Machine Learning Coursework 1

100135292 (`yqm15fqu`)

Wed, 1 May 2019 14:34

Contents

KNN.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package machinelearning_coursework;

import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import weka.classifiers.AbstractClassifier;
import weka.classifiers.Classifier;
import weka.core.Debug.Random;
import weka.core.Instance;
import weka.core.Instances;
import static weka.core.Utils.mean;

/**
 *
 * @author yqm15fqu
 */
public class KNN extends AbstractClassifier {

    private ArrayList<Instance> nearestNeighbours;
    private boolean standardiseAttributes = true;
    private boolean weightedVoting = false;
    private boolean leaveOneOut = false;
    private boolean draw = false;
    private Instances trainData;
    private double[] distances;
    private int maxK = 100;
    private int k = 1;

    public void setK(int x) {
        this.k = x;
    }

    public void setStandardise(boolean x) {
        this.standardiseAttributes = x;
    }

    public void setLeaveOneOut(boolean x) {
        this.leaveOneOut = x;
    }

    public void setWeightedVoting(boolean x) {
        this.weightedVoting = x;
    }

    public int getK() {
        return this.k;
    }

    public static double accuracy(Classifier c, Instances test) throws Exception
       {

        double result = 0;
        double accuracy = 0;
        int counter = 0;
        for (Instance instance : test) {
            result = c.classifyInstance(instance);
            if (result == instance.classValue()) {
                counter++;
            }
        }
        accuracy = (((double) counter / test.size()) * 100);

        return accuracy;
    }

    public static int mode(double[] input) {
        int[] count = new int[input.length + 1];
        //cont the occurrences
        for (int i = 0; i < input.length; i++) {
            count[(int) input[i]]++;
        }
        //go backwards and find the count with the most occurrences
        int index = count.length - 1;
        for (int i = count.length - 2; i >= 0; i--) {
            if (count[i] >= count[index]) {
                index = i;
            }
//                System.out.println(i + " is a count of " + count[i]);
        }

        return index;
    }

    private double[] kAccuracy(double[][] distArray) {
        double[][] allClassValues = new double[trainData.size()][maxK + 1];

        for (int i = 0; i < distArray.length; i++) {//loop through all rows
            ArrayList<InstanceContainer> allVals = new ArrayList<>(trainData.size
                ());

            for (int l = 0; l < distArray.length; l++) {     // loop through each
                column in the current row
                 InstanceContainer temporary = new InstanceContainer(trainData.get
                    (l), distArray[i][l]);
                 allVals.add(temporary);
            }
            //sort the arrayList by distances, so now have ascending order of
                distances and their respective instances
            Collections.sort(allVals);

            //get all instane class numbers, so now have a 2d array of ordered
                class numbers based on the instance distance.
            //each row of the 2d array represents the training data position that
                 has the distances relate to.
            //for instance row allClassVals[3][0] is trainingData.get(3) and all
                values associated with it.
            for (int j = 0; j <= this.maxK; j++) {
                allClassValues[i][j] = allVals.get(j).getInstance().classValue();
            }
        }
        double[][] kAccArray = new double[distArray.length][this.maxK + 1];

        for (int i = 0; i < trainData.size(); i++) {
            double classNum = trainData.get(i).classValue();
            //calculate accuracy
            for (int s = 1; s <= this.maxK; s++) {
                double totalPercent = 0;
                double[] rowClassVals = new double[trainData.numClasses()];
                for (int l = 1; l <= s; l++) {
                    rowClassVals[(int) allClassValues[i][l]]++;
                }

                totalPercent = rowClassVals[(int) classNum];
                totalPercent = (totalPercent / s) * 100;
                kAccArray[i][s] = totalPercent;
            }
        }

        double[] bestKValues = new double[trainData.size()];
        for (int i = 0; i < trainData.size(); i++) {
            //find the position that is highest and return it.
            double kVal = findHighestPos(kAccArray[i]);

            bestKValues[i] = kVal;
        }

        return bestKValues;
    }
    //find the mean
    public double mean(double[] accuracies) {
        double answer = 0;

        for (int i = 0; i < accuracies.length; i++) {
            answer += accuracies[i];
        }

        answer = answer / accuracies.length;

        return answer;
    }
    //find the highest pos
    private int findHighestPos(double[] x) {
        Random rand = new Random();
        double max = x[1];
        int value = 0;
        int doubleCounter = 0;
        double[] drawPos = new double[x.length];

        for (int i = 0; i < x.length; i++) {
            if (x[i] > max) {
                draw = false;
                max = x[i];
                drawPos = new double[x.length];
                drawPos[0] = i;
                doubleCounter = 1;
            } else if (x[i] == max) {
                draw = true;
                drawPos[doubleCounter] = i;
                doubleCounter++;
            }
        }
        //randomly sorts draws
        if (draw) {
//              System.out.println("Draw! settled randomly."); //For debugging,
   remove later.
            double[] posArr = new double[doubleCounter];
            for (int i = 0; i < doubleCounter; i++) {
                posArr[i] = drawPos[i];
            }
```

```
                int chosen = rand.nextInt(posArr.length);
                value = (int) posArr[chosen];
                draw = false;
            } else {
                value = (int) drawPos[0];
            }

            return value;
        }
        //standardisation of attributes
        private void standardiseAttributes(Instances train) {
            for (int j = 0; j < this.trainData.numAttributes() - 1; j++) {
                double mean = 0;
                double stdDev = 0;
                for (Instance instance : this.trainData) {
                    mean += instance.value(j);
                }
                mean = mean / this.trainData.size();
                for (int k = 0; k < this.trainData.size(); k++) {
                    stdDev += Math.pow(this.trainData.get(k).value(j) - mean, 2);
                }
                stdDev = stdDev / this.trainData.size();
                stdDev = Math.sqrt(stdDev);
                for (Instance instance : this.trainData) {
                    instance.setValue(j, ((instance.value(j) - mean) / stdDev));
//                      System.out.println(instance.value(j));
                }
            }
        }

        private int LOOCV() {
            //Set the size of maxK

            if (trainData.size() * 0.2 < 100) {
                this.maxK = (int) (this.trainData.size() * 0.2);
            } else {
                this.maxK = 100;
            }

            double[] kAccuracyArray = new double[this.maxK];
            // for all distances to all train data
            double[][] allDistances
                    = new double[this.trainData.size()][this.trainData.size()];

            for (int j = 0; j < trainData.size(); j++) {
                Instance curInst = this.trainData.get(j);
                //done to populate the distances array
                double temp = classifyInstance(curInst);
                //stores each collection of distances into each row
                for (int l = 0; l < this.trainData.size(); l++) {
                    if (l == j) {
                        allDistances[j][l] = -1;
                    } else {
                        allDistances[j][l] = this.distances[l];
                    }
                }
            }
            kAccuracyArray = kAccuracy(allDistances);
//            System.out.println(mode(kAccuracyArray));
//set k to the highest accuracy
            return mode(kAccuracyArray);
        }

```

```java
    @Override
    public void buildClassifier(Instances i) {
        this.trainData = i;

        if (standardiseAttributes) {
            standardiseAttributes(trainData);
        }
        if (leaveOneOut) { //need to double check this as always returns 1.
           Moving on for now.
            this.k = LOOCV();

            System.out.println("LOOCV done, K set to: " + this.k);
        }
    }

    private double setNewWeight(double dist) {
        double newWeight = (double) 1 / (double) (1 + dist);
        return newWeight;
    }

    @Override
    public double classifyInstance(Instance test) {

        ArrayList<InstanceContainer> allVals = new ArrayList<>(this.trainData.
           size());
        double[] distRatio = new double[test.numClasses()];
        //Used to clear distances of the previously stored distances.
        this.distances = new double[trainData.size()];
        ArrayList<Instance> lowestFound = new ArrayList();

        //Populates the double array "lowest" with the first k instances of the
           training data
        //so that there are instances to compare the rest of the train data to.
        for (int i = 0; i < trainData.size(); i++) {
            distances[i] = euclidDist(test, trainData.get(i));
            InstanceContainer temp = new InstanceContainer(trainData.get(i),
               distances[i]);
            allVals.add(temp);
        }

        Collections.sort(allVals);

        for (int i = 0; i < k; i++) {
            lowestFound.add(allVals.get(i).getInstance());
        }

        nearestNeighbours = lowestFound;

        if (weightedVoting) {

            for (int i = 0; i < k; i++) {
                allVals.get(i).getInstance().setWeight(setNewWeight(allVals.get(i
                   ).getDistance()));
            }

            for (Instance instance : lowestFound) {
                distRatio[(int) instance.classValue()] += instance.weight();
            }
            return findHighestPos(distRatio);
        }

        for (int i = 0; i < lowestFound.size(); i++) {
            distRatio[(int) lowestFound.get(i).classValue()]++;
        }

//          if(findHighestPos(distRatio)!= test.classValue())
//              System.out.println("INCORRECT");
        return findHighestPos(distRatio);

    }

    public double euclidDist(Instance test, Instance train) {
        double[] distance = new double[train.numAttributes()];
        double sum = 0;

        for (int i = 0; i < train.numAttributes() - 1; i++) {
            distance[i] = Math.pow((test.value(i) - train.value(i)), 2);
        }
        for (int i = 0; i < distance.length; i++) {
            sum += (distance[i]);
        }
        return Math.sqrt(sum);
    }

    @Override
    public double[] distributionForInstance(Instance test) {

        double[] distRatio = new double[trainData.numClasses()];

        if (nearestNeighbours != null) {
            for (Instance instance : nearestNeighbours) {
                int classValue = (int) instance.classValue();
                distRatio[classValue]++;
            }

            for (int i = 0; i < distRatio.length; i++) {
                double x = distRatio[i];
                distRatio[i] = ((x / nearestNeighbours.size()));
            }
//          System.out.println(Arrays.toString(distRatio));
            return distRatio;
        }
//          nearestNeighbours.clear();
        classifyInstance(test);


        for (Instance instance : nearestNeighbours) {
            int classValue = (int) instance.classValue();
            distRatio[classValue]++;
        }

        for (int i = 0; i < distRatio.length; i++) {
            double x = distRatio[i];
            distRatio[i] = ((x / nearestNeighbours.size()));
        }
//          System.out.println(Arrays.toString(distRatio));
        return distRatio;
    }
}
```

KnnEnsemble.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package machinelearning_coursework;

import static java.lang.Math.floor;
import java.util.ArrayList;
import machinelearning_coursework.*;
import static machinelearning_coursework.KNN.mode;
import weka.classifiers.Classifier;
import weka.core.Debug.Random;
import weka.core.Instance;
import weka.core.Instances;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.Remove;

/**
 *
 * @author yqm15fqu
 */
public class KnnEnsemble {

    private int ensembleSize = 50;
    private ArrayList<KNN> ensembleContainer = new ArrayList(ensembleSize);
    private Instances trainData;
    private boolean removeRandAttributes = false;
    private double[] ensemblePredictions = new double[ensembleSize];
    // for removing attributes.
    public void setRemoveRandAtt(boolean x) {
        removeRandAttributes = x;
    }

    //distribution for instance for the ensemble.
    public double[] getEnsemblePredictions(){
        double [] classVals = new double[trainData.numClasses()];
        for (int i = 0; i < ensemblePredictions.length; i++) {
            classVals[(int) ensemblePredictions[i]]++;
        }
        for (int i = 0; i < classVals.length; i++) {
            classVals[i] = (classVals[i]/ensemblePredictions.length);
        }
        return classVals;
    }

    public void setEndembleSize(int x) {
        this.ensembleSize = x;
        ensemblePredictions = new double[ensembleSize];
    }

    public int getEnsembleSize() {
        return ensembleSize;
    }
    //calculate ensemble accuracy by looking at what each classifier picks the
    //class value as, make a distribution, pick the highest.
    public static double accuracyEns(KnnEnsemble c, Instances test) throws
       Exception {

        double result = 0;
        double accuracy = 0;
        int counter = 0;
        for (Instance instance : test) {
            result = c.classifyInstanceEns(instance);
            if (result == instance.classValue()) {
                counter++;
            }
        }
        accuracy = (((double) counter / test.size()) * 100);

        return accuracy;
    }

    public void buildEnsemble(Instances ins) throws Exception {
        Random rand = new Random();

        //So that the Data is randomised
        ins.randomize(rand);
        Instances ensembleInstances = new Instances(ins, 0);
        int sampleSize = (int) Math.floor(ins.size() * 0.3);
        int attributeSize = (int) Math.floor(ins.numAttributes() * 0.4);

        if (removeRandAttributes) {
            int[] indices = new int[attributeSize];

            for (int i = 0; i < 10; i++) {
                int a = ins.numAttributes() - 1;
                for (int k = 0; k < attributeSize - 1; k++) {
                    indices[k] = rand.nextInt(a);
                    a--;
                }
                Remove removeFilter = new Remove();
                removeFilter.setAttributeIndicesArray(indices);
                removeFilter.setInputFormat(ins);
                trainData = Filter.useFilter(ins, removeFilter);
            }
        } else {
            trainData = ins;
        }
        //populate ensemble
        for (int i = 0; i < ensembleSize; i++) {
            KNN kNN = new KNN();
            Instances tempInst = ins;
            tempInst.randomize(rand);

            for (int j = 0; j < sampleSize; j++) {
//                  Instance inst = trainData.get(x).deleteAttributeAt(i);
                ensembleInstances.add(tempInst.get(j));
            }
            kNN.setLeaveOneOut(true);
            kNN.setStandardise(true);
            kNN.setWeightedVoting(true);
            kNN.buildClassifier(ensembleInstances);
            ensembleInstances = new Instances(trainData, 0);
//              System.out.println("KNN: " + i + " added. \n \n");
            ensembleContainer.add(kNN);
        }
        System.out.println("Completed Ensemble build.");
    }

    public double classifyInstanceEns(Instance test) {

        for (int i = 0; i < ensembleContainer.size(); i++) {
            ensemblePredictions[i] = ensembleContainer.get(i).classifyInstance(
                    test);
        }

//          System.out.println("THIS IS THE ENSEMBLE    ANSWER: " + mode(
    ensemblePredictions));
        double Answer = mode(ensemblePredictions);

        return Answer;
    }

}
```

Analysis.java

```
package machinelearning_coursework;

import java.io.File;
import java.io.IOException;
import evaluation.MultipleClassifierEvaluation;
import java.io.FileFilter;
import java.util.ArrayList;
import java.util.List;

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
/**
 *
 * @author yqm15fqu
 */
public class Analysis {

    public static String[] findFoldersInDirectory(String directoryPath) {
        File directory = new File(directoryPath);

        FileFilter directoryFileFilter = new FileFilter() {
            public boolean accept(File file) {
                return file.isDirectory();
            }
        };

        File[] directoryListAsFile = directory.listFiles(directoryFileFilter);
        String[] foldersInDirectory = new String[directoryListAsFile.length];
        int counter = 0;
        for (File directoryAsFile : directoryListAsFile) {

            foldersInDirectory[counter] = directoryAsFile.getName();
            counter++;
        }

        return foldersInDirectory;
    }

    public static void makeDirs(String[] folders) {
//          String filePath = "\\\\ueahome4\\stusci1\\yqm15fqu\\data\\NTProfile\\
    Desktop\\temp test\\machinelearning\\testing\\Results\\1NN\\";
        String filePath = " \\\\ueahome4\\stusci1\\yqm15fqu\\data\\NTProfile\\
           Desktop\\LAST REPO\\machinelearning\\testing\\Ensemble1\\";

        for (int i = 0; i < folders.length; i++) {
            new File(filePath + folders[i]).mkdirs();

        }

    }

    public static void changeNames(String[] folders) throws IOException {

        for (int j = 0; j < folders.length; j++) {

            File folder = new File("\\\\ueahome4\\stusci1\\yqm15fqu\\data\\
               NTProfile\\Desktop\\temp test\\machinelearning\\testing\\Ensemble
               \\" + folders[j] + "\\");
```

```java
            File[] listOfFiles = folder.listFiles();

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

                if (listOfFiles[i].isFile()) {

                    File f = new File("\\\\ueahome4\\stusci1\\yqm15fqu\\data\\
                        NTProfile\\Desktop\\temp test\\machinelearning\\testing\\
                        Ensemble\\" + folders[j] + "\\" + listOfFiles[i].getName
                        ());

                    f.renameTo(new File(folder + "\\" + "testFold" + i + ".csv"))
                        ;
                }
            }
            System.out.println("conversion is done");
        }

    }

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

        String[] datasets = findFoldersInDirectory("\\\\ueahome4\\stusci1\\
            yqm15fqu\\data\\NTProfile\\Desktop\\temp test\\machinelearning\\
            testing\\REAL TEST 1\\");

        String[] classifiers = findFoldersInDirectory("\\\\ueahome4\\stusci1\\
            yqm15fqu\\data\\NTProfile\\Desktop\\temp test\\machinelearning\\
            testing\\Results\\");

        for (int i = 0; i < datasets.length; i++) {
            datasets[i] = (String) datasets[i].subSequence(0, datasets[i].length
                () - 5);
            System.out.print(datasets[i]+"\n");
        }
//          RenameFileDirectory(datasets);
//          makeDirs(datasets);
//
//          changeNames(datasets);
//
//          String[] datasets = {"1NNbank", "1NNblood", "1NNbreast-cancer-wisc-diag
    ",
//              "1NNbreast-tissue", "1NN"};
////          String[] classifiers = new String[]{"1NN"};


        MultipleClassifierEvaluation mC = new MultipleClassifierEvaluation("\\\\
            ueahome4\\stusci1\\yqm15fqu\\data\\NTProfile\\Desktop\\temp test\\
            machinelearning\\testing\\Anlysis\\", "FourthAnal", 30);
        mC.setTestResultsOnly(true);
//              setTestResultsOnly(false).
        mC.setBuildMatlabDiagrams(true);
//              setBuildMatlabDiagrams(false).
        mC.setUseAccuracyOnly();

        mC.setDatasets(datasets);
        mC.readInClassifiers(classifiers, "//ueahome4/stusci1/yqm15fqu/data/
            NTProfile/Desktop/temp test/machinelearning/testing/Results/");
        mC.runComparison();
    }

}
```

InstanceContainer.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package machinelearning_coursework;

import weka.core.Instance;

/**
 *
 * @author yqm15fqu
 * @param <Instance>
 * @param <distance>
 */
public class InstanceContainer implements Comparable<InstanceContainer>{

    private Instance instance;
    private double distance;

    public InstanceContainer(Instance inst, double dist) {
        this.instance = inst;
        this.distance = dist;
    }

    public Instance getInstance(){
        return this.instance;
    }

    public double getDistance(){
        return this.distance;
    }

    @Override
    public int compareTo(InstanceContainer inst) {
            if(this.distance < inst.distance){
                return -1;
            }else if(inst.distance < this.distance){
                return 1;
            }
            return 0;
    }








}
```

ml-cswk2-100135292-file.java

Filename scrubbed (one or more forbidden characters found). Original name:

MachineLearning_Coursework.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package machinelearning_coursework;

import evaluation.ClassifierResultsAnalysis;
import evaluation.storage.ClassifierResults;
import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import static machinelearning_coursework.KNN.accuracy;
import static machinelearning_coursework.KnnEnsemble.accuracyEns;
import static machinelearning_coursework.WekaTools.*;
import static machinelearning_coursework.testing.*;
import weka.classifiers.Classifier;
import weka.classifiers.bayes.NaiveBayes;
import weka.core.Instance;
import weka.core.Instances;
import static weka.core.Utils.*;

/**
 *
 * @author yqm15fqu
 */
public class MachineLearning_Coursework {

    /**
     * @param args the command line arguments
     */
    public static void writeResult(Instances test, Classifier c, StringBuilder
       result, PrintWriter writer) throws Exception {
        result.append("No Parameter Info").append('\n');
        result.append(accuracy(c, test)).append('\n'); //The overallAccuracy Put
            it into a method. Build then call this.

        for (Instance instance : test) {

            double[] dist = c.distributionForInstance(instance);
            String distStr = "";
            for (int i = 0; i < dist.length; i++) {
                distStr += "," + dist[i];
            }
            result.append(c.classifyInstance(instance)).append(",").append(
               instance.classValue()).append(',').append(distStr).append('\n');
        }
        writer.write(result.toString());
        writer.close();
    }

    public static void writeResultEnsemble(Instances test, KnnEnsemble c,
       StringBuilder result, PrintWriter writer) throws Exception {
        result.append("No Parameter Info").append('\n');
```

```java
        result.append(accuracyEns(c, test)).append('\n'); //The overallAccuracy Put it into a method. Build then call this.

        for (Instance instance : test) {

            double instClassVal = c.classifyInstanceEns(instance);
            double[] dist = c.getEnsemblePredictions();
            String distStr = "";
            for (int i = 0; i < dist.length; i++) {
                distStr += "," + dist[i];
            }
            result.append(instance.classValue()).append(",").append(instClassVal).append(',').append(distStr).append('\n');
        }
        writer.write(result.toString());
        writer.close();
    }

    public static void main(String[] args) throws Exception {
//        Instances spamTrain1 = loadClassificationData("//ueahome4/stusci1/yqm15fqu/data/NTProfile/Desktop/Machine Learning/optical/optical_TRAIN.arff");
//        Instances spamTest1 = loadClassificationData("//ueahome4/stusci1/yqm15fqu/data/NTProfile/Desktop/Machine Learning/optical/optical_TEST.arff");
//
//        KNN knN = new KNN();
//        knN.buildClassifier(spamTrain1);
//        knN.setK(5);
//        System.out.println(accuracy(knN, spamTest1));
//        Instances spamTrain = loadClassificationData("//ueahome4/stusci1/yqm15fqu/data/NTProfile/Desktop/Machine Learning/statlog-vehicle/statlog-vehicle_TRAIN.arff");
//        Instances spamTest = loadClassificationData("//ueahome4/stusci1/yqm15fqu/data/NTProfile/Desktop/Machine Learning/statlog-vehicle/statlog-vehicle_TEST.arff");
//        File folder = new File("C:/Users/Rob/Desktop/Machine Learning Cw/machinelearning/ArffFiles");

        File folder = new File("\\\\ueahome4\\stusci1\\yqm15fqu\\data\\NTProfile\\Desktop\\temp test\\machinelearning\\ArffFilesFinal");
        String[] folderNames = listFilesForFolder(folder);

        for (int i = 0; i < folderNames.length; i++) {
            Instances wholeInst = loadClassificationData("\\\\ueahome4\\stusci1\\yqm15fqu\\data\\NTProfile\\Desktop\\temp test\\machinelearning\\ArffFilesFinal\\" + folderNames[i]);

            ///////////////////////////////////////////////////FOR 3.1//////////////////////////////////////////////////////////////
            for (int j = 0; j < 30; j++) {
                Instances[] split = splitData(wholeInst, 0.7);//Splits data 70-30
                Instances spamTrain = split[1];
                Instances spamTest = split[0];

                startTest(spamTrain, spamTest, folderNames[i], j);
                startTest1(spamTrain, spamTest, folderNames[i], j);
                startTest2(spamTrain, spamTest, folderNames[i], j);
                startTest3(spamTrain, spamTest, folderNames[i], j);

            }
            ///////////////////////////////////////////////////FOR 3.2//////////////////////////////////////////////////////////////
            for (int j = 0; j < 30; j++) {
                Instances[] split = splitData(wholeInst, 0.7);//Splits data 70-30
                Instances spamTrain = split[1];
                Instances spamTest = split[0];

                startTest(spamTrain, spamTest, folderNames[i], j);
                startEnsemble(spamTrain, spamTest, folderNames[i], j);

            }
            /////////////////////////////////////////////////FOR
               3.3//////////////////////////////////////////////////////////////
            for (int j = 0; j < 30; j++) {
                Instances[] split = splitData(wholeInst, 0.7);//Splits data 70-30
                Instances spamTrain = split[1];
                Instances spamTest = split[0];

                //1NN
                startTest(spamTrain, spamTest, folderNames[i], j);
                //Ensemble
                startEnsemble(spamTrain, spamTest, folderNames[i], j);
                //Naive Bayes
                startTest5(spamTrain, spamTest, folderNames[i], j);
                //Random Forest
                startTest6(spamTrain, spamTest, folderNames[i], j);
                //Rotation Forest
                startTest7(spamTrain, spamTest, folderNames[i], j);

            }
        }

        int[] realVals = {0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0,
           1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1,
           0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1,
           0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
           0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
           0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0,
           0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
           0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1,
           0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0,
           0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0,
           1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,
           0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
           0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
           0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1,
           0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0,
           0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0,
           1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
           0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
           0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1,
           1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0,
           0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
           0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,
           1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0,
           0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0,
           0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1,
           0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
           1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1,
           1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
           0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0,
           1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1,
           1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
           0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
           0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
           0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
           0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
           1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1,
           0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1,
           1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0,
           0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0,
           0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
           0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
           1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0,
           0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0,
           0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1};

        int[] predictedVals = {0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1,
            0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1,
            1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0,
            1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
            0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
            0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
            0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
            0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
            1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1,
            0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1,
            0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
            1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,
            0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
            1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1,
            1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1,
            0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0,
            1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0,
            1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1,
            1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0,
            1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1,
            0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
            0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
            0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1,
            0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
            0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
            0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
            1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1,
            1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
            1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0,
            0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1,
            1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
            1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
            1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0,
            1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0,
            0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1,
            1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
            1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1,
            1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1,
            0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
            1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1,
            0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
            0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1,
            0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0,
            0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1};

        findStats(realVals, predictedVals);

    }
```

```
}
```

testing.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package machinelearning_coursework;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.util.Arrays;
import java.util.Random;
import static machinelearning_coursework.MachineLearning_Coursework.writeResult;
import static machinelearning_coursework.MachineLearning_Coursework.
    writeResultEnsemble;
import static machinelearning_coursework.WekaTools.confusionMatrix;
import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.meta.RotationForest;
import weka.classifiers.trees.RandomForest;
import weka.core.Instances;
import weka.core.PropertyPath.Path;

/**
 *
 * @author yqm15fqu
 */
public class testing {

    public static void startTest(Instances train, Instances test, String probName
        , int foldNo) throws FileNotFoundException, Exception {
        String filePath = "\\\\ueahome4\\stusci1\\yqm15fqu\\data\\NTProfile\\
           Desktop\\temp test\\machinelearning\\testing\\";
        String x = probName.substring(0, probName.length() - 5);
        KNN kNN = new KNN();

        kNN.setStandardise(false);
        StringBuilder result = new StringBuilder();

        if (!new File(filePath + "1NN").exists()) {
            new File(filePath + "1NN").mkdirs();
        }

        if (!new File(filePath + "1NN\\" + probName + "").exists()) {
            new File(filePath + "1NN\\" + probName + "").mkdirs();
        }

        try (PrintWriter writer = new PrintWriter(new File(filePath + "\\1NN\\" +
            x + "\\testFold" + foldNo + ".csv"))) {
            kNN.buildClassifier(train);
            result.append(probName).append(", 1NN").append('\n');
            writeResult(test, kNN, result, writer);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

        public static void startTest1(Instances train, Instances test, String
           probName, int foldNo) throws FileNotFoundException, Exception {
        String filePath = "\\\\ueahome4\\stusci1\\yqm15fqu\\data\\NTProfile\\
           Desktop\\temp test\\machinelearning\\testing\\";
        String x = probName.substring(0, probName.length() - 5);
        KNN kNN = new KNN();

        kNN.setStandardise(true);
        StringBuilder result = new StringBuilder();

        if (!new File(filePath + "Standardise").exists()) {
            new File(filePath + "Standardise").mkdirs();
        }

        if (!new File(filePath + "Standardise\\" + probName + "").exists()) {
            new File(filePath + "Standardise\\" + probName + "").mkdirs();
        }

        try (PrintWriter writer = new PrintWriter(new File(filePath + "\\
           Standardise\\" + x + "\\testFold" + foldNo + ".csv"))) {
            kNN.buildClassifier(train);
            result.append(probName).append(", Standardise").append('\n');
            writeResult(test, kNN, result, writer);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
            public static void startTest2(Instances train, Instances test, String
                probName, int foldNo) throws FileNotFoundException, Exception {
        String filePath = "\\\\ueahome4\\stusci1\\yqm15fqu\\data\\NTProfile\\
           Desktop\\temp test\\machinelearning\\testing\\";
        String x = probName.substring(0, probName.length() - 5);
        KNN kNN = new KNN();

        kNN.setStandardise(false);
        kNN.setLeaveOneOut(true);
        StringBuilder result = new StringBuilder();

        if (!new File(filePath + "LOOCV").exists()) {
            new File(filePath + "LOOCV").mkdirs();
        }

        if (!new File(filePath + "LOOCV\\" + probName + "").exists()) {
            new File(filePath + "LOOCV\\" + probName + "").mkdirs();
        }

        try (PrintWriter writer = new PrintWriter(new File(filePath + "\\LOOCV\\"
            + x + "\\testFold" + foldNo + ".csv"))) {
            kNN.buildClassifier(train);
            result.append(probName).append(", LOOCV").append('\n');
            writeResult(test, kNN, result, writer);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }     public static void startTest3(Instances train, Instances test, String
        probName, int foldNo) throws FileNotFoundException, Exception {
        String filePath = "\\\\ueahome4\\stusci1\\yqm15fqu\\data\\NTProfile\\
           Desktop\\temp test\\machinelearning\\testing\\";
        String x = probName.substring(0, probName.length() - 5);
        KNN kNN = new KNN();

        kNN.setStandardise(false);
        kNN.setWeightedVoting(true);
        StringBuilder result = new StringBuilder();

        if (!new File(filePath + "Weighted").exists()) {
            new File(filePath + "Weighted").mkdirs();
```

```java
        }

        if (!new File(filePath + "Weighted\\" + probName + "").exists()) {
            new File(filePath + "Weighted\\" + probName + "").mkdirs();
        }

        try (PrintWriter writer = new PrintWriter(new File(filePath + "\\Weighted
           \\" + x + "\\testFold" + foldNo + ".csv"))) {
            kNN.buildClassifier(train);
            result.append(probName).append(", Weighted").append('\n');
            writeResult(test, kNN, result, writer);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
        public static void startEnsemble(Instances train, Instances test, String
           probName, int foldNo)throws FileNotFoundException, Exception {
//          String filePath = "\\\\ueahome4\\stusci1\\yqm15fqu\\data\\NTProfile\\
    Desktop\\temp test\\machinelearning\\testing\\";
        String filePath = "C:\\Users\\Rob\\Desktop\\Machine Learning Cw\\
           machinelearning\\testing\\Ensemble\\";
        String x = probName.substring(0, probName.length() - 5);
        KnnEnsemble kNN = new KnnEnsemble();

        StringBuilder result = new StringBuilder();

        if (!new File(filePath + probName + "").exists()) {
            new File(filePath + probName + "").mkdirs();
        }

        try (PrintWriter writer = new PrintWriter(new File(filePath + probName +
           "\\Ensemble" + x + "Fold" + foldNo + ".csv"))) {
            kNN.buildEnsemble(train);
            result.append(probName).append(", Ensemble").append('\n');
            writeResultEnsemble(test, kNN, result, writer);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void startTest5(Instances train, Instances test, String
        probName, int foldNo) throws FileNotFoundException, Exception {
        String filePath = "\\\\ueahome4\\stusci1\\yqm15fqu\\data\\NTProfile\\
           Desktop\\temp test\\machinelearning\\testing\\Results\\NaiveBayes\\
           Predictions\\";
        String x = probName.substring(0, probName.length() - 5);
        NaiveBayes nBayes = new NaiveBayes();
        StringBuilder result = new StringBuilder();

        if (!new File(filePath + probName + "").exists()) {
            new File(filePath + x + "").mkdirs();
        }

        try (PrintWriter writer = new PrintWriter(new File(filePath + x + "\\" +
           "testFold" + foldNo + ".csv"))) {
            nBayes.buildClassifier(train);
            result.append(probName).append(", naiveBayes").append('\n');
            writeResult(test, nBayes, result, writer);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static void startTest6(Instances train, Instances test, String
        probName, int foldNo) throws FileNotFoundException, Exception {
        String filePath = "\\\\ueahome4\\stusci1\\yqm15fqu\\data\\NTProfile\\
            Desktop\\temp test\\machinelearning\\testing\\Results\\RandomForest\\
            Predictions\\";
        String x = probName.substring(0, probName.length() - 5);
        Random rand = new Random();
        RandomForest randFor = new RandomForest();
        randFor.setSeed(rand.nextInt());
        StringBuilder result = new StringBuilder();

        if (!new File(filePath + probName + "").exists()) {
            new File(filePath + x + "").mkdirs();
        }

        try (PrintWriter writer = new PrintWriter(new File(filePath + x + "\\" +
            "testFold" + foldNo + ".csv"))) {
            randFor.buildClassifier(train);
            result.append(probName).append(", RandomForest").append('\n');
            writeResult(test, randFor, result, writer);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void startTest7(Instances train, Instances test, String
        probName, int foldNo) throws FileNotFoundException, Exception {
        String filePath = "\\\\ueahome4\\stusci1\\yqm15fqu\\data\\NTProfile\\
            Desktop\\temp test\\machinelearning\\testing\\Results\\RotationForest
            \\Predictions\\";
        String x = probName.substring(0, probName.length() - 5);
        RotationForest rotFor = new RotationForest();
        StringBuilder result = new StringBuilder();

        if (!new File(filePath + probName + "").exists()) {
            new File(filePath + x + "").mkdirs();
        }

        try (PrintWriter writer = new PrintWriter(new File(filePath + x + "\\" +
            "testFold" + foldNo + ".csv"))) {
            rotFor.buildClassifier(train);
            result.append(probName).append(", RotationForest").append('\n');
            writeResult(test, rotFor, result, writer);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void findStats(int[] actual, int[] predicted) {

        int[] act = {1};
        int[] pre = {1};

        int[][] answer = confusionMatrix(predicted, actual);

        for (int i = 0; i < answer.length; i++) {
            for (int j = 0; j < answer.length; j++) {
                System.out.println(answer[i][j] + "\t");
            }
            System.out.println("\n");
        }
        System.out.println(Arrays.deepToString(answer));

        double a = answer[0][0];
        double b = answer[0][1];
        double c = answer[1][0];
        double d = answer[1][1];

        double TPR = a / (a + c);
        double FPR = b / (b + d);
        double FNR = c / (a + c);
        double TNR = d / (b + d);

        System.out.println("TPR: " + TPR + " FPR: " + FPR + " FNR: " + FNR + "
            TNR: " + TNR);
    }

}
```

WekaTools.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package machinelearning_coursework;

import java.io.File;
import java.io.FileReader;
import java.util.Arrays;
import weka.classifiers.Classifier;
import weka.core.Debug.Random;
import weka.core.Instance;
import weka.core.Instances;

/**
 *
 * @author yqm15fqu
 */
public class WekaTools {

    public static double measureAccuracy(Classifier c, Instances test) throws
        Exception {
        int counter = 0;

        for (Instance instance : test) {
            if (c.classifyInstance(instance) == instance.classValue()) {
                counter++;
            }
        }

//          System.out.println("Correctness: " + counter);
        double numInst = test.numInstances();

        System.out.println("Accuracy: " + ((double) counter / numInst) * 100);

        double accuracy = ((counter / numInst) * 100);

        return accuracy;
    }

    public static Instances loadClassificationData(String fullPath) {
        String dataLocation = fullPath;

        Instances train = null;
        try {
            FileReader reader = new FileReader(dataLocation);
            train = new Instances(reader);
        } catch (Exception e) {
            System.out.println("Exception caught: " + e);
        }
        train.setClassIndex(train.numAttributes() - 1);
        return train;
    }

    public static Instances[] splitData(Instances all, double proportion) {
        Instances[] split = new Instances[2];
        Random random = new Random();
        split[0] = new Instances(all);
        split[1] = new Instances(all, 0);
        split[0].randomize(random);

        int splitAmount = (int) Math.floor((proportion * all.numInstances()));

        for (int i = 0; i < splitAmount; i++) {
            split[1].add(split[0].remove(0));
        }
//          System.out.println(split[0].size());
//          System.out.println(split[1].size());
        return split;
    }

    public static double[] classDistribution(Instances data) {
        double[] temp = new double[data.numClasses()];
        for (int i = 0; i < data.numInstances(); i++) {
            temp[(int) data.get(i).classValue()] += 1;
        }

        for (int i = 0; i < temp.length; i++) {
            temp[i] = temp[i] / data.numInstances();
        }
        return temp;
    }

    public static void main(String[] args) throws Exception {
        Instances train = loadClassificationData("//ueahome4/stusci1/yqm15fqu/
            data/Documents/MachineLearning_lab_2new/src/machinelearning_lab_2new/
            Arsenal_TRAIN.arff");

        double[] test = classDistribution(train);
        System.out.println(Arrays.toString(test));
    }

    public static int[][] confusionMatrix(int[] predicted, int[] actual) {
        int[][] confMatrix = new int[2][2];
        for (int i = 0; i < predicted.length; i++) {
            if (predicted[i] == 0) {
                if (actual[i] == predicted[i]) {
                    confMatrix[0][0] += 1;
                } else {
                    confMatrix[0][1] += 1;
                }
            } else if (actual[i] == predicted[i]) {
                confMatrix[1][1] += 1;
            } else {
                confMatrix[1][0] += 1;
            }
        }
        return confMatrix;
    }

    public static String[] listFilesForFolder(File folder) {
        File[] files = folder.listFiles();
        String[] bla = new String[files.length];
        int as = 0;
        for (final File fileEntry : folder.listFiles()) {
            if (fileEntry.isDirectory()) {
                listFilesForFolder(fileEntry);
            } else {
                bla[as] = fileEntry.getName();
                as++;
                System.out.println(fileEntry.getName());
            }
        }
        return bla;
    }

    public static int[] classifyInstances(Classifier c, Instances test) throws
       Exception {
        int[] r = new int[test.size()];
        for (int i = 0; i < test.size(); i++) {
            r[i] = (int) c.classifyInstance(test.get(i));
        }
        return r;
    }

    public static int[] getClassValues(Instances data) {
        int[] classValues = new int[data.size()];
        for (int i = 0; i < data.size(); i++) {
            classValues[i] = (int) data.get(i).classValue();
        }

        return classValues;
    }

}
```

An experimental assessment of nearest neighbour classifiers

Robert Levett[1]

[1]Department of Computer Science, University of East Anglia, UK
[2]School of Computing Sciences (CMP), University of East Anglia, UK
yqm15fqu@uea.ac.uk

Abstract

Evaluation of Nearest Neighbour Classifiers.

Machine learning has become ever more popular over the past 2 decades and with this the development of classification algorithms to be able to accurately predict the class of an object, with data that has been captured. Many complex classification algorithms have arisen throughout the years such as Rotation Forest [1], Random Forest [2], Support Vector Machines (SVM's) and many more. But there are also some not so complex algorithms, such as the classification algorithm which will be discussed in this paper, the KNN classifier.

This poses the question: is it possible to achieve an acceptably high classification accuracy with a KNN classifier? and which implementations affect the accuracy the greatest?

Although the KNN classifier is easy in theory to understand, there are many different methods that can be implemented in an attempt to improve the accuracy overall. In this paper, I will be discussing the implementation of Leave One Out Cross Validation (LOOCV), Standardisation of attributes, Weighted voting and Ensembles. These will be compared to other conventional classifiers to see how they compare. The biggest impact on accuracy was LOOCV, and the smallest impact on accuracy was Weight. This is likely as the LOOCV picks a pre-determined value of K that provided the highest degree of accuracy based on the training data (at the cost of significantly increased processing time.). Weighting the votes likely has a less noticeable impact as there are certain use cases where the vote would change the outcome of the class value being picked.

1 Introduction

A Nearest Neighbour Classifier (KNN) is a classification algorithm that is simple to understand, easy to implement and achieves a consistently high-performance [3]. A Nearest Neighbour classifier is a classification algorithm which chooses the class of the instance being passed based on it's closest neighbours (usually based on Euclidean distance). While the capabilities of KNN classifiers are well documented, there are many different ways which have been suggested in recent years which provide solutions that can be used to further enhance the performance of the classifier (although it should be noted that usually a caveat of additional features to improve performance is greatly increased processing time). Given that there is a vast range of academic and research papers that have been published online proposing different methods that can be implemented in an attempt to improve the performance of the KNN classifier, only the following changes will be implemented, tested and discussed in this paper:

- Leave One Out Cross Validation (LOOCV).
- Weighted Voting.
- Standardisation of attributes.
- Ensembles.

Current assumptions made about the advanced features are that LOOCV will have the greatest impact on the accuracy overall. Although LOOCV will increase the processing time considerably (especially noticeable on larger data sets), one of the more suitable value for K is automatically found and set for the classifier. Another assumption is that Ensembles will produce a higher degree of accuracy as it is the average vote of many KNN classifiers. [4] states that "for at least one arbitrary chosen example of a small training set, a lower probability of misclassification was obtained". From this, it's assumed that the impact of the weighted voting will be minor, but noticeable. [5] states "An important finding during the performance evaluation of k-nearest neighbour was that feature standardization improved accuracy for some data sets and did not reduce accuracy" which would imply that Standardisation of data should increase accuracy overall, but not greatly. Rotation Forest will be the classifier with the highest accuracy overall from all classifiers tested.

There are many classifiers that are much more complex in implementation and understanding, that may provide an improvement in the performance of KNN classifier. A comparison between KNN, Random Forest, Rotation Forest, Ensemble of KNN and Naive Bayes will be carried out and evaluated in an attempt to see if KNN can out-perform some more complex algorithms.

2 Data Description

Data sets used were part of the "UCIContinuous" machine learning repository [6], as all data sets available there are specialised for use in machine learning. Table 2 describes all of the data sets in detail, however, a case study was done into the data set labelled "spambase". [6] states on the page for spambase:

"The "spam" concept is diverse: advertisements for products/web sites, make money fast schemes, chain letters, pornography...

Our collection of spam e-mails came from our postmaster and individuals who had filed spam. Our collection of non-spam e-mails came from filed work and personal e-mails, and hence the word 'george' and the area code '650' are indicators of non-spam... The last column of 'spambase.data' denotes whether the e-mail was considered spam (1) or not (0),".

Now that the context of the data set has been determined, the data collected was used in an attempt to make an e-mail spam filter to exclude unwanted emails. To do this, an attempt to determine if any incoming mail was spam mail, determined by a class value of 1, or non-spam mail, determined by a class value of 0. From the table we are also able to see that there are 4601 instances, of which 2788 were not spam mail and 1813 were spam mail, there are 57 attributes per instance, and the last attribute is the class value of the instance. The problem of spam email detection has been well documented in the machine learning community over the last 20 years, and there are many examples of attempts to train a classifier so that it can detect spam emails. One such example is noted when [7] et al attempted to train a Naive Bayes classifier in an attempt to make spam mail detection automated. The findings of the experiment were that although using a Naive Bayes proved to not provide a high enough accuracy when used over n=999 emails, it provided significant improvements over a keyword/keyphrase based spam filtration system. It is noted that the only classifier used in this situation was Naive Bayes, which will be included in experiments for the spambase data set.

2.1 Classifier Description

When approaching the development of the KNN classifier that would form the base of any experiments undertaken throughout this paper, decisions had to be made to ensure that the classifier was implemented correctly and robust. Some decisions made include:

- Euclidean distance only

  There are two global variables used, one which stores the distances of n-1 instances to the current instance being classified into an array, which is used for the LOOCV method to be able to store all the distances from each instance to every other. It is possible to use other distance metrics if it was required, however, it was decided that in this particular situation that Euclidean would be sufficient.

- Global Variables for distances and instances to increase ease of passing information between methods.
  Another global variable which is an ArrayList of the K lowest instance objects in regard to the current instance being classified; this was chosen as a simplistic solution to passing the K closest instances from the classifyInstance method to the distributionForInstance method.

- Use of the ArrayList data structure.
  ArrayLists were used extensively as they provide convenient storage of objects (such as instances) which can then be passed between methods/classes. The flexibility of the ArrayList proved particularly useful in the implementation of the KNN classifier as it's dynamic size allowed for more generic and versatile code, as allocation of ArrayList size was not always required.

- Creation of a container class that pairs a distance with its corresponding instance.
  A custom container class (named InstanceContainer) was implemented, which simply pairs an Instance object with it's corresponding distance to the current object being classified. The use of the comparable interface for this container was required as it enabled the sorting of instances objects based on their Euclidean distance to the current object being classified. Once created and stored into an ArrayList it was possible to sort

the InstanceContainer objects into ascending order based on distances, then it becomes possible to simply select the first K instances from an ArrayList of InstanceContainer objects to find the K closest neighbours. (Saving time on having to reclassify instances for different values of K.)

- Standardisation of attributes

$$Z = \frac{x - \overline{x}}{\sigma} \tag{1}$$

  Standardisation of attributes was implemented for the KNN as it transforms all data to have zero mean and a standard deviation of 1. KNN classifiers are sensitive to attribute scaling, so standardising the attributes helps to prevent this.

- Leave One Out Cross Validation (LOOCV) Leave one out cross validation was implemented for the KNN. The theory behind LOOCV is that every instance in the training set is used as the test instance once, and is classified against the remaining n-1 instances. The accuracy for K = 1 to the maxK value is then recorded for each instance being tested, then an average of all K values across all the data is recorded, and the K value which provided the highest average accuracy is chosen. This can ensure that a value for K which is most appropriate (or at most, more appropriate) will be picked.

- Weighted Voting

$$\frac{1}{1 + d(x, y)} \tag{2}$$

  Weighted voting can be applied to the weight variable of an instance (default to 1.). The formula ensures that values with a lower euclidean distance receive a higher weight when voting for which class an instance would be classified as. In certain situations, this can change the classification result from one class to another, as the sum of each weight per class is calculated then the highest value is chosen.

- Helper methods Some helper methods were implemented as it was found that a particular operation (such as finding the highest value in an array) was needed many times across development. Methods used are: accuracy, mode (returns most common number) and findHighestPos (finds the highest position within an array).

When the KNN is first created, it has LOOCV and weighted voting disabled by default, however, standardisation of attributes is enabled. Accessor methods are found in the KNN class that can be used to enable LOOCV and weighted voting (via methods setLOOCV and setWeightedVoting respectively). Once all methods have been enabled or disabled, it is possible to simply pass the training data into the buildClassifier method and the KNN will be ready to classify instances using the classifyInstance method (Building a classifier with LOOCV enabled can greatly increase the build time if the data set used is large in size.).

Other classifiers also needed to be implemented in order to perform experiments on the performance of the KNN classifier. The choice of classifiers picked were: Random Forest[8], Rotation Forest[1], 1NN(1 nearest neighbour), Naive Bayes, and an Ensemble of KNN classifiers with all methods enabled.

2.2 Results

For experiments it was decided that each classifier would be run against all of the data sets found in 2. Each classifier has its accuracy recorded across all data sets, and each data set was recorded 30 times (30 fold cross-validation). The external classifiers used were also tested in this way to keep results consistent and comparable, which means all tests were undertaken on the same training/testing data splits for each classifier being compared.

2.3 Procedure

The data was read in one data set at a time, it was then randomly shuffled using the Collections interface and split into a 70/30 training/test distribution respectively. Once loaded and split, the data was passed to each classifier so that they would be classifying the same test instances with the same training instances, allowing for true comparison between each classifier. The classifications were recorded for each test instance from each classifier, then compared to the actual class values so that statistics such as True Positive, True Negative, False Positive and False Negatives could be recorded alongside the accuracy of the classifier. This was repeated 30 times for each data set, and the results were averaged.

2.4 is it possible to achieve a high classification accuracy with a KNN classifier?

Once the implementation of the KNN classifier and the methods designed to improve accuracy (LOOCV, standardisation, weighted voting) was completed, testing was to be done into the impact each method had on the 1NN base classifier. 4 shows the critical difference between each method and the base 1NN classifier. From the picture, it is obvious to see that the improvements implemented so far do not increase the accuracy over the base 1NN classifier when averaged across all data sets.

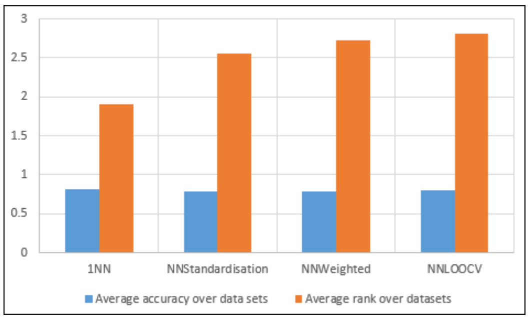

Figure 1: Average accuracy and ranks.

Shows the average accuracy of each classifier across all the problems, and the average of each classifier over the data set. This shows that standardisation of attributes improved the classifier greater than the other methods, and gave it a higher rank overall, but still no improvement over the base 1NN.

2.5 LOOCV will have the greatest impact on the accuracy overall.

As mentioned before, 1 shows the accuracy of the classifier improvements overall data sets and folds, and shows that the LOOCV had the lowest average rank overall, however, had the second highest accuracy of 80%. This is likely because there were certain data sets which the LOOCV did not perform well on, which becomes apparent when looking at Fig 7. LOOCV performed well on certain data sets (such as blood, post operative), but slightly underperformed on others.

Although LOOCV produced the second highest accuracy, the program took ¿20 hours to run across all 35 data sets and all 30 folds, which is a significant amount of time if under timing constraints.

2.6 Weighted voting will have a minor, but noticeable effect on accuracy.

4 shows that weight performed poorly for the implementation over the other metrics. It is possible that this is due to incorrect implementation of weighing in during development, or that the weighting function actually caused some miss-classifications throughout the data sets, leading to lower overall accuracy. This result is contrary to research found online, such as from [4] et al, which leads to the assumption that the implementation was not understood correctly, or was implemented within the wrong sections.

2.7 Standardisation of data should increase accuracy overall, but not great.

Although standardisation of attributes actually was worse in accuracy than the 1NN, it proved to have the greatest impact on average rank across all the data sets as shown in 4. This is likely because when the standardisation was applied to the attributes it gives zero mean and unit variance attribute values, which can improve the classifier as KNN classifiers are sensitive to attribute scaling. Standardisation also made the classifier achieved the third highest accuracy with 79.2% overall.

2.8 Ensembles will have higher accuracy than 1NN.

The KNN Ensemble proved to have a higher classifier rank overall when compared to the 1NN, as shown in Fig5. Fig **??** shows that for different data sets that the ensemble does not perform as well as the 1NN. Particularly in the image segmentation data set, the 1NN across all experiments and classifiers performs with much higher accuracy than all other classifiers.

Ensembles have an average accuracy of 80.3%, and the 1NN had an average accuracy of 80.9%, making the margin between both classifiers small. This likely indicates that the KNN Ensemble is the better classifier of the two, but with the caveat of greatly increased processing time due to all the KNN classifiers in the Ensemble having weighted voting, standardisation and LOOCV were all enabled. From Fig 8 it is possible to see that the Ensemble performed better/the same on as many as 22/35 data sets, leading to the assumption that for certain use cases it does not work well, but in the majority of data sets in our sample it works well.

2.9 Rotation Forest will be the classifier with the highest accuracy overall.

Research (such as by [1] et al) suggests that Rotation forest is a classification algorithm which can consistently outperform classifiers. A test was set where Rotation Forest, Random Forest, 1NN, Ensemble of KNN classifiers and a Naive Bayes classifier were all used to classify the same test data, using the same training data, across 30 fold cross-validation. The results were recorded (See Fig 9), and accuracy/rank are shown in Fig10. From Fig10 it's possible to see that Rotation forest achieved an average accuracy of 86%, Random Forest 85%, Ensemble 80%, 1NN 81% and Naive Bayes 73%. The outcome of the experiment here that Rotation Forest performed the best holds true to the conclusion of many different research articles found online and discussed in this paper. Fig 9 shows the crit-

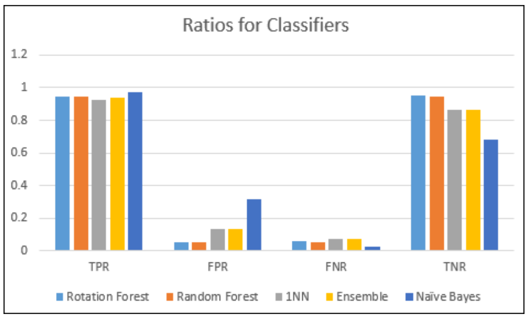

Figure 2: True Positive, False Positive, False Negative, True Negative.

ical difference diagram of the same experiment, where it becomes apparent that Rotation Forest is the highest performer overall data sets/folds, with Random Forest producing close results.

2.10 Naive Bayes will under-perform.

[9] states " These probabilistic approaches make strong assumptions about how the data is generated, and posit a probabilistic model that embodies these assumptions; then they use a collection of labeled training examples to estimate the parameters of the generative model. Classication on new examples is performed with Bayes rule by selecting the class that is most likely to have generated the example.".

Because Naive Bayes works by making such heavy assumptions about data, it was hypothesised that it would not perform well across all the data sets that were used in our sample of data as there is no way to ensure that the data fits the specific use case that Naive Bayes may perform well in. From 9 it is clear that throughout many of the data sets the Naive Bayes classifier performed poorly, often being the worst of the 5 classifiers used. 6 Also shows that Naive Bayes was ranked as 4.3/5 across all folds/data sets for classifier on accuracy.

2.11 Case Study Results

For the case study, it was decided that 5 classifiers would be used to classify the spambase data set. It was decided that the classifiers which would be used would be : Naive Bayes, 1NN, Ensemble of KNN, Rotation Forest and Random Forest. Table 1 shows the values used for:

From Fig 2 it's possible to deduct that Naive Bayes has the highest rate of false positives, alongside the lowest rate of true negative classifications, making it an unrliable solution for the spambase data set. Rotation forest has the second highest True Positive Ratio, and the highest True Negative Ratio. Random Forest is closely drawn with Rotation Forest, but falls short with the True Negative Ratio. 1NN is a consistent under performing classifier for the spambase data set as has lower TPR, higher FPR, FNR, and lower TNR, making it the worst of all the classifiers for the spambase data set. Ensembles also under perform, likely as the ensemble is just made of KNN classifiers, thus giving it the same disadvantages as 1NN.

Rotation Forest would be chosen as the classifier for the spambase data set, as it consistently produces high true positive and true negative ratios, although it should be noted that Random Forest would also be a viable choice, as even though it has a slightly lower true negative ratio than Rotation Forest, it also has a smallest False Negative Ratio. From Fig10 it is also possible to see that Rotation Forest produced an overall average accuracy of 85.7% across all data sets and folds, whereas Rotation Forest achieved 85%, Ensembles 80.3%, 1NN 80.9% and Naive Bayes 73.3%, Making Rotation Forest also the most accurate across all data sets and folds on average.

2.12 Conclusions

After running experiments and gathering all the data needed, then analysing the data, several conclusions can be made about the original hypothesis put forward. One such example of this is the hypothesis that the LOOCV would be the highest performer on accuracy of all methods implemented. This turned out to be the case in some sense of the words, of the implemented methods the LOOCV had the highest accuracy (second to 1NN). However as Fig 4 shows, it was actually ranked as 3rd of the 4 methods implemented, which is without inclusion of time taken for the method to actually run.

Standardisation of attributes proved to be the second highest method on accuracy, and ranked second only to 1NN in the critical difference diagram.

Weight was the lowest when it come to accuracy and performance in the critical difference diagram. Although it is shown here to almost have a negative effect, it is likely that this comes from incorrect implementation of the method, as many resources online have suggested an improvement to accuracy because of the weighted voting scheme. If time permitted this would be investigated and any required adjustments would be implemented.

Ensembles were built with every option enabled, and performed very well across all data sets as Fig 8 shows. Across all data sets, the Ensemble of KNN classifiers outperformed the 1NN by 22/35. This shows that Ensembles can improve the accuracy of a classifier by collating all of the results of many different classifiers.

Rotation Forest was the most accurate classifier used, and provided the highest results. However, Random Forest also performed greatly, only just falling short of Rotation Forest in accuracy and critical difference diagrams. As both classifiers performed incredibly well, either could be chosen, but for this particular experiment Rotation Forest was the highest performer.

References

[1] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso, "Rotation forest: A new classifier ensemble method," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 10, pp. 1619–1630, 2006.

[2] A. Liaw, M. Wiener *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[3] H. Beniwal, "Handwritten digit recognition using machine learning," Dec 2018. [Online]. Available: https://medium.com/@himanshubeniwal/handwritten-digit-recognition-using-machine-learning-ad30562a9b64

[4] S. A. Dudani, "The distance-weighted k-nearest-neighbor rule," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 4, pp. 325–327, 1976.

[5] L. E. Peterson, "K-nearest neighbor," *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009.

[6] "Uci machine learning repository." [Online]. Available: http://archive.ics.uci.edu/ml/machine-learning-databases/

[7] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, and C. D. Spyropoulos, "An experimental comparison of naive bayesian and keyword-based anti-spam filtering with personal e-mail messages," in *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2000, pp. 160–167.

[8] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[9] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22, 2001, pp. 41–46.

Appendices

	Rotation Forest	Random Forest	1NN	Ensemble	Nave Bayes
TPR	94.3	94.6	92.7	93.7	97.3
FPR	5	5.3	13.5	13	31.7
FNR	5.7	5.4	7.3	6.3	2.7
TNR	95	94.7	86.5	87	68.3

Table 1: Classifier and TPR,FPR,FNR,TNR for spambase.

Table 2: A table depicting all data sets used for experimentation.

Data-Set	No.Cases	No.Attributes	No.Classes	Distribution
bank	4521	16	2	[4000,521]
blood	748	4	2	[570,178]
breast-cancer-wisc-diag	569	30	2	[357,212]
breast-tissue	106	9	6	[21,15,18,16,14,22]
cardiotocography-10clases	2126	21	10	[384,579,53,81,72,332,252,107,69,197]
conn-bench-sonar-mines-rocks	208	60	2	[111,97]
conn-bench-vowel-deterding	990	11	11	[90,90,90,90,90,90,90,90,90,90,90]
ecoli	336	7	8	[143,77,52,35,20,5,2,2]
glass	214	9	6	[70,76,17,13,9,29]
hill-valley	1212	100	2	[606,606]
image-segmentation	2310	18	7	[330,330,330,330,330,330,330]
ionosphere	351	33	2	[126,225]
iris	150	4	3	[50,50,50]
libras	360	90	15	[24,24,24,24,24,24,24,24,24,24,24,24,24,24,24]
oocytes_merluccius_nucleus_4d	1022	41	2	[337,685]
oocytes_trisopterus_states_5b	912	32	3	[525,14,373]
optical	5620	62	10	[554,571,557,572,568,558,558,566,554,562]
ozone	2536	72	2	[2463,73]
page-blocks	5473	10	5	[4913,329,28,88,115]
parkinsons	195	22	2	[48,147]
planning	182	12	2	[130,52]
post-operative	90	8	3	[64,2,24]
ringnorm	7400	20	2	[3664,3736]
seeds	210	7	3	[70,70,70]
spambase	4601	57	2	[2788,1813]
statlog-landsat	6435	36	6	[1533,709,1358,626,707,1508]
statlog-vehicle	846	18	4	[218,212,217,199]
steel-plates	1941	27	7	[158,190,391,72,55,402,673]
synthetic-control	600	60	6	[100,100,100,100,100,100]
twonorm	7400	20	2	[3703,3697]
vertebral-column-3clases	310	6	3	[60,100,150]
wall-following	5456	24	4	[2205,826,2097,328]
waveform-noise	5000	40	3	[1692,1653,1655]
wine-quality-white	4898	11	7	[20,163,1457,2198,880,175,5]
yeast	1484	8	10	[463,429,244,163,51,44,35,30,20,5]

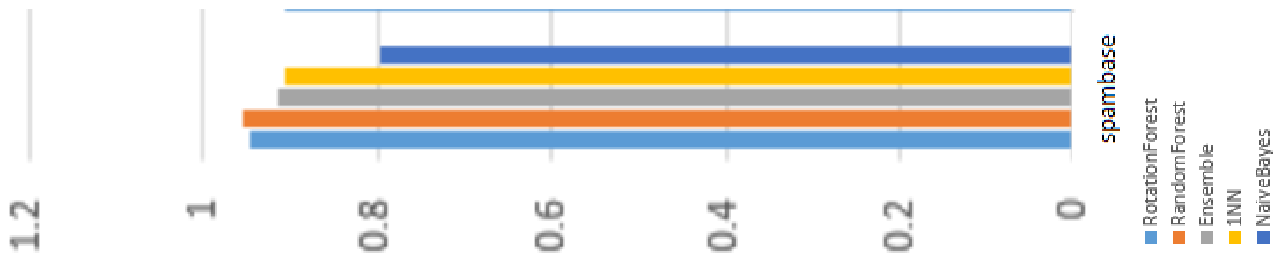

Figure 3: Spambase & classifier accuracy

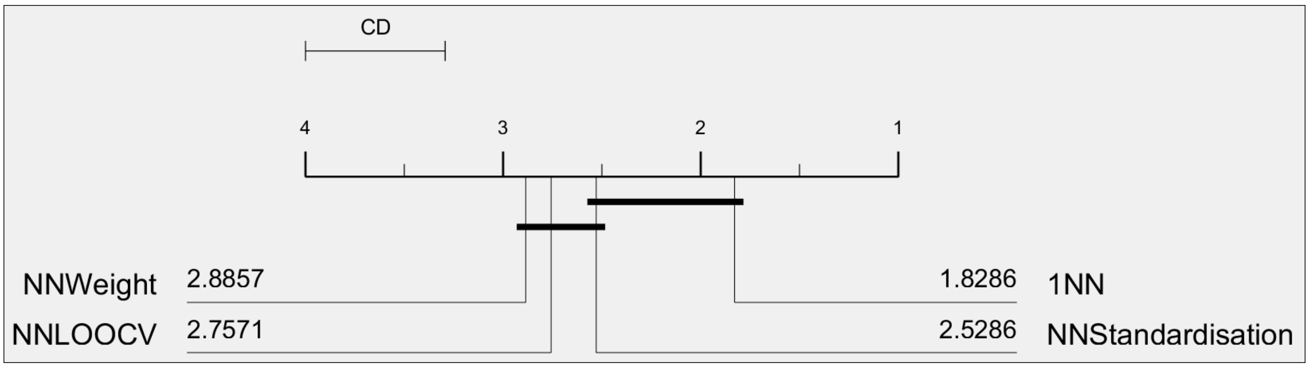

Figure 4: KNN classifier method critical difference

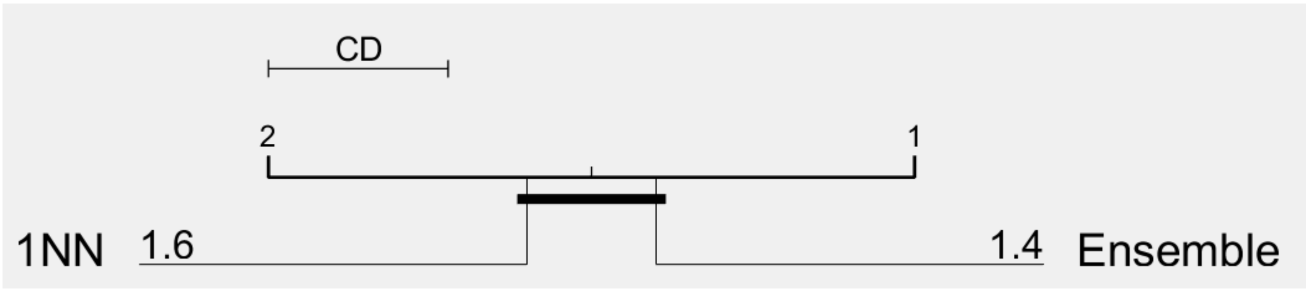

Figure 5: Critical Difference Ensemble vs 1NN

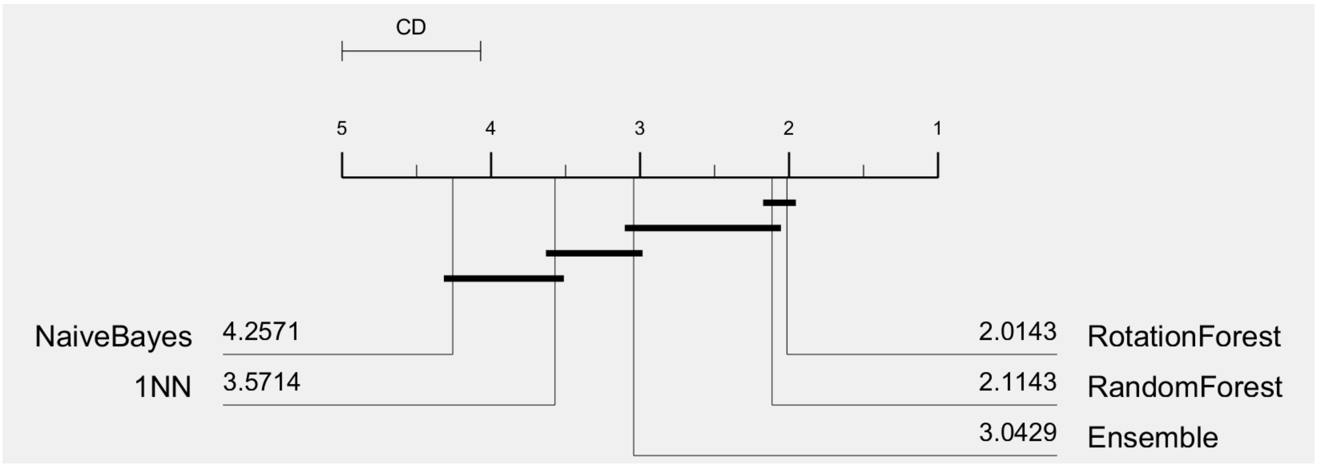

Figure 6: Rotation Forest, Random Forest, Ensemble, 1NN, Naive Bayes, critical difference

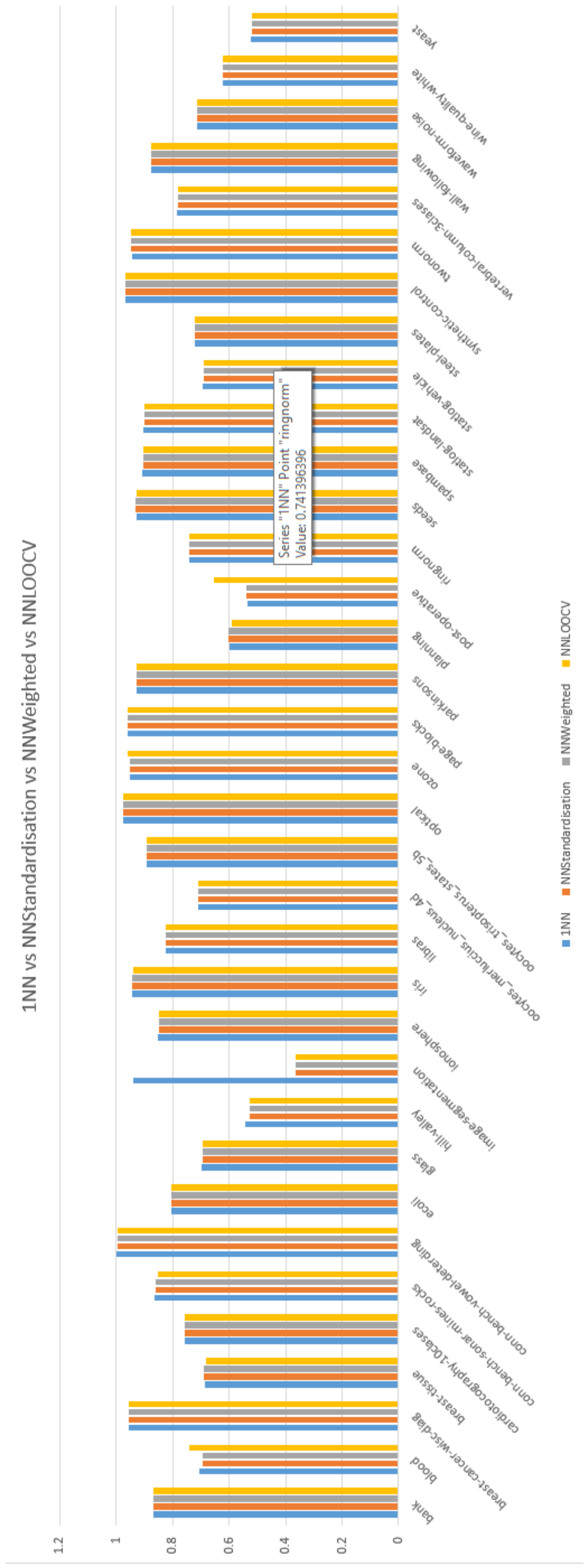

Figure 7: Classifier methods and their impact.

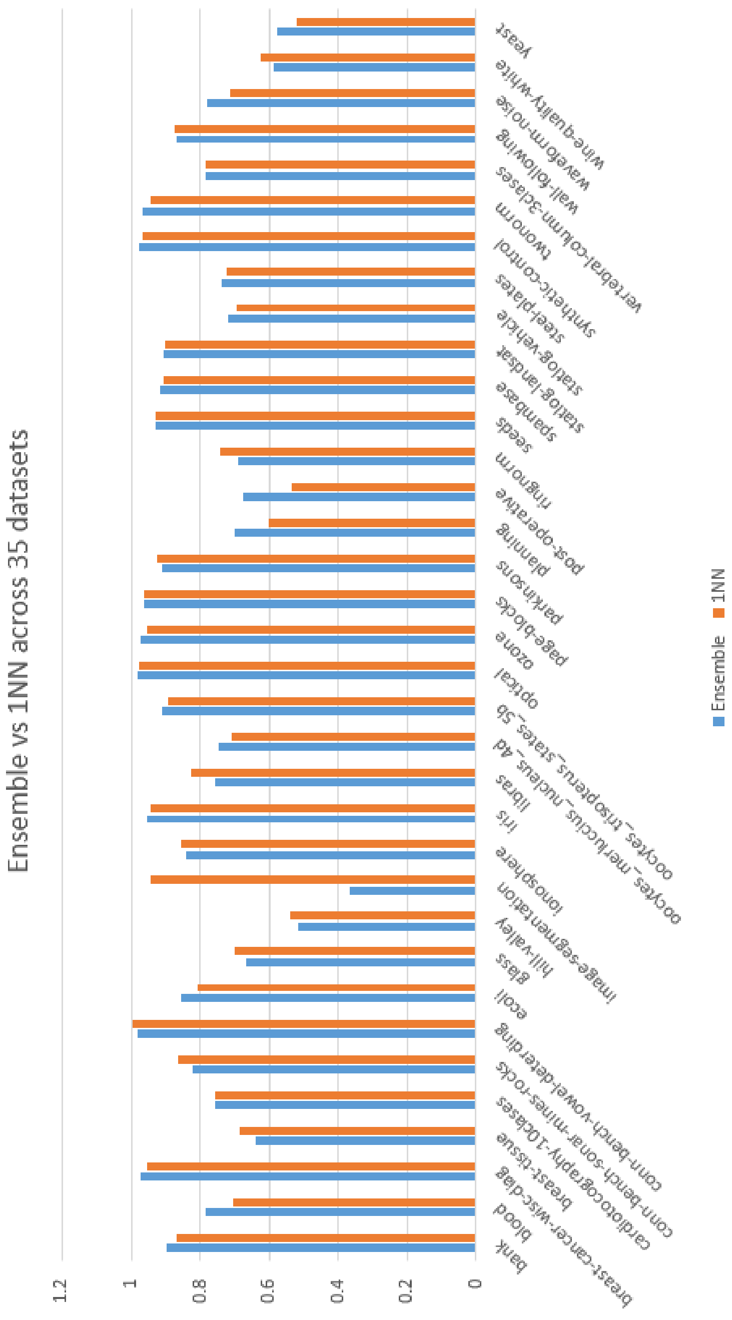

Figure 8: Ensemble accuracy vs 1nn

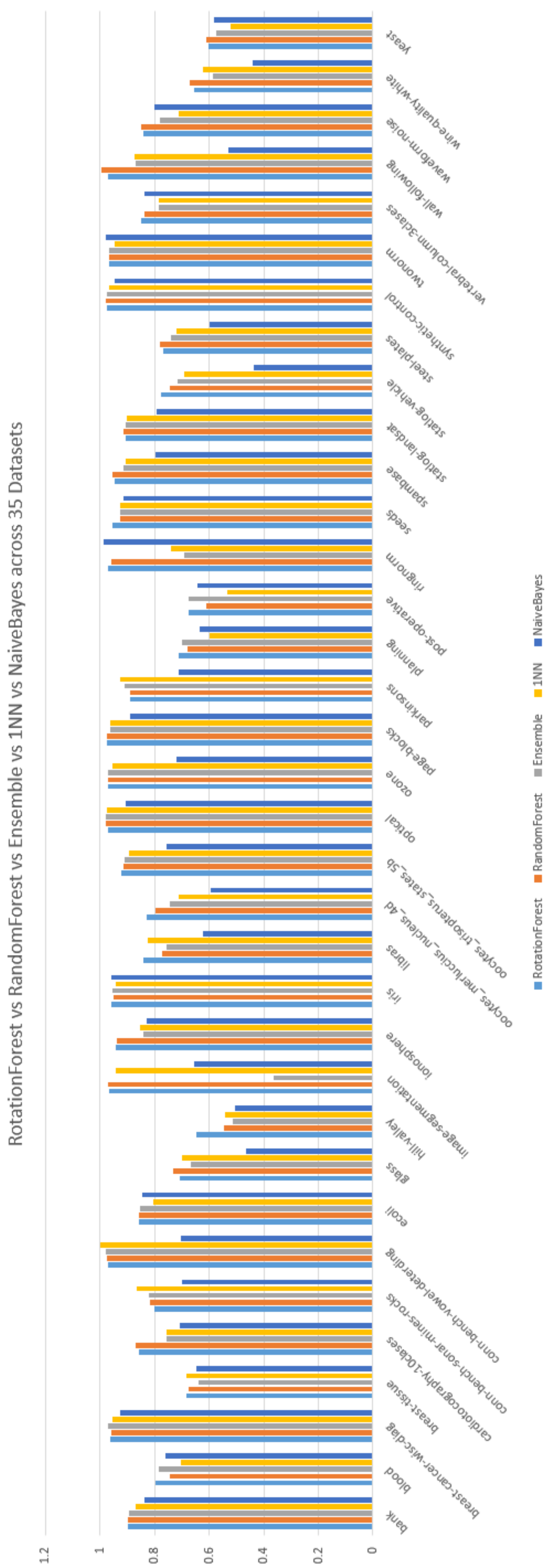

Figure 9: Rotation Forest, Random Forest, Ensemble, 1NN, Naive Bayes. 30 fold accuracy

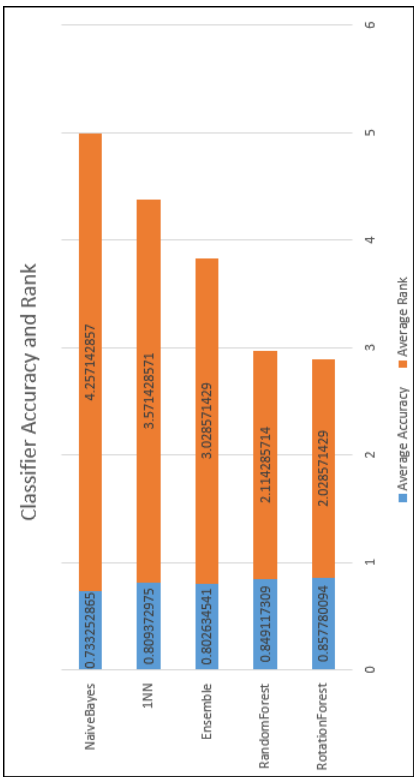

Figure 10: Rank and Accuracy/classifier

ml-cswk2-100135292-file1.pdf

Filename scrubbed (one or more forbidden characters found). Original name:

`Machine Learning - Part 1.pdf`

(Included PDF starts on next page.)

Table 1 - Pitcher

width (cm)	length (cm)	Species
9	13	N. truncata
5	14	N. raja
7	14	N. truncata
6	11	N. raja
6	12	N. raja
5	12	N. raja
8	15	N. raja
6	15	N. truncata
8	13	N. raja
7	12	N. truncata
4	13	N. raja
9	16	N. truncata

Table 2 – Unclassified

width (cm)	length (cm)	Species
5	13	N. raja
6	14	N. truncata
7	15	N. truncata
8	12	N. truncata

Distance	**5, 13**	**6, 14**	**7, 15**	**8, 12**
9, 13	4	3.162	2.828	1.414
5, 14	1	1	2.236	3.606
7, 14	2.236	1	1	2.236
6, 11	2.236	3	4.123	2.236
6, 12	1.141	2	3.162	2
5, 12	1	2.236	3.606	3
8, 15	3.606	2.236	1	3
6, 15	2.236	1	1	3.606
8, 13	3	2.236	2.236	1
7, 12	2.236	2.236	3	1
4, 13	1	2.236	3.606	4.123
9, 16	5	3.605	2.236	4.123

Formula = $\sqrt{((q_i - c_i)^2 + (q2_i - c2_i)^2)}$

Working out: 5, 13 Nearest Neighbors: 2,6,11	Working out: 6, 14 Nearest Neighbors: 2,3,8
1. $\sqrt{((5-9)^2+(13-13)^2)}=4$	1. $\sqrt{((6-9)^2+(14-13)^2)}=3.162$
2. $\sqrt{((5-5)^2+(13-14)^2)}=1$	2. $\sqrt{((6-5)^2+(14-14)^2)}=1$
3. $\sqrt{((5-7)^2+(13-14)^2)}=2.236$	3. $\sqrt{((6-7)^2+(14-14)^2)}=1$
4. $\sqrt{((5-6)^2+(13-11)^2)}=2.236$	4. $\sqrt{((6-6)^2+(14-11)^2)}=3$
5. $\sqrt{((5-6)^2+(13-12)^2)}=1.414$	5. $\sqrt{((6-6)^2+(14-12)^2)}=2$
6. $\sqrt{((5-5)^2+(13-12)^2)}=1$	6. $\sqrt{((6-5)^2+(14-12)^2)}=2.236$
7. $\sqrt{((5-8)^2+(13-15)^2)}=3.606$	7. $\sqrt{((6-8)^2+(14-15)^2)}=2.236$
8. $\sqrt{((5-6)^2+(13-15)^2)}=2.236$	8. $\sqrt{((6-6)^2+(14-15)^2)}=1$
9. $\sqrt{((5-8)^2+(13-13)^2)}=3$	9. $\sqrt{((6-8)^2+(14-13)^2)}=2.236$
10. $\sqrt{((5-7)^2+(13-12)^2)}=2.236$	10. $\sqrt{((6-7)^2+(14-12)^2)}=2.236$
11. $\sqrt{((5-4)^2+(13-13)^2)}=1$	11. $\sqrt{((6-4)^2+(14-13)^2)}=2.236$
12. $\sqrt{((5-9)^2+(13-16)^2)}=5$	12. $\sqrt{((6-9)^2+(14-16)^2)}=3.606$
Working out: 7, 15 Nearest Neighbors: 3,7,8	Working out: 8, 12 Nearest Neighbors: 1,9,10
1. $\sqrt{((7-9)^2+(15-13)^2)}=2.828$	1. $\sqrt{((8-9)^2+(12-13)^2)}=1.414$
2. $\sqrt{((7-5)^2+(15-14)^2)}=2.236$	2. $\sqrt{((8-5)^2+(12-14)^2)}=3.606$
3. $\sqrt{((7-7)^2+(15-14)^2)}=1$	3. $\sqrt{((8-7)^2+(12-14)^2)}=2.236$
4. $\sqrt{((7-6)^2+(15-11)^2)}=4.123$	4. $\sqrt{((8-6)^2+(12-11)^2)}=2.236$
5. $\sqrt{((7-6)^2+(15-12)^2)}=3.162$	5. $\sqrt{((8-6)^2+(12-12)^2)}=2$
6. $\sqrt{((7-5)^2+(15-12)^2)}=3.606$	6. $\sqrt{((8-5)^2+(12-12)^2)}=3$
7. $\sqrt{((7-8)^2+(15-15)^2)}=1$	7. $\sqrt{((8-8)^2+(12-15)^2)}=3$
8. $\sqrt{((7-6)^2+(15-15)^2)}=1$	8. $\sqrt{((8-6)^2+(12-15)^2)}=3.606$
9. $\sqrt{((7-8)^2+(15-13)^2)}=2.236$	9. $\sqrt{((8-8)^2+(12-13)^2)}=1$
10. $\sqrt{((7-7)^2+(15-12)^2)}=3$	10. $\sqrt{((8-7)^2+(12-12)^2)}=1$
11. $\sqrt{((7-4)^2+(15-13)^2)}=3.606$	11. $\sqrt{((8-4)^2+(12-13)^2)}=4.123$
12. $\sqrt{((7-9)^2+(15-16)^2)}=2.236$	12. $\sqrt{((8-9)^2+(12-16)^2)}=4.123$