize

Normalizer is a process after Inverse - Document Frequency in which we normalize each vector that is generated for each document. In other words, Normalizing is dividing each vector element with the maximum value of the vector so that the resulting value ranges from 0 to 1.

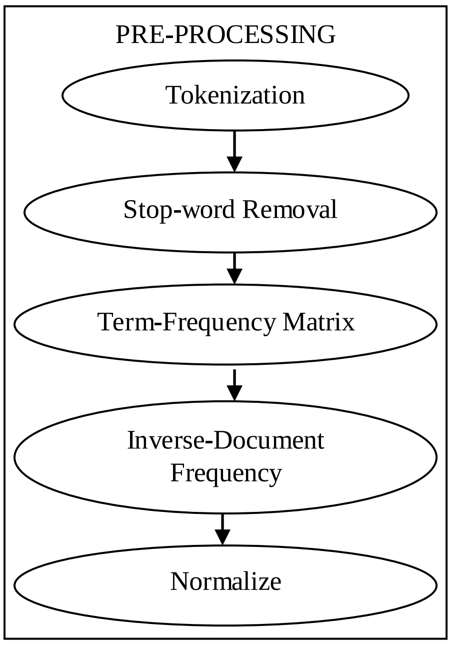
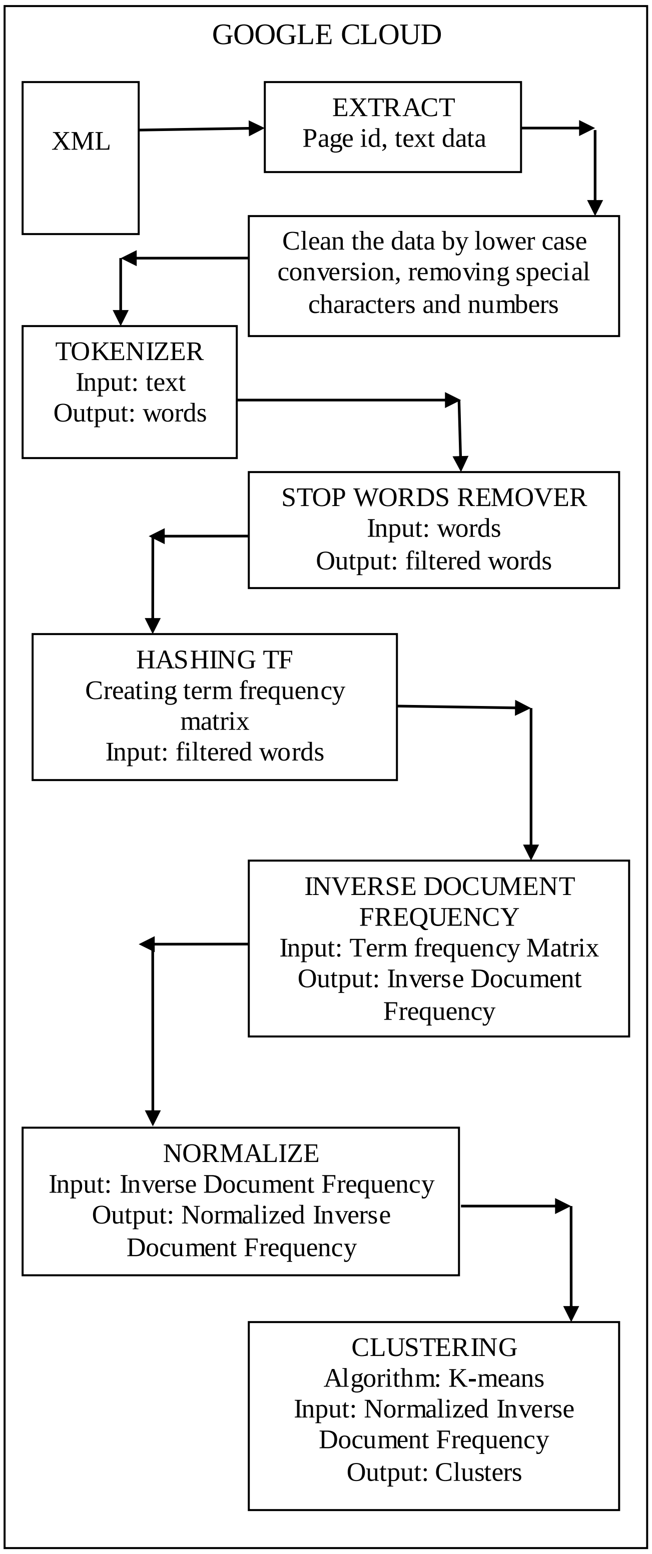


Fig. 3: Pre-Processing Workflow

K-Means

After Normalizing each document vector applying K-means will form clusters of all the documents. It takes k as the input which decides the number of clusters. The output of K-means will give you the cluster number of each document. If you group, the documents based on their cluster number you will get the documents of same topic in each cluster.

A. IMPLEMENTATION

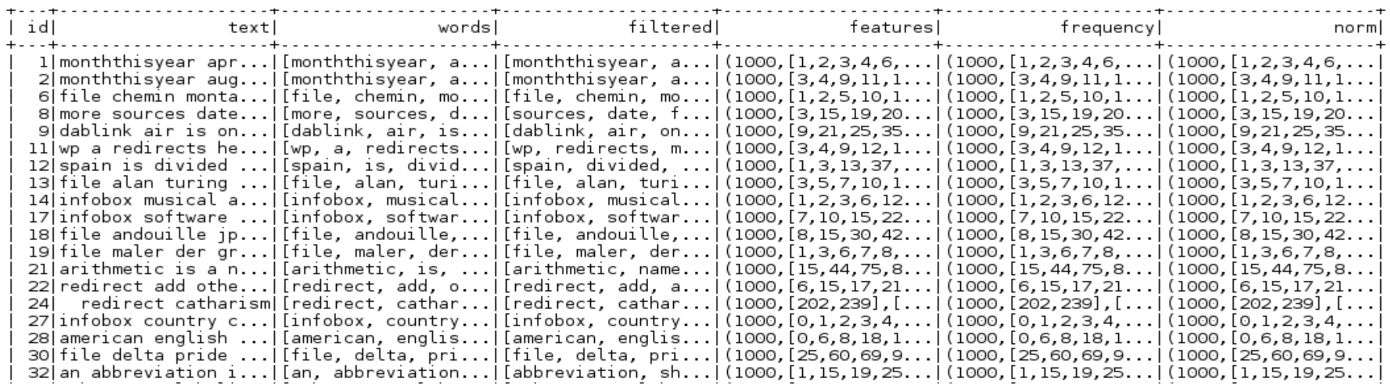


GOOGLE CLOUD
XML
EXTRACT
Page id, text data
Clean the data by lower case conversion, removing special characters and numbers
TOKENIZER
Input: text
Output: words
STOP WORDS REMOVER
Input: words
Output: filtered words
HASHING TF
Creating term frequency matrix
Input: filtered words
INVERSE DOCUMENT FREQUENCY
Input: Term frequency Matrix
Output: Inverse Document Frequency
NORMALIZE
Input: Inverse Document Frequency
Output: Normalized Inverse Document Frequency
CLUSTERING
Algorithm: K-means
Input: Normalized Inverse Document Frequency
Output: Clusters

B. *Clustering Scala Code:[12]*

```
   package com.sparkscala
   import java.io.PrintWriter
   import java.io.File
   import org.apache.spark.SparkConf
   import org.apache.spark.SparkContext
   import org.apache.spark.sql.SQLContext
   import org.apache.spark.rdd.RDD
   import org.apache.spark.ml.feature.IDF
   import org.apache.spark.ml.feature.StopWordsRemover
   import org.apache.spark.ml.feature.Tokenizer
   import org.apache.spark.ml.feature.HashingTF
   import org.apache.spark.sql.Row
   import org.apache.spark.ml.clustering.KMeans
   import org.apache.spark.ml.Pipeline
   import org.apache.spark.ml.feature.VectorAssembler
   import org.apache.spark.ML.linalg.Vector
   import org.apache.spark.ML.util.MLUtils

   object DocumentTF {

     case class Article(id: Integer, text: String)

     def main(args: Array[String]) {
   //Loading the xml file
       val xml = scala.xml.XML.loadFile(args(0))
   //Reading the data from the file
       val pw = new PrintWriter(new File(args(1)))
       val mediawiki = (xml \\ "mediawiki" \\ "page").foreach {
         page =>
           var result = (page \ "id").text + "," + (page \ "revision" \
"text").text.toLowerCase().trim().replaceAll("[^A-Za-z ]+", "
").trim().replaceAll(" +", " ").toString()
           pw.println(result)
       }
       pw.close()

       val conf = new
SparkConf().setAppName("SparkKMeans").setMaster("local")
       val sc = new SparkContext(conf)
       val sqlContext = new SQLContext(sc)

       import sqlContext.implicits._
       import sqlContext._

       def parseArticle(str: String): Article = {
         val line = str.split(",", -1)
         Article(line(0).toInt, line(1))
       }

       val textRDD = sc.textFile(args(1))
       val articleRDD = textRDD.map(parseArticle).cache()
       val dataset = articleRDD.toDF()

       val tokenizer = new
Tokenizer().setInputCol("text").setOutputCol("words")
       val wordsData = tokenizer.transform(dataset)

       val remover = new StopWordsRemover()
         .setInputCol(tokenizer.getOutputCol)
         .setOutputCol("filtered")

       val filtereddata = remover.transform(wordsData)

       val hashingTF = new
HashingTF().setNumFeatures(1000).setInputCol(remover.getOutput
Col).setOutputCol("features")

       val featurizedData = hashingTF.transform(filtereddata)

       val inverseDocumentFreq = new
IDF().setInputCol(hashingTF.getOutputCol).setOutputCol("frequenc
y")

       val idfOutput =
inverseDocumentFreq.fit(featurizedData).transform(featurizedData
       import org.apache.spark.ml.feature.Normalizer
       val normalizer = new
Normalizer().setInputCol(inverseDocumentFreq.getOutputCol).setOu
tputCol("norm")
       val normOutput = normalizer.transform(idfOutput)
       val km = new
KMeans().setK(4).setSeed(43L).setFeaturesCol(normalizer.getOutput
Col).setPredictionCol("prediction")
       val pipeline = new Pipeline().setStages(Array(tokenizer,
remover, hashingTF, inverseDocumentFreq, normalizer, km))
       val pipelineModel = pipeline.fit(dataset)
       val pipelineResult = pipelineModel.transform(dataset)
       val finalResult = pipelineResult.select("id", "prediction")
finalResult.repartition(1).write.format("com.databricks.spark.csv").o
ption("header", true).save("prediction.csv")
     sc.parallelize(pipelineResult.repartition(1).collect().toSeq).saveA
sTextFile("prediction.txt")
     }

}
```

C. *OUTPUT BEFORE CLUSTERING*



```
+---+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
| id|                text|               words|            filtered|            features|           frequency|                norm|
+---+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|  1|monththisyear apr...|[monththisyear, a...|[monththisyear, a...|(1000,[1,2,3,4,6,...|(1000,[1,2,3,4,6,...|(1000,[1,2,3,4,6,...|
|  2|monththisyear aug...|[monththisyear, a...|[monththisyear, a...|(1000,[3,4,9,11,1...|(1000,[3,4,9,11,1...|(1000,[3,4,9,11,1...|
|  6|file chemin monta...|[file, chemin, mo...|[file, chemin, mo...|(1000,[1,2,5,10,1...|(1000,[1,2,5,10,1...|(1000,[1,2,5,10,1...|
|  8|more sources date...|[more, sources, d...|[sources, date, f...|(1000,[3,15,19,20...|(1000,[3,15,19,20...|(1000,[3,15,19,20...|
|  9|dablink air is on...|[dablink, air, is...|[dablink, air, on...|(1000,[9,21,25,35...|(1000,[9,21,25,35...|(1000,[9,21,25,35...|
| 11|wp a redirects he...|[wp, a, redirects...|[wp, redirects, m...|(1000,[3,4,9,12,1...|(1000,[3,4,9,12,1...|(1000,[3,4,9,12,1...|
| 12|spain is divided ...|[spain, is, divid...|[spain, divided, ...|(1000,[1,3,13,37,...|(1000,[1,3,13,37,...|(1000,[1,3,13,37,...|
| 13|file alan turing ...|[file, alan, turi...|[file, alan, turi...|(1000,[3,5,7,10,1...|(1000,[3,5,7,10,1...|(1000,[3,5,7,10,1...|
| 14|infobox musical a...|[infobox, musical...|[infobox, musical...|(1000,[1,2,3,6,12...|(1000,[1,2,3,6,12...|(1000,[1,2,3,6,12...|
| 17|infobox software ...|[infobox, softwar...|[infobox, softwar...|(1000,[7,10,15,22...|(1000,[7,10,15,22...|(1000,[7,10,15,22...|
| 18|file andouille jp...|[file, andouille,...|[file, andouille,...|(1000,[8,15,30,42...|(1000,[8,15,30,42...|(1000,[8,15,30,42...|
| 19|file maler der gr...|[file, maler, der...|[file, maler, der...|(1000,[1,3,6,7,8,...|(1000,[1,3,6,7,8,...|(1000,[1,3,6,7,8,...|
| 21|arithmetic is a n...|[arithmetic, is, ...|[arithmetic, name...|(1000,[15,44,75,8...|(1000,[15,44,75,8...|(1000,[15,44,75,8...|
| 22|redirect add othe...|[redirect, add, o...|[redirect, add, a...|(1000,[6,15,17,21...|(1000,[6,15,17,21...|(1000,[6,15,17,21...|
| 24|  redirect catharism|[redirect, cathar...|[redirect, cathar...|(1000,[202,239],[...|(1000,[202,239],[...|(1000,[202,239],[...|
| 27|infobox country c...|[infobox, country...|[infobox, country...|(1000,[0,1,2,3,4,...|(1000,[0,1,2,3,4,...|(1000,[0,1,2,3,4,...|
| 28|american english ...|[american, englis...|[american, englis...|(1000,[0,6,8,18,1...|(1000,[0,6,8,18,1...|(1000,[0,6,8,18,1...|
| 30|file delta pride ...|[file, delta, pri...|[file, delta, pri...|(1000,[25,60,69,9...|(1000,[25,60,69,9...|(1000,[25,60,69,9...|
| 32|an abbreviation i...|[an, abbreviation...|[abbreviation, sh...|(1000,[1,15,19,25...|(1000,[1,15,19,25...|(1000,[1,15,19,25...|
```

D. PART OF THE OUTPUT AFTER CLUSTERING:

Doc_Id	Cluster
22	2
24	1
27	0
28	0
30	2
32	2
33	0
35	0
37	3

V. CLASSIFICATION

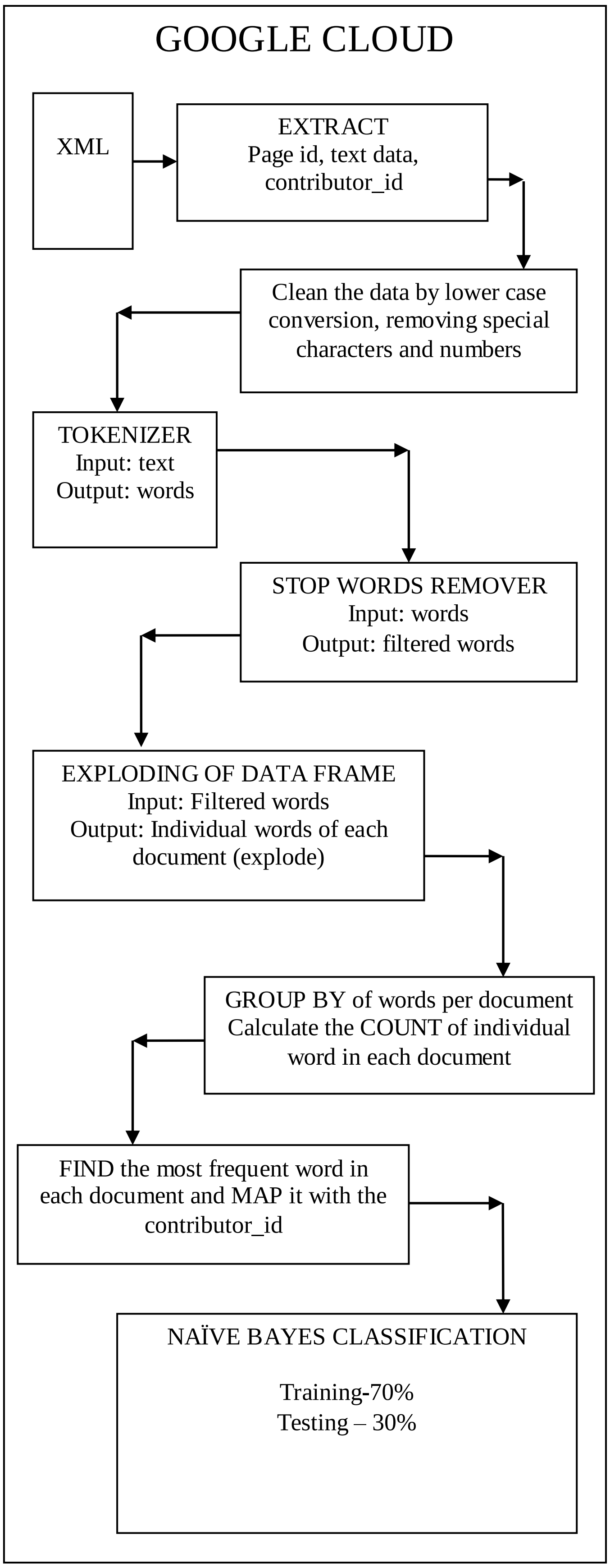
The Naive Bayes Classification is based on Bayesian theorem. It is suited if the dimensionality of the inputs is high. It is simple in implementation. It is one of the most common classifiers available [13].

In this paper, we built Naïve Bayes classification based on the text data of the articles and contributor_id of each article. This output will be predicting the contributor based on the most frequently used word in the article.

We extract the data in the similar manner as of clustering, from XML document and then perform tokenization, stop words remover. The output from the stop words remover is exploded to extract each individual word. The count of each word in each document is calculated by using GROUP BY clause.

From each document the most frequent word is identified and mapped with the contributor_id. This will be sent as the input to the Naïve Bayes classification.

A. IMPLEMENTATION


GOOGLE CLOUD
XML
EXTRACT Page id, text data, contributor_id
Clean the data by lower case conversion, removing special characters and numbers
TOKENIZER Input: text Output: words
STOP WORDS REMOVER Input: words Output: filtered words
EXPLODING OF DATA FRAME Input: Filtered words Output: Individual words of each document (explode)
GROUP BY of words per document Calculate the COUNT of individual word in each document
FIND the most frequent word in each document and MAP it with the contributor_id
NAÏVE BAYES CLASSIFICATION
Training-70%
Testing – 30%

B. Classification Scala Code:

```
package com.sparkscala

import java.io.PrintWriter
import java.io.File
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.sql.SQLContext
import org.apache.spark.rdd.RDD
import org.apache.spark.ml.feature.IDF
import org.apache.spark.ml.feature.StopWordsRemover
import org.apache.spark.ml.feature.Tokenizer
import org.apache.spark.ml.feature.HashingTF
import org.apache.spark.sql.Row
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.mllib.linalg.Vector
import org.apache.spark.mllib.util.MLUtils
import scala.collection.mutable.WrappedArray
import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.ml.feature.VectorIndexer
import
org.apache.spark.ml.classification.DecisionTreeClassifier
import org.apache.spark.ml.feature.IndexToString
import
org.apache.spark.ml.evaluation.MulticlassClassificationEval
uator
import
org.apache.spark.ml.classification.DecisionTreeClassificatio
nModel
import org.apache.spark.ml.classification.NaiveBayes

object SparkNaiveBayes {

  case class Article(id: Integer, text: String, cont_name:
Integer)

  def main(args: Array[String]) {
    val xml = scala.xml.XML.loadFile(args(0))
    val pw = new PrintWriter(new File(args(1)))
    val mediawiki = (xml \\ "mediawiki" \\
"page").foreach {
      page =>
        var result = (page \ "id").text + "," +
          (page \ "revision" \
"text").text.toLowerCase().trim().replaceAll("[^A-Za-z ]+",
" ").trim().replaceAll(" +", " ").toString() + ","
        if ((page \ "revision" \ "contributor" \
"id").text.length() > 0)
          result = result + (page \ "revision" \ "contributor" \
"id").text
        else
          result = result + "0"
        pw.println(result)
    }
    pw.close()

    val conf = new
SparkConf().setAppName("SparkDecisionTree").setMaster(
"local")
    val sc = new SparkContext(conf)
    val sqlContext = new SQLContext(sc)

    import sqlContext.implicits._

    def parseArticle(str: String): Article = {
      val line = str.split(",", -1)
      Article(line(0).toInt, line(1), line(2).toInt)
    }

    val textRDD = sc.textFile(args(1))

    val articleRDD = textRDD.map(parseArticle).cache()

    val dataset = articleRDD.toDF()

    val tokenizer = new
Tokenizer().setInputCol("text").setOutputCol("words")

    val wordsData = tokenizer.transform(dataset)

    val remover = new StopWordsRemover()
      .setInputCol(tokenizer.getOutputCol)
      .setOutputCol("filtered")

    val filtereddata = remover.transform(wordsData)

    val wordcount = filtereddata.explode("filtered",
"explode")((line: WrappedArray[String]) => line)

    import org.apache.spark.sql.functions._

    import org.apache.spark.sql.expressions.Window

    val count = wordcount.groupBy("filtered",
"cont_name", "explode").count()

    val finalcount =
Window.partitionBy(count("filtered")).orderBy(count("coun
t").desc)

    val result = count.withColumn("max_count",
first(count("count")).over(finalcount).as("max_count")).filte
r("count = max_count")

    val data = result.select("cont_name", "explode")

    sc.parallelize(data.select("cont_name",
"explode").repartition(1).collect().toSeq).saveAsTextFile("r
esult_nb.txt")

    val assembler = new
VectorAssembler().setInputCols(Array("cont_name")).setO
utputCol("cont")
```

```
        val inputData = assembler.transform(data)

        val labelIndexer = new
StringIndexer().setInputCol("explode").setOutputCol("index
edLabel").fit(inputData)

        val contCount =
inputData.select("cont").distinct().count().toInt

        val featureIndexer = new
VectorIndexer().setMaxCategories(contCount).setInputCol(
"cont").setOutputCol("indexedFeatures").fit(inputData)

        val Array(trainingData, testData) =
inputData.randomSplit(Array(0.7, 0.3))

        val nb = new
NaiveBayes().setLabelCol("indexedLabel").setFeaturesCol(
"indexedFeatures")

        val labelConverter = new
IndexToString().setInputCol("prediction").setOutputCol("pr
edictedLabel").setLabels(labelIndexer.labels)

        val pipeline = new
Pipeline().setStages(Array(labelIndexer, featureIndexer, nb,
labelConverter))

        val model = pipeline.fit(trainingData)

        val predictions = model.transform(testData)

        sc.parallelize(predictions.select("predictedLabel",
"explode",
"cont").repartition(1).collect().toSeq).saveAsTextFile("nb_p
rediction.txt")

        val evaluator = new
MulticlassClassificationEvaluator().setLabelCol("indexedLa
bel").setPredictionCol("prediction").setMetricName("accura
cy")

        val accuracy = evaluator.evaluate(predictions)

        println("Accuracy = " + accuracy)

      }
}
```

Intermediate Output showing contributor with the most frequent word in the article text:

```
[293183,commonscat]
[293183,birthdecade]
[293183,births]
[293183,th]
[22027,howard]
293183,category]
[239710,category]
[22027,northern]
```

Naive Bayes Output:

```
[redirect,anjou,[0.0]]
[redirect,indiana,[0.0]]
[redirect,math,[0.0]]
[redirect,math,[0.0]]
[redirect,redirect,[0.0]]
[redirect,veneto,[0.0]]
[redirect,redirect,[2.0]]
[redirect,grandparent,[2077.0]]
[redirect,redirect,[2077.0]]
[redirect,pneumonia,[2133.0]]
```

VI. CONCLUSION

In this paper, Clustering and Classification of the Wikipedia articles is performed using Apache Spark Machine Learning(ML) Library. It has pre-defined classes for clustering using K-means and Naïve Bayes Classification and has rich APIs in Scala, Java, Python, R which makes it easier for coding and performing analysis on any dataset.

A brief execution of the entire project is as follows: https://youtu.be/x_6sqlZ9wxY

REFERENCES

[1] https://en.wikipedia.org/wiki/Machine_learning

[2] https://en.wikipedia.org/wiki/Cluster_analysis

[3] https://en.wikipedia.org/wiki/Classification

[4] https://en.wikipedia.org/wiki/Document_classification

[5] https://dzone.com/articles/6-sparkling-features-apache

[6] https://databricks.com/spark/about

[7] https://databricks.com/blog/2016/05/31/apache-spark-2-0-preview-machine-learning-model-persistence.html

[8] http://spark.apache.org/docs/latest/ml-clustering.html#k-means

[9]https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Clustering/K-Means

[10] http://spark.apache.org/docs/latest/mllib-classification-regression.html

[11]http://spark.apache.org/docs/latest/mllib-naive-bayes.html

[12] Laurent Weichberger. Databricks and Apache Spark, with Spark Streaming, Twitter4j, DataFrames and Machine Learning (ML) KMeans 1.4. [Part III]. October 2015. https://www.linkedin.com/pulse/databricks-apache-spark-streaming-twitter4j-machine-ml-weichberger?trk=hp-feed-article-title-like

[13]https://documents.software.dell.com/statistics/textbook/naive-bayes-classifier