**CLASSIFIER.JAVA:**

package com.chimpler.example.agri;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.StringReader;

import java.util.HashMap;

import java.util.Map;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.lucene.analysis.Analyzer;

import org.apache.lucene.analysis.TokenStream;

import org.apache.lucene.analysis.standard.StandardAnalyzer;

import org.apache.lucene.analysis.tokenattributes.CharTermAttribute;

import org.apache.lucene.util.Version;

"import org.apache.mahout.classifier.naivebayes.BayesUtils;

import org.apache.mahout.classifier.naivebayes.NaiveBayesModel;

import org.apache.mahout.classifier.naivebayes.StandardNaiveBayesClassifier;"

import org.apache.mahout.common.Pair;

import org.apache.mahout.common.iterator.sequencefile.SequenceFileIterable;

import org.apache.mahout.math.RandomAccessSparseVector;

import org.apache.mahout.math.Vector;

import org.apache.mahout.math.Vector.Element;

import org.apache.mahout.vectorizer.TFIDF;

import com.google.common.collect.ConcurrentHashMultiset;

import com.google.common.collect.Multiset;

/\*\*

\* http://www.chimpler.com

\*/

public class Classifier {

public static Map<String, Integer> readDictionnary(Configuration conf, Path dictionnaryPath) {

Map<String, Integer> dictionnary = new HashMap<String, Integer>();

for (Pair<Text, IntWritable> pair : new SequenceFileIterable<Text, IntWritable>(dictionnaryPath, true, conf)) {

dictionnary.put(pair.getFirst().toString(), pair.getSecond().get());

}

return dictionnary;

}

public static Map<Integer, Long> readDocumentFrequency(Configuration conf, Path documentFrequencyPath) {

Map<Integer, Long> documentFrequency = new HashMap<Integer, Long>();

for (Pair<IntWritable, LongWritable> pair : new SequenceFileIterable<IntWritable, LongWritable>(documentFrequencyPath, true, conf)) {

documentFrequency.put(pair.getFirst().get(), pair.getSecond().get());

}

return documentFrequency;

}

public static void main(String[] args) throws Exception {

if (args.length < 5) {

System.out.println("Arguments: [model] [label index] [dictionnary] [document frequency] [tweet file]");

return;

}

String modelPath = args[0];

String labelIndexPath = args[1];

String dictionaryPath = args[2];

String documentFrequencyPath = args[3];

String tweetsPath = args[4];

Configuration configuration = new Configuration();

// model is a matrix (wordId, labelId) => probability score

"NaiveBayesModel model = NaiveBayesModel.materialize(new Path(modelPath), configuration);

StandardNaiveBayesClassifier classifier = new StandardNaiveBayesClassifier(model);

// labels is a map label => classId

Map<Integer, String> labels = BayesUtils.readLabelIndex(configuration, new Path(labelIndexPath));

Map<String, Integer> dictionary = readDictionnary(configuration, new Path(dictionaryPath));

Map<Integer, Long> documentFrequency = readDocumentFrequency(configuration, new Path(documentFrequencyPath));

// analyzer used to extract word from tweet

Analyzer analyzer = new StandardAnalyzer(Version.LUCENE\_43);

int labelCount = labels.size();

int documentCount = documentFrequency.get(-1).intValue();

System.out.println("Number of labels: " + labelCount);

System.out.println("Number of documents in training set: " + documentCount);

BufferedReader reader = new BufferedReader(new FileReader(tweetsPath));

while(true) {

String line = reader.readLine();

if (line == null) {

break;

}

String[] tokens = line.split("/t");

int i=0;

if (tokens.length != 24) {

i++;

System.out.println("Skip line: " + line);

continue;

}

System.out.println("skipped lines: " + i );

String tweetId = tokens[0];

String tweet = tokens[19];

System.out.println("Tweet: " + tweetId + "\t" + tweet);

Multiset<String> words = ConcurrentHashMultiset.create();

// extract words from tweet

TokenStream ts = analyzer.tokenStream("text", new StringReader(tweet));

CharTermAttribute termAtt = ts.addAttribute(CharTermAttribute.class);

ts.reset();

int wordCount = 0;

while (ts.incrementToken()) {

if (termAtt.length() > 0) {

String word = ts.getAttribute(CharTermAttribute.class).toString();

Integer wordId = dictionary.get(word);

// if the word is not in the dictionary, skip it

if (wordId != null) {

words.add(word);

wordCount++;

}

}

}

// create vector wordId => weight using tfidf

Vector vector = new RandomAccessSparseVector(10000);

TFIDF tfidf = new TFIDF();

for (Multiset.Entry<String> entry:words.entrySet()) {

String word = entry.getElement();

int count = entry.getCount();

Integer wordId = dictionary.get(word);

Long freq = documentFrequency.get(wordId);

double tfIdfValue = tfidf.calculate(count, freq.intValue(), wordCount, documentCount);

vector.setQuick(wordId, tfIdfValue);

}

// With the classifier, we get one score for each label

// The label with the highest score is the one the tweet is more likely to

// be associated to

Vector resultVector = classifier.classifyFull(vector);

double bestScore = -Double.MAX\_VALUE;

int bestCategoryId = -1;

for(Element element: resultVector.all()) {

int categoryId = element.index();

double score = element.get();

if (score > bestScore) {

bestScore = score;

bestCategoryId = categoryId;

}

System.out.print(" " + labels.get(categoryId) + ": " + score);

}

System.out.println(" => " + labels.get(bestCategoryId));

}

analyzer.close();

reader.close();

}

}"

**TOP CATEGORY WORDS.JAVA:**

package com.chimpler.example.agri;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.SortedSet;

import java.util.TreeSet;

import java.util.Map.Entry;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.mahout.classifier.naivebayes.BayesUtils;

import org.apache.mahout.classifier.naivebayes.NaiveBayesModel;

import org.apache.mahout.common.Pair;

import org.apache.mahout.common.iterator.sequencefile.SequenceFileIterable;

/\*\*

\* http://www.chimpler.com

\*/

"public class TopCategoryWords {

public static Map<Integer, String> readInverseDictionnary(Configuration conf, Path dictionnaryPath) {

Map<Integer, String> inverseDictionnary = new HashMap<Integer, String>();

for (Pair<Text, IntWritable> pair : new SequenceFileIterable<Text, IntWritable>(dictionnaryPath, true, conf)) {

inverseDictionnary.put(pair.getSecond().get(), pair.getFirst().toString());

}

return inverseDictionnary;

}

public static Map<Integer, Long> readDocumentFrequency(Configuration conf, Path documentFrequencyPath) {

Map<Integer, Long> documentFrequency = new HashMap<Integer, Long>();

for (Pair<IntWritable, LongWritable> pair : new SequenceFileIterable<IntWritable, LongWritable>(documentFrequencyPath, true, conf)) {

documentFrequency.put(pair.getFirst().get(), pair.getSecond().get());

}

return documentFrequency;

}

public static Map<Integer, Long> getTopWords(Map<Integer, Long> documentFrequency, int topWordsCount) {

List<Map.Entry<Integer, Long>> entries = new ArrayList<Map.Entry<Integer, Long>>(documentFrequency.entrySet());

Collections.sort(entries, new Comparator<Map.Entry<Integer, Long>>() {

@Override

public int compare(Entry<Integer, Long> e1, Entry<Integer, Long> e2) {

return -e1.getValue().compareTo(e2.getValue());

}

});

Map<Integer, Long> topWords = new HashMap<Integer, Long>();

int i = 0;

for(Map.Entry<Integer, Long> entry: entries) {

topWords.put(entry.getKey(), entry.getValue());

i++;

if (i > topWordsCount) {

break;

}

}

return topWords;

}

public static class WordWeight implements Comparable<WordWeight> {

private int wordId;

private double weight;

public WordWeight(int wordId, double weight) {

this.wordId = wordId;

this.weight = weight;

}

public int getWordId() {

return wordId;

}

public Double getWeight() {

return weight;

}

@Override

public int compareTo(WordWeight w) {

return -getWeight().compareTo(w.getWeight());

}

}

public static void main(String[] args) throws Exception {

if (args.length < 4) {

System.out.println("Arguments: [model] [label index] [dictionnary] [document frequency]");

return;

}

String modelPath = args[0];

String labelIndexPath = args[1];

String dictionaryPath = args[2];

String documentFrequencyPath = args[3];

Configuration configuration = new Configuration();

// model is a matrix (wordId, labelId) => probability score

NaiveBayesModel model = NaiveBayesModel.materialize(new Path(modelPath), configuration);

// labels is a map label => classId

Map<Integer, String> labels = BayesUtils.readLabelIndex(configuration, new Path(labelIndexPath));

Map<Integer, String> inverseDictionary = readInverseDictionnary(configuration, new Path(dictionaryPath));

Map<Integer, Long> documentFrequency = readDocumentFrequency(configuration, new Path(documentFrequencyPath));"

Map<Integer, Long> topWords = getTopWords(documentFrequency, 10);

System.out.println("Top words");

for(Map.Entry<Integer, Long> entry: topWords.entrySet()) {

System.out.println(" - " + inverseDictionary.get(entry.getKey())

+ ": " + entry.getValue());

}

int labelCount = labels.size();

int documentCount = documentFrequency.get(-1).intValue();

System.out.println("Number of labels: " + labelCount);

System.out.println("Number of documents in training set: " + documentCount);

for(int labelId = 0 ; labelId < model.numLabels() ; labelId++) {

SortedSet<WordWeight> wordWeights = new TreeSet<WordWeight>();

for(int wordId = 0 ; wordId < model.numFeatures() ; wordId++) {

WordWeight w = new WordWeight(wordId, model.weight(labelId, wordId));

wordWeights.add(w);

}

System.out.println("Top 10 words for label " + labels.get(labelId));

int i = 0;

for(WordWeight w: wordWeights) {

System.out.println(" - " + inverseDictionary.get(w.getWordId())

+ ": " + w.getWeight());

i++;

if (i >= 10) {

break;

}

}

}

}

}

**TWEETTSVToSeq:**

package com.chimpler.example.agri;

import java.io.BufferedReader;

import java.io.FileReader;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.SequenceFile;

import org.apache.hadoop.io.SequenceFile.Writer;

import org.apache.hadoop.io.Text;

/\*\*

\* http://www.chimpler.com

\*/

"public class TweetTSVToSeq {

public static void main(String args[]) throws Exception {

if (args.length != 2) {

System.err.println("Arguments: [input tsv file] [output sequence file]");

return;

}

String inputFileName = args[0];

String outputDirName = args[1];

Configuration configuration = new Configuration();

FileSystem fs = FileSystem.get(configuration);

Writer writer = new SequenceFile.Writer(fs, configuration, new Path(outputDirName + "/chunk-0"),

Text.class, Text.class);

int count = 0;

BufferedReader reader = new BufferedReader(new FileReader(inputFileName));

Text key = new Text();

Text value = new Text();

while(true) {

String line = reader.readLine();

if (line == null) {

break;

}

String[] tokens = line.split("\t", 24);

if (tokens.length != 24) {

System.out.println("Skip line: " + line);

continue;

}

String category = tokens[6];

String id = tokens[0];

String message = tokens[19];

key.set("/" + category + "/" + id);

value.set(message);

writer.append(key, value);

count++;

}

reader.close();

writer.close();

System.out.println("Wrote " + count + " entries.");"

}

}

**TWEETTSV TO TRAININGSET.JAVA:**

package com.chimpler.example.agri;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.StringReader;

import java.util.HashMap;

import java.util.Map;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.SequenceFile;

import org.apache.hadoop.io.SequenceFile.Writer;

import org.apache.hadoop.io.Text;

import org.apache.lucene.analysis.Analyzer;

import org.apache.lucene.analysis.TokenStream;

import org.apache.lucene.analysis.standard.StandardAnalyzer;

import org.apache.lucene.analysis.tokenattributes.CharTermAttribute;

import org.apache.lucene.util.Version;

import org.apache.mahout.common.Pair;

import org.apache.mahout.common.iterator.sequencefile.SequenceFileIterable;

import org.apache.mahout.math.RandomAccessSparseVector;

import org.apache.mahout.math.Vector;

import org.apache.mahout.math.VectorWritable;

import org.apache.mahout.vectorizer.TFIDF;

import com.google.common.collect.ConcurrentHashMultiset;

import com.google.common.collect.Multiset;

public class TweetTSVToTrainingSetSeq {

public static Map<String, Integer> readDictionnary(Configuration conf, Path dictionnaryPath) {

Map<String, Integer> dictionnary = new HashMap<String, Integer>();

for (Pair<Text, IntWritable> pair : new SequenceFileIterable<Text, IntWritable>(dictionnaryPath, true, conf)) {

dictionnary.put(pair.getFirst().toString(), pair.getSecond().get());

}

return dictionnary;

}

public static Map<Integer, Long> readDocumentFrequency(Configuration conf, Path documentFrequencyPath) {

Map<Integer, Long> documentFrequency = new HashMap<Integer, Long>();

for (Pair<IntWritable, LongWritable> pair : new SequenceFileIterable<IntWritable, LongWritable>(documentFrequencyPath, true, conf)) {

documentFrequency.put(pair.getFirst().get(), pair.getSecond().get());

}

return documentFrequency;

}

public static void main(String[] args) throws Exception {

if (args.length < 4) {

System.out.println("Arguments: [dictionnary] [document frequency] [tweet file] [output file]");

return;

}

String dictionaryPath = args[0];

String documentFrequencyPath = args[1];

String tweetsPath = args[2];

String outputFileName = args[3];

Configuration configuration = new Configuration();

FileSystem fs = FileSystem.get(configuration);

Map<String, Integer> dictionary = readDictionnary(configuration, new Path(dictionaryPath));

Map<Integer, Long> documentFrequency = readDocumentFrequency(configuration, new Path(documentFrequencyPath));

int documentCount = documentFrequency.get(-1).intValue();

Writer writer = new SequenceFile.Writer(fs, configuration, new Path(outputFileName),

Text.class, VectorWritable.class);

Text key = new Text();

VectorWritable value = new VectorWritable();

Analyzer analyzer = new StandardAnalyzer(Version.LUCENE\_43);

BufferedReader reader = new BufferedReader(new FileReader(tweetsPath));

while(true) {

String line = reader.readLine();

if (line == null) {

break;

}

String[] tokens = line.split("\t", 24);

String label = tokens[6];

String tweetId = tokens[0];

String tweet = tokens[19];

key.set("/" + label + "/" + tweetId);

Multiset<String> words = ConcurrentHashMultiset.create();

// extract words from tweet

TokenStream ts = analyzer.tokenStream("text", new StringReader(tweet));

CharTermAttribute termAtt = ts.addAttribute(CharTermAttribute.class);

ts.reset();

int wordCount = 0;

while (ts.incrementToken()) {

if (termAtt.length() > 0) {

String word = ts.getAttribute(CharTermAttribute.class).toString();

Integer wordId = dictionary.get(word);

// if the word is not in the dictionary, skip it

if (wordId != null) {

words.add(word);

wordCount++;

}

}

}

// create vector wordId => weight using tfidf

Vector vector = new RandomAccessSparseVector(10000);

TFIDF tfidf = new TFIDF();

for (Multiset.Entry<String> entry:words.entrySet()) {

String word = entry.getElement();

int count = entry.getCount();

Integer wordId = dictionary.get(word);

// if the word is not in the dictionary, skip it

Long freq = documentFrequency.get(wordId);

double tfIdfValue = tfidf.calculate(count, freq.intValue(), wordCount, documentCount);

vector.setQuick(wordId, tfIdfValue);

}

value.set(vector);

writer.append(key, value);

}

analyzer.close();

reader.close();

writer.close();

}

}

MOBILE APPLICATIONCODE:

<html>

<head>

<meta charset="ISO-8859-1">

<title>Revive</title>

</head>

<body>

<p> <h1>

<centre>Revive yourself from Recalls</centre>

</h1></p>

<form id="Recalls details" name="Recall details">

<label>Specify the Genre:</label>

<select id="Genre">

<option value="select one from below">Select one option</option>

<option value="EQUIPMENT:OTHER:LABELS">EQUIPMENT:OTHER:LABELS</option>

<option value="SERVICE BRAKES, AIR:CONTROLS:HAND CONTROL">SERVICE BRAKES, AIR:CONTROLS:HAND CONTROL</option>

<option value="VEHICLE SPEED CONTROL:LINKAGES ELECTRONIC MOBILITY CONTROLS">VEHICLE SPEED CONTROL:LINKAGES ELECTRONIC MOBILITY CONTROLS</option>

<option value="VEHICLE SPEED CONTROL CHANCE COACH, INC">VEHICLE SPEED CONTROL CHANCE COACH, INC</option>

<option value="POWER TRAIN:MANUAL TRANSMISSION CHANCE COACH, INC">POWER TRAIN:MANUAL TRANSMISSION CHANCE COACH, INC</option>

<option value="EQUIPMENT:OTHER:LABELS">EQUIPMENT:OTHER:LABELS</option>

<option value="SUSPENSION:REAR:AXLE:SPINDLE">SUSPENSION:REAR:AXLE:SPINDLE</option>

</select>

<button id="search">Search</button>

<br />

</form>

<p></p>

<div id="results1"></div>

<p></p>

<div id="results"></div>

</body>

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js

"></script>

<script>

function on\_data(data) {

$('#results1').empty();

var users = data.response.docs;

$.each(users, function(id,title) {

var endlist = 'Recommended Movies'+users;

$('#results1').prepend('<div>' + endlist + '</div>');

}

function on\_search() {

var query = $('#Genre').val();

if (query.length == 0) {

return;"

}

REFERENCE:

USED THE CODE OF CLASSIFIER SYSTEM DEVELOPED FOR LAB-4 ASSIGNMENT.