|  |
| --- |
|  |
| CMSC678 – INTRODUCTION to Machine Learning – HW1 |
| Homework 1 |

|  |
| --- |
| Shrinivas Kane  2-17-2015 |

# Part 1:

|  |  |  |
| --- | --- | --- |
| Task | Accuracy | Underfit / overfit |
| J48 default | 85.17 |  |
| J48 unpruned | 83.44 | Yes it has overfit pruned version of J48 is smaller and more generalized. Please note there is no node *native-country = United-States* in pruned tree which gives better generalization |
| decision stump | 76.07 | Yes Underfit. As Decision stump splits on just one single input feature. It splits on marital-status which has maximum information gain but may wrongly classifies instances having different education, native country and so on. |

# Part 2:

## 1.

Total number of iterations for convergence: 30

|  |  |  |  |
| --- | --- | --- | --- |
| Cluster id | Most common digit | Number of Instances other than most common digit | Total number of instances in cluster |
| 1 | 7 | 55 | 616 |
| 2 | 5 | 643 | 955 |
| 3 | 4 | 651 | 934 |
| 4 | 1 | 499 | 1614 |
| 5 | 9 | 506 | 989 |
| 6 | 4 | 372 | 689 |
| 7 | 0 | 57 | 801 |
| 8 | 8 | 375 | 930 |
| 9 | 3 | 501 | 1133 |
| 10 | 6 | 609 | 1339 |
| Total | - | 4268 | 10000 |

## 2.

Average number of iterations to convergence: 25

Average number of average number of instances that are in the wrong cluster: 4095

## 3.

|  |  |
| --- | --- |
| Cluster id | Most Dominant digits |
| 1 | 3,8,5 |
| 2 | 7,9 |
| 3 | 0,6 |
| 4 | 9,4 |
| 5 | 1,2 |

Digits are grouped in same cluster due to their similar shapes (having closed ranged entries in their feature vector). Number Of iteration to converge: 18

## 4.

Average number of iterations to convergence: 23

Average number of average number of instances that are in the wrong cluster: 3908

By randomly choosing an instance that represents each of the digits we got slight improvement in accuracy and less number of iterations to converge.

This behavior is obvious as we have chosen centroid which represent each digit. Thus it found similar digits in proximity resulting into slightly better in performance. Furthermore as we picked up centroid within cluster there is less changes in centroid vector which gave us slightly less number of iterations for convergence.

# 5. Source code

### Data.java

package edu.umbc.cmsc678.hw1.utils;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.security.InvalidAlgorithmParameterException;

import java.util.ArrayList;

import java.util.List;

/\*\*

\*

\* @author Shrinivas

\* Class to hold

\* 1. input data

\* 2. assign actual labels(if available)

\*/

public class Data {

private double[] featureVector;

private int actualLable;

private int assignedLable;

public Data(double[] vector, int label) {

featureVector = vector;

actualLable = label;

}

public Data(){

}

/\*\*\*

\* Get feature vector from input data

\* @param FileName input file name for data

\* @return feature vectors as a list

\* @throws IOException

\*/

public List<double[]> getVector(String FileName) throws IOException {

FileReader fr = null;

File inputFile = null;

BufferedReader br = null;

inputFile = new File(FileName);

boolean isEof = false;

List<double[]> inputData = new ArrayList<double[]>();

try {

fr = new FileReader(inputFile);

br = new BufferedReader(fr);

while (!isEof) {

String line = br.readLine();

if (line == null) {

isEof = true;

} else {

String[] rowEntries = line.split("\\s+");

double[] lineEntries = new double[rowEntries.length];

int loopIndex = 0;

for (String entry : rowEntries) {

lineEntries[loopIndex++] = Double.parseDouble(entry);

}

inputData.add(lineEntries);

}

}

} catch (IOException e) {

System.err.println("Falied to read Line");

e.printStackTrace();

} finally {

br.close();

fr.close();

}

return inputData;

}

/\*\*\*

\* get label from input file

\* @param labelFileName

\* @return actual cluster labels

\* @throws IOException

\*/

public List<Integer> getlabels(String labelFileName) throws IOException {

List<Integer> lablesData = new ArrayList<Integer>();

BufferedReader br = new BufferedReader(new FileReader(new File(

labelFileName)));

boolean isEof = false;

while (!isEof) {

String line = br.readLine();

if (line == null) {

isEof = true;

} else {

lablesData.add(Integer.parseInt(line));

//System.out.println(line);

}

}

return lablesData;

}

/\*\*\*

\*

\* @param FeatureVectorFilePath read raw data from input text file

\* @param labelFilePath read actual cluster label from input text file path

\* @return Data object with feature vector along with actual label (if available)

\* @throws IOException

\* @throws InvalidAlgorithmParameterException

\*/

public List<Data> getData(String FeatureVectorFilePath, String labelFilePath)

throws IOException, InvalidAlgorithmParameterException {

List<double[]> inputData = getVector(FeatureVectorFilePath);

List<Integer> lablesData = getlabels(labelFilePath);

List<Data> data = new ArrayList<>();

if (inputData.size() != lablesData.size()) {

throw new InvalidAlgorithmParameterException("Lable and vectors mismatch");

} else {

for (int i=0; i<inputData.size(); i++ ) {

Data d = new Data(inputData.get(i), lablesData.get(i));

data.add(d);

}

}

return data;

}

// getters and setters

public double[] getFeatureVector() {

return featureVector;

}

public void setFeatureVector(double[] featureVector) {

this.featureVector = featureVector;

}

public int getActualLable() {

return actualLable;

}

public void setActualLable(int actualLable) {

this.actualLable = actualLable;

}

public int getAssignedLable() {

return assignedLable;

}

public void setAssignedLable(int assignedLable) {

this.assignedLable = assignedLable;

}

}

### CentroidPicker.java

package edu.umbc.cmsc678.hw1.utils;

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

/\*\*\*

\* Picks centroid vectors

\* @author Shrinivas

\*

\*/

public class CentroidPicker {

int centroidNum;

int vectorSize;

// Switch to Randomly sample k = 10 instances

public static final int RANDOM\_CENTROIDS =1;

// Switch to Randomly choose an instance that represents each of the digits

public static final int SPECIFIC\_CENTROIDS=2;

public int getCentroidNum() {

return centroidNum;

}

public void setCentroidNum(int centroidNum) {

this.centroidNum = centroidNum;

}

/\*\*

\* Sets number of centroids

\*/

public CentroidPicker(int centroids) {

centroidNum = centroids;

}

/\*\*\*

\* Generates random centroids from data

\* @param data Data as feature vectors

\* @param type Switch 1 for Randomly sample or 2 for instance that represents each of the digits

\* @return list of centroids

\*/

public List<double[]> getCentroids(List<Data> data, int type) {

Random r = new Random();

int j=0;

List<double[]> centroids = new ArrayList<double[]>();

for (int i = 0; i < centroidNum; i++) {

if (type == 1) {

centroids.add((data.get(r.nextInt(data.size()))

.getFeatureVector()));

} else {

while(true){

//System.out.println(data.size());

int index = r.nextInt(data.size());

Data d = data.get(index);

if(j == d.getActualLable()){

//System.out.println("j "+j+"Vales "+data.get(index).getActualLable()+" Index Vale"+ index);

centroids.add(data.get(index).getFeatureVector());

j++;

break;

}

}

}

}

/\*System.out.println("Centroids Are-");

for (double[] is : centroids) {

for (double i : is) {

System.out.print(i+"\t");

}

System.out.println("\n");

}\*/

return centroids;

}

}

### SimpleKMeans.java

package edu.umbc.cmsc678.hw1;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.security.InvalidAlgorithmParameterException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Map.Entry;

import edu.umbc.cmsc678.hw1.utils.CentroidPicker;

import edu.umbc.cmsc678.hw1.utils.Data;

/\*\*\*

\* Basic implementation of k-means clustering

\* @author Shrinivas

\*

\*/

public class SimpleKMeans {

int itrCOunter = 0;

/\*\*\*

\* get cluster id based on centroids. i have used euclidean distance to categorize the cluster

\* @param centroids centroid vector for each cluster

\* @param vector feature vector from data set

\* @return

\*/

int getClusterId(List<double[]> centroids, double[] vector) {

int centroidIndex = 0;

double distSum = 0;

double minDistance = Double.MAX\_VALUE;

int clusterIndex = 0;

for (double[] centroid : centroids) {

distSum = 0;

for (int i = 0; i < vector.length; i++) {

double d1 = Math.abs((vector[i] - centroid[i]));

distSum = distSum + (d1 \* d1);

}

double distance = Math.sqrt(distSum);

// System.out.println(distance);

if (distance < minDistance) {

clusterIndex = centroidIndex;

minDistance = distance;

}

centroidIndex++;

}

return clusterIndex;

}

/\*\*\*

\* get all vectors and update centroid vector by averaging out all data points in feature vector(s)

\* @param clusters cluster id and associated feature vectors

\* @return new set of centroids

\*/

List<double[]> updateCentoid(Map<Integer, List<Data>> clusters) {

List<double[]> centoids = new ArrayList<double[]>();

for (Entry<Integer, List<Data>> e : clusters.entrySet()) {

int vectorSize = e.getValue().get(0).getFeatureVector().length;

// System.err.println(vectorSize);

double[] centroidArr = new double[vectorSize];

for (int i = 0; i < vectorSize; i++) {

for (Data d : e.getValue()) {

centroidArr[i] += d.getFeatureVector()[i];

}

}

for (int i = 0; i < vectorSize; i++) {

centroidArr[i] = (int) (centroidArr[i] / e.getValue().size());

}

centoids.add(centroidArr);

}

/\*

\* System.out.println("Centroids Are-"); for (double[] is : centoids) {

\* for (double i : is) { System.out.print(i+"\t"); }

\* System.out.println("\n"); }

\*/

return centoids;

}

/\*\*\*

\* 1. Generates centroids

\* 2. Classify each data point to cluster

\* 3. Check for convergence

\* @param inputData input data as feature vectors

\* @param cnetroidType 1 if to select random data point 2 if to select specific point in cluster

\* @param clusterNum number of k in k-means cluster

\* @return clusters as map of <cluster id,data vector(s)>

\*/

public Map<Integer, List<Data>> getCluster(List<Data> inputData,

int cnetroidType, int clusterNum) {

Map<Integer, List<Data>> clusters = null;

List<double[]> centroids = new CentroidPicker(clusterNum)

.getCentroids(inputData, cnetroidType);

int loopIndex = 0;

boolean isToStop = false;

while (!isToStop) {

clusters = new HashMap<Integer, List<Data>>();

for (Data d : inputData) {

int cid = getClusterId(centroids, d.getFeatureVector());

List<Data> clusteredData = clusters.get(cid);

if (clusteredData == null) {

clusteredData = new ArrayList<Data>();

}

clusteredData.add(d);

clusters.put(cid, clusteredData);

}

List<double[]> newCentroids = updateCentoid(clusters);

loopIndex++;

if (isConverged(newCentroids, centroids)) {

isToStop = true;

}

centroids = newCentroids;

}

// System.err.println(loopIndex+",0,0");

itrCOunter += loopIndex;

// System.out.println(itrCOunter+",0,0");

return clusters;

}

/\*\*\*

\* As centroid vectors datatype is double thus if centroids don't change by 1 percent we say k- means is converged

\* @param newCentroids old centroid vectors

\* @param oldCentroids new centroid vectors

\* @return true if converged false else

\*/

boolean isConverged(List<double[]> newCentroids, List<double[]> oldCentroids) {

int j = 0;

for (double[] oldVector : oldCentroids) {

double[] newVector = newCentroids.get(j);

for (int i = 0; i < oldVector.length; i++) {

//

if (newVector[i] > oldVector[i] + 2

|| newVector[i] < oldVector[i] - 2) {

return false;

}

}

j++;

}

return true;

}

/\*\*\*

\*

\* @param args

\* @throws IOException

\* @throws InvalidAlgorithmParameterException

\*/

public static void main(String[] args) throws IOException,

InvalidAlgorithmParameterException {

Data d = new Data();

int errRate = 0;

List<Data> testData = d.getData(args[0], args[1]);

// for(int i=0;i< 10; i++){

// System.out.println("Itration,"+i+",");

Map<Integer, List<Data>> x = new SimpleKMeans().getCluster(testData,

CentroidPicker.SPECIFIC\_CENTROIDS, Integer.parseInt(args[2]));

FileWriter fr = new FileWriter(new File(args[3]));

List<Map<Integer, Integer>> outputStats = new ArrayList<Map<Integer, Integer>>();

Map<Integer, Integer> clusterStats = null;

for (Entry<Integer, List<Data>> e : x.entrySet()) {

clusterStats = new HashMap<Integer, Integer>();

// System.err.println("\n"+e.getKey());

for (Data data : e.getValue()) {

if (clusterStats.containsKey(data.getActualLable())) {

int freq = clusterStats.get(data.getActualLable());

clusterStats.put(data.getActualLable(), freq + 1);

} else {

clusterStats.put(data.getActualLable(), 1);

}

for (double p : data.getFeatureVector()) {

fr.append(p + ",");

// System.out.print(p+",");

}

// System.out.print(data.getActualLable()+","+e.getKey()+"\n");

fr.append(data.getActualLable() + "," + e.getKey() + "\n");

}

outputStats.add(clusterStats);

fr.flush();

}

fr.close();

int totalCorrctInstance = 0;

int totalInstance = 0;

for (Map<Integer, Integer> map : outputStats) {

// System.out.println(map.toString());

int maxIns = 0;

int totCluster = 0;

int cid = 0;

for (Entry<Integer, Integer> e : map.entrySet()) {

if (maxIns < e.getValue()) {

maxIns = e.getValue();

cid = e.getKey();

}

totCluster += e.getValue();

totalInstance += e.getValue();

}

// System.err.println(cid+","+(totCluster-maxIns)+","+totCluster+","+(double)(((totCluster-maxIns)\*100)/totCluster));

totalCorrctInstance += maxIns;

}

System.out.println("Total," + totalCorrctInstance + ","

+ (totalInstance - totalCorrctInstance) + "," + totalInstance);

errRate += (totalInstance - totalCorrctInstance);

}

// System.out.println("tota Err Rate:"+ (errRate/10));

// System.out.println("avg itr:"+(itrCOunter/10));

// }

}