Problem

When presented with a set of data, how effective are the largest class and first nearest neighbor algorithms at determining the class of unknown data?

Algorithms Implemented:

The two algorithms implemented were largest class and first nearest neighbor. Largest class was the easier of the two to implement, as it simply counts which of the classes is most apparent and assigning all the unknown data that class. First nearest neighbor uses the Euclidian distance formula, ignoring the square root as it isn’t necessary for comparisons sake, to see which of all the known data points is nearest the unknown datum and assigns that class to each point of datum as they are compared.

Experimental Data

Two experiments were run, one where the percent split was changed, and one where the data sent for prediction was the training set and not the testing set.

Discussion of results

For the first experiment the following percentages were used: 1, 5, 10, 20, 25, 50, 75, 80 & 90. These percentages were used to highlight the trends that arise as the split percentage reaches 100%, which returns that all the data matches regardless. With only five percent of the data to work with nearest neighbor shoots pass random guesses (33% for three categories) to almost 90% accuracy and hitting 100% accuracy with 75 of the data. Largest class stays near random guess throughout, getting better most likely due to the data decreasing in number and depending on how the split is done, might make it more likely to guess better as a result. Over all the data shows that nearest neighbor is by far more effective.

The second test had rather predictable results. Regardless of size nearest neighbor hit 100% since all the data being compared had itself in the training set, so the distance never reached anything lower than zero, since a point can’t get any closer to any other point than itself. Because of this however largest class never broke past random guesses. For the same reason the guess got better slightly as percentage split increased in the first test, it did so poorly in this test. Regardless the test shows that the algorithms work as they are supposed to, with nearest neighbor getting better the more the data represents an actual correlation to classification, and largest class only being as good as the largest class is large.

Conclusion

The algorithms work as intended, which is to say that the effectiveness of both varies. Largest class is only effective if the largest class is truly much larger than the other classes, like with the case of handedness in people, otherwise being only as good as a random guess. Nearest neighbor worked well but also has problems. If the data doesn’t exactly correlate to the classes outcome it could potentially throw some errors as a result of trying to make comparisons that don’t actually mean anything. Furthermore, some data could end up being the most important due to its ‘weight’ in comparison to the other data points. In conclusion nearest neighbor is a good algorithm since its simplicity allows for easy understanding of why data may be important to determining a class, whereas largest class amounts in most cases to being a random guess.

Appendix

// Simple program to read classify Iris dataset

// using majority class predition and one-bnearest-neighbor

// Programmed by Olac Fuentes

// Last modified January 18, 2018

package ml1;

import java.io.\*;

import java.util.Random;

public class ml1{

public static dataset readData(String filename){

double [][] features = new double [150][4];

int [] label = new int[150];

String line;

String labelName;

String [] splitString;

try{

FileReader fileReader = new FileReader(filename);

BufferedReader bufferedReader = new BufferedReader(fileReader);

for (int i=0;i<150;i++){

line = bufferedReader.readLine();

splitString = line.split(",");

for (int j=0;j<4;j++)

features[i][j] = Double.valueOf(splitString[j]);

labelName = splitString[4];

if (labelName.equals("setosa "))

label[i]=0;

else if (labelName.equals("versicolor "))

label[i]=1;

else

label[i]=2;

}

}

catch(FileNotFoundException ex) {

System.out.println(

"Unable to open file '" +

filename + "'");

}

catch(IOException ex) {

System.out.println(

"Error reading file '"

+ filename + "'");

}

return new dataset(features, label);

}

public static dataset[] splitData(dataset X, float percent\_train){

// Shuffles X and then splits it such that the first percent\_train examples

// go to the train set (T[0])and the rest go to the test set (T[1]).

int limit = (int)(percent\_train/100\*X.label.length);

dataset [] T = new dataset[2];

T[0] = new dataset(limit,X.features[0].length);

T[1] = new dataset(X.label.length-limit,X.features[0].length);

Random rand = new Random(314159);

double[] lineHold;

int holder;

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

int pos = rand.nextInt((X.label.length - i)) + i;

lineHold = X.features[i];

X.features[i] = X.features[pos];

X.features[pos] = lineHold;

holder = X.label[i];

X.label[i] = X.label[pos];

X.label[pos] = holder;

}

for(int i = 0; i < X.label.length; i++)

if(limit > 0){

T[0].features[i] = X.features[i];

T[0].label[i] = X.label[i];

limit--;

}else{

T[1].features[X.label.length-i-1] = X.features[i];

T[1].label[X.label.length-i-1] = X.label[i];

}

return T;

}

public static double accuracy(int [] true\_label, int [] predicted\_label){

// Returns the fraction of examples for which the prediction is correct. That is, the fraction

// of examples for which true\_label[i] == predicted\_label[i]

int numCorrect = 0;

for(int i = 0; i < true\_label.length; i++)

if(true\_label[i] == predicted\_label[i])

numCorrect++;

return 100 - (Math.abs(numCorrect - true\_label.length)/(float) true\_label.length) \* 100;

}

public static int [] predict\_majority(dataset train\_set, double [][] test\_features){

// Finds majority class in train\_set and predicts that class for every element in test set

// It doesn't take values of test\_features into consideration

int [] pred = new int[test\_features.length];

int[] val = new int[3];

for(int i = 0; i < train\_set.label.length; i++)

val[train\_set.label[i]]++;

int holder = 0;

int holder2 = Integer.MAX\_VALUE;

int max = 0;

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

holder = val[i];

if(holder < holder2){

holder2 = holder;

max = i;

}

}

for(int i = 0; i < test\_features.length; i++)

pred[i] = max;

return pred;

}

public static int [] predict\_nearest\_neighbor(dataset train\_set, double [][] test\_features){

// Assigns to every test example the class of the most similar example in the training set

int [] pred = new int[test\_features.length];

double distance = Integer.MAX\_VALUE;

double temp = 0;

for(int i = 0; i < test\_features.length; i++){

distance = Integer.MAX\_VALUE;

for(int j = 0; j < train\_set.features.length; j++){

temp = 0;

for(int k = 0; k < test\_features[i].length; k++)

temp += Math.pow((test\_features[i][k] - train\_set.features[j][k]), 2);

if(temp < distance){

distance = temp;

pred[i] = train\_set.label[j];

}

}

}

return pred;

}

public static void printArray(double[] X){

// Prints 1D array

// Used for debugging purposes

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

System.out.print(X[i]+" ");

}

public static void printlnArray(double[] X) {

// Prints 1D array

// Used for debugging purposes

printArray(X);

System.out.println();

}

public static void printlnArray(double[][] X, int n) {

// Prints n first lines of 2D array

// Used for debugging purposes

for(int i=0;i<n;i++)

printlnArray(X[i]);

}

public static void main(String[] args) {

dataset X = readData("iris.txt");

dataset[] T = splitData(X,75); //T[0] is the training set, T[1] is the test set

//Next few lines used for debugging

//System.out.println("X:");

//printlnArray(X.features,10);

//System.out.println("T[0]:");

//printlnArray(T[0].features,10);

//System.out.println("T[1]:");

//printlnArray(T[1].features);

//System.out.println(T[0].label.length);

//System.out.println(T[1].label.length);

int [] pred\_majority = predict\_majority(T[0], T[1].features);

int [] pred\_1nn = predict\_nearest\_neighbor(T[0], T[1].features);

System.out.printf("Accuracy using majority label prediction: %.2f\n", accuracy(T[1].label,pred\_majority));

System.out.printf("Accuracy using nearest\_neighbor prediction: %.2f\n", accuracy(T[1].label,pred\_1nn));

}

}