ΑΝΑΓΝΩΡΙΣΗ ΠΡΟΤΥΠΩΝ
1η Σειρά Ασκήσεων
Ταξινόμηση (κατηγοριοποίηση δεδομένων)

Τσάλεσης Ευάγγελος, ΑΜ: 1779

1. ΕΙΣΑΓΩΓΗ

Στόχος της άσκησης ήταν η υλοποίηση τεσσάρων μεθόδων. Οι μέθοδοι είναι:

- ***kNN*** – ***Nearest Neighbor Classifier με ευκλείδεια απόσταση***. Δοκιμάστηκαν οι εξής τιμές του k = [1,3,5,7,9].
- ***Naive Bayes Classifier,*** υποθέτοντας κανονική κατανομή.
- ***Least-Squares Linear Classifier.***
- ***Gradient-Descent Linear Classifier,*** υποθέτοντας σιγμοειδή συνάρτηση. Δοκιμάστηκαν διάφορες τιμές του ρυθμού μάθησης ***η*** για τις οποίες η μέθοδος συγκλίνει.

Η υλοποίηση έγινε σε γλώσσα προγραμματισμού java (ο πλήρης κώδικας παρατίθεται στο παράρτημα Α).

Μετά την υλοποίηση οι μέθοδοι δοκιμάστηκαν σε δύο πειραματικά σύνολα δεδομένων 2 διαστάσεων και 2 κατηγοριών (Synthetic: 250 δειγμάτων, Clouds: 5000 δειγμάτων). Υπολογίστηκε το ποσοστό επιτυχίας πρόβλεψης κάθε μεθόδου για κάθε σύνολο δεδομένων και με βάση αυτά τα αποτελέσματα συγκρίθηκε η απόδοση των μεθόδων ταξινόμησης.

2. ΥΛΟΠΟΙΗΣΗ

2.1 Εισαγωγή

Κατά την εκτέλεση των μεθόδων κάθε σύνολο δεδομένων χωρίζεται σε 10 ίσα υποσύνολα. Από αυτά τα υποσύνολα, το 1 χρησιμοποιείται ως δεδομένα ελέγχου και τα υπόλοιπα 9 ως δεδομένα εκπαίδευσης. Η διαδικασία επαναλαμβάνεται 10 φορές, μια για κάθε υποσύνολο (με τα υπόλοιπα να είναι τα δεδομένα εκπαίδευσης). Η τελική επίδοση κάθε μεθόδου προκύπτει ως μέση επίδοση των 10 επαναλήψεων.

Πριν την ανάγνωση των αρχείων δεδομένων από το πρόγραμμα ταξινόμησης, τα δείγματα «ανακατεύτηκαν» με τυχαίο τρόπο ώστε να διασφαλιστεί η ανεξαρτησία των υποσυνόλων.

2.2 Nearest Neighbor classifier

Για κάθε δείγμα του υποσυνόλου ελέγχου υπολογίζεται η ευκλείδεια απόσταση του από κάθε δείγμα των υποσυνόλων εκπαίδευσης. Οι αποστάσεις τοποθετούνται ταξινομημένες σε μία συνδεδεμένη λίστα k στοιχείων(με κόμβους που περιέχουν τα στοιχεία απόσταση, κατηγορία), έτσι ώστε μετά το τέλος του υπολογισμού αποστάσεων αυτή η λίστα να περιέχει τις κατηγορίες των κοντινότερων δειγμάτων.

Στην συνέχεια μετράται ο αριθμός των γειτονικών δειγμάτων κατηγορίας 0 ή 1 και η κατηγορία με τον μεγαλύτερο αριθμό δειγμάτων επιλέγεται ως πρόβλεψη. Αν συμπίπτει με την πραγματική κατηγορία του δείγματος ελέγχου η μέθοδος επιτυγχάνει.

Ο λόγος του αριθμού επιτυχιών προς τον αριθμό των δειγμάτων ελέγχου αποτελεί και το ποσοστό επιτυχίας της μεθόδου.

Όπως αναφέρθηκε και παραπάνω η διαδικασία επαναλαμβάνεται 10 φορές και η τελική επίδοση προκύπτει ως μέση επίδοση των 10 επαναλήψεων.

Το πρόγραμμα ταξινομεί με βάση 1, 3, 5, 7 και 9 γείτονες (μονοί αριθμοί ώστε να υπάρχει πάντα πλειοψηφούσα κατηγορία).

2.3 Naïve Bayes classifier

Τα δείγματα εκπαίδευσης χωρίζονται σε δύο πίνακες, ένα για κάθε κατηγορία. Για κάθε χαρακτηριστικό και κάθε κατηγορία των δειγμάτων υπολογίζονται οι μέσες τιμές και οι διακυμάνσεις.

Επίσης υπολογίζονται οι πιθανότητες των 2 κατηγοριών $P(\omega_0)$, $P(\omega_1)$.

Για κάθε χαρακτηριστικό d και κάθε κατηγορία ω_i κάθε δείγματος ελέγχου υπολογίζεται η τιμή $p(x_d \mid \omega_i)$, υποθέτοντας κανονική κατανομή.

Τελικά για κάθε δείγμα ελέγχου (x_1,x_2) ο εκτιμητής υπολογίζεται ως:

$P(\omega_0)\, p(x_1 \mid \omega_0)\, p(x_2 \mid \omega_0)$ $(> \text{ή} <)$ $P(\omega_1)\, p(x_2 \mid \omega_1)\, p(x_2 \mid \omega_1)$

(αν είναι μεγαλύτερος ο πρώτος όρος επιλέγεται η κατηγορία 0 και αν είναι ο δεύτερος επιλέγεται η κατηγορία 1).

Αν συμπίπτει η εκτίμηση με την πραγματική κατηγορία του δείγματος ελέγχου η μέθοδος επιτυγχάνει.

Στην συνέχεια η απόδοση της μεθόδου υπολογίζεται όπως και στην παραπάνω μέθοδο.

2.4 Least-Squares Linear Classifier

Αρχικά ορίζονται οι πίνακες X, Y από τα δείγματα εκπαίδευσης της μεθόδου. Κάθε γραμμή του X περιέχει τις τιμές $\{1, x_1, x_2\}$ και του Y περιέχει τις τιμές -1 για κατηγορία 0 και 1 για κατηγορία 1.

Στην συνέχεια υπολογίζονται κατά σειρά, ο ανάστροφος X^T, ο X^TX, ο αντίστροφος $(X^TX)^{-1}$, ο $(X^TX)^{-1}X^T$ και τελικά ο $W = (X^TX)^{-1}X^TY$ με τα βάρη της γραμμικής συνάρτησης ταξινόμησης.

Τελικά γίνεται εκτίμηση της κατηγορίας κάθε δείγματος ελέγχου σύμφωνα με τα βάρη της γραμμικής συνάρτησης που υπολογίστηκε παραπάνω.

$f(x_1,x_2) = w_0 + w_1x_1 + w_2x_2$

- Κατηγορία 0 αν $f(x_1,x_2) < 0$
- Κατηγορία 1 αν $f(x_1,x_2) >= 0$

Αν συμπίπτει η εκτίμηση με την πραγματική κατηγορία του δείγματος ελέγχου η μέθοδος επιτυγχάνει.

Στην συνέχεια η απόδοση της μεθόδου υπολογίζεται όπως και στις παραπάνω μεθόδους.

2.5 *Gradient-Descent Linear Classifier*

Η μέθοδος εκτελείται επαναληπτικά για διάφορους ρυθμούς μάθησης, $\eta = \{0.001, 0.0005, 0.0001\}$, οι οποίοι επιλέχθηκαν δοκιμαστικά ώστε η μέθοδος να συγκλίνει για τα σύνολα δεδομένων που επεξεργαζόμαστε.

Σε κάθε επανάληψη ορίζεται ο πίνακας για τις αρχικές τιμές των βαρών ω καθώς και οι πίνακες X, Y όπως στην παραπάνω μέθοδο των ελαχίστων τετραγώνων, με την διαφορά ότι εδώ η τιμή για την κατηγορία 0 στον πίνακα Y είναι 0.

Στην συνέχεια προσεγγίζονται επαναληπτικά τα βάρη ω, υποθέτοντας συνάρτηση ενεργοποίησης σιγμοειδή συνάρτηση, μέχρι η διαφορά του τετραγωνικού σφάλματος μεταξύ

δύο διαδοχικών προσεγγίσεων των βαρών να είναι μικρότερη από ένα κατώφλι σφάλματος ***ε = 0.0001.***

Αφού προσεγγιστούν τα βάρη, η εκτίμηση κατηγορίας και ο υπολογισμός της απόδοσης της μεθόδου γίνεται όπως στην μέθοδο ελαχίστων τετραγώνων

3. ΑΠΟΤΕΛΕΣΜΑΤΑ

3.1 Nearest Neighbor classifier

Η απόδοση της μεθόδου, καθώς και ο χρόνος εκτέλεσης του αλγορίθμου, για διάφορες τιμές του ***k*** για τα δύο πειραματικά σύνολα ***"Synthetic"*** (250 δείγματα) και ***"Clouds"*** (5000 δείγματα) φαίνονται στους παρακάτω πίνακες.

Synthetic

k	Accuracy	Time (sec)
1	0.852	0.097
3	0.848	0.057
5	0.820	0.058
7	0.860	0.059
9	0.860	0.057

Clouds

k	Accuracy	Time (sec)
1	0.844	24.126
3	0.874	24.776
5	0.880	24.581
7	0.885	24.463
9	0.889	24.686

Από τα παραπάνω αποτελέσματα συμπεραίνεται ότι για τα υπό μελέτη πειραματικά σύνολα δεδομένων η επιρροή στην απόδοση του αριθμού των γειτονικών δειγμάτων που λαμβάνονται υπόψη είναι μικρή αλλά υπαρκτή, ενώ ο χρόνος εκτέλεσης εξαρτάται από τον αριθμό των δειγμάτων. Η μέθοδος υλοποιήθηκε με τέτοιο τρόπο ώστε ο παράγοντας k να μην επηρεάζει τον χρόνο εκτέλεσης.

3.2 Naive Bayes classifier

Η απόδοση της μεθόδου, καθώς και ο χρόνος εκτέλεσης του αλγορίθμου, για την μέθοδο είναι:

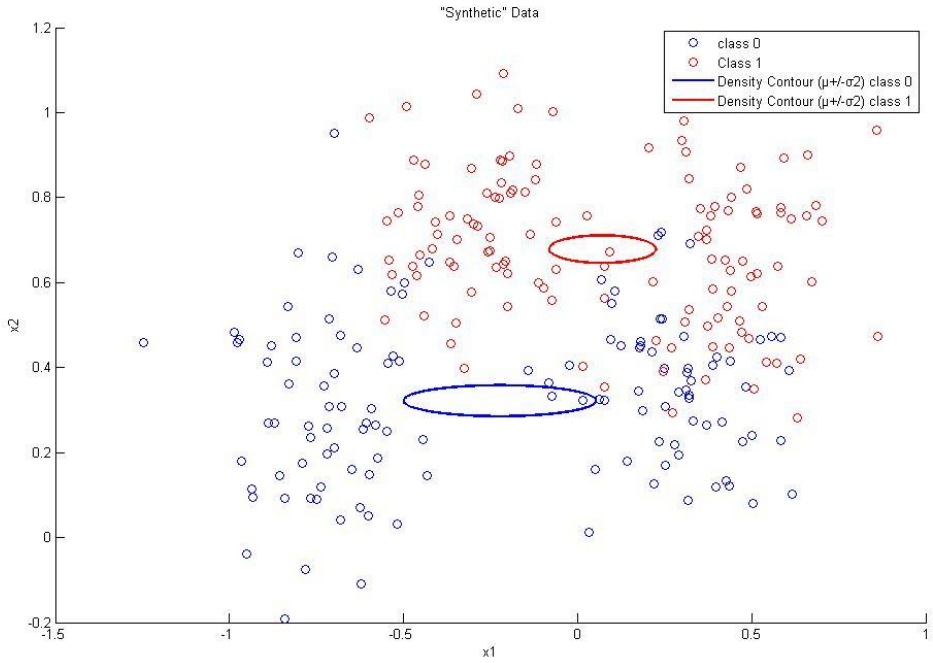
Synthetic

Accuracy: 0.840 **Time:** 0.004 sec

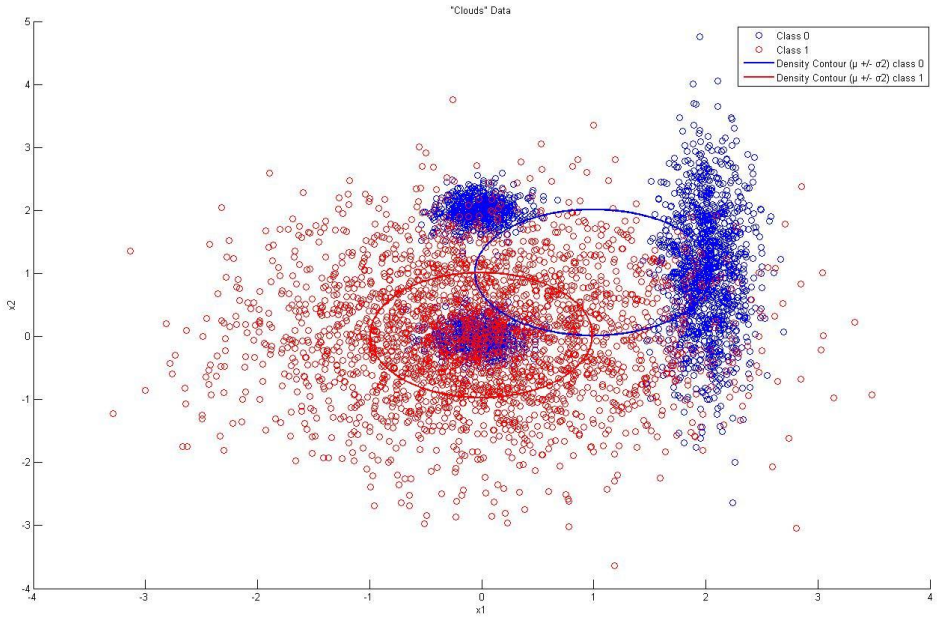
Clouds

Accuracy: 0.750 **Time:** 0.079 sec

Ενδεικτικά παρουσιάζονται τα δεδομένα και ισοϋψείς των κανονικών κατανομών που προέκυψαν από την μέθοδο.



Εικόνα 1: Μέθοδος naive bayes classifier για πειραματικά δεδομένα "Synthetic"



Εικόνα 2: Μέθοδος naive bayes classifier για πειραματικά δεδομένα "Clouds"

Η μέθοδος εμφανίζει καλή απόδοση (ιδιαίτερα για το σύνολο Synthetic) η οποία απόδοση εξαρτάται από τον βαθμό που ισχύουν οι κύριες υποθέσεις της μεθόδου, περί ανεξαρτησίας των χαρακτηριστικών των δειγμάτων και της κανονικής κατανομής τους.

Το μεγάλο πλεονέκτημα της μεθόδου είναι η ταχύτητα της (κλάσμα του δευτερολέπτου ακόμα και για το σύνολο των 5000 δειγμάτων)

3.3 Least-Squares linear classifier

Η απόδοση της μεθόδου, καθώς και ο χρόνος εκτέλεσης του αλγορίθμου, για την μέθοδο είναι:

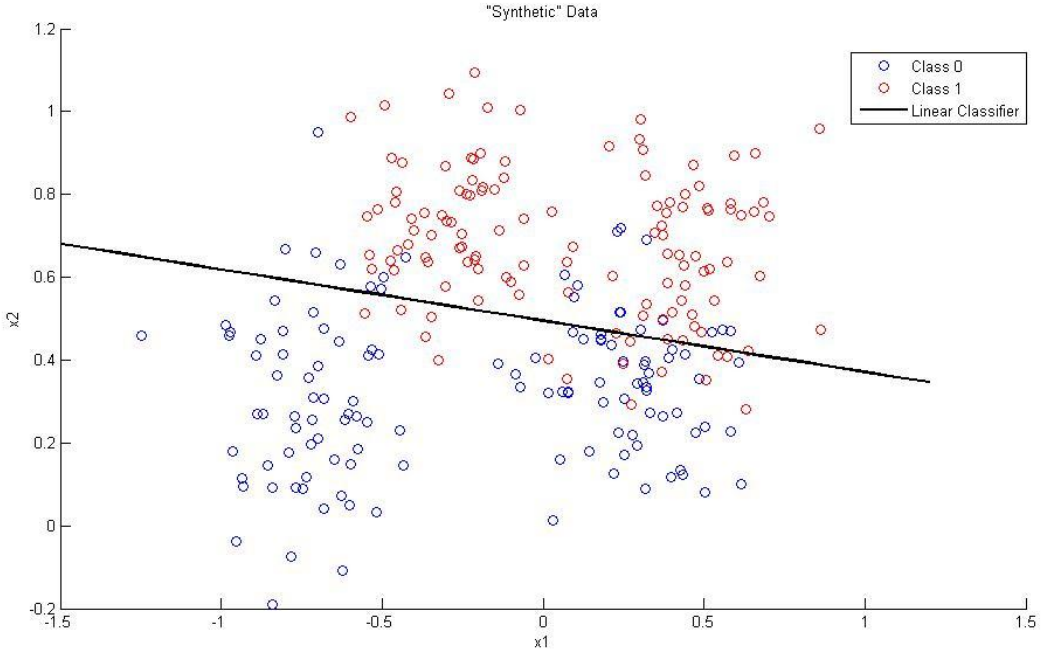
Synthetic

Accuracy: 0.848 **Time:** 0.006 sec

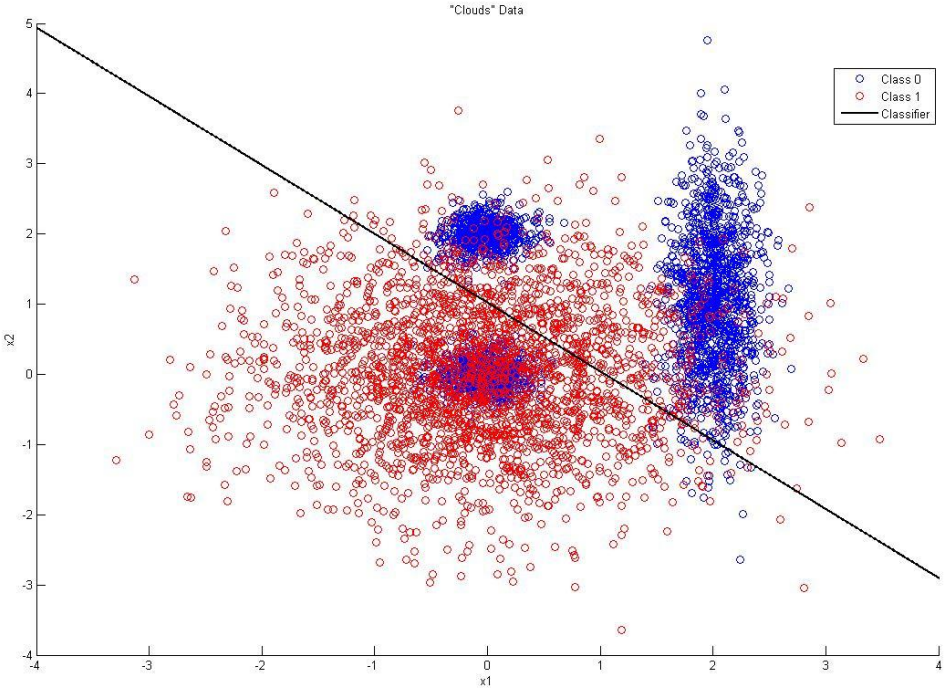
Clouds

Accuracy: 0.749 **Time:** 0.067 sec

Ενδεικτικά παρουσιάζονται τα δεδομένα και η γραμμική διαχωριστική επιφάνεια που προέκυψε από την μέθοδο για κάθε πειραματικό σύνολο:



Εικόνα 3: Μέθοδος least-squares linear classifier για πειραματικά δεδομένα "Synthetic"



Εικόνα 4: Μέθοδος least-squares linear classifier για πειραματικά δεδομένα "Clouds"

Η μέθοδος εμφανίζει παρόμοια αποτελέσματα απόδοσης και χρόνου με την παραπάνω μέθοδο Naive bayes classifier με υπόθεση κανονικής κατανομής, κάτι που είναι και συμβατό με την θεωρία, και οι δύο μέθοδοι παράγουν γραμμική επιφάνεια διάκρισης (αφου οι πιθανότητες $P(\omega_0)$, $P(\omega_1)$ της μεθόδου naïve bayes και για τα δύο σύνολα δεδομένων είναι περίπου ίσες).

3.4 *Gradient-Descent Linear Classifier*

Η απόδοση της μεθόδου, καθώς και ο χρόνος εκτέλεσης του αλγορίθμου, για διάφορες τιμές του ρυθμού μάθησης $\boldsymbol{\eta}$ για τα δύο πειραματικά σύνολα φαίνονται στους παρακάτω πίνακες.

Κατώφλι σφάλματος $\boldsymbol{\varepsilon = 0.0001}$.

Synthetic

η	Accuracy	Time (sec)
0.001	0.856	7.568
0.0005	0.856	10.591
0.0001	0.856	21.556

Clouds

η	Accuracy	Time (sec)
0.001	0.75	1.575
0.0005	0.75	3.073
0.0001	0.749	12.729

Η απόδοση της μεθόδου για διάφορους ρυθμούς μάθησης συγκλίνει στην ίδια τιμή (η οποία είναι παρόμοια και με τις προηγούμενες 2 γραμμικές μεθόδους) και ο χρόνος εκτέλεσης αυξάνει με την μείωση του ρυθμού μάθησης. Η σύγκλιση της μεθόδου εξαρτάται από τα χαρακτηριστικά των πειραματικών συνόλων, τον ρυθμό μάθησης και το ύψος του κατωφλίου σφάλματος.

4. ΣΥΜΠΕΡΑΣΜΑΤΑ

Από τα παραπάνω αποτελέσματα συνάγεται ότι για τα εξεταζόμενα πειραματικά σύνολα δεδομένων η μέθοδος με την μεγαλύτερη απόδοση είναι αυτή των κοντινότερων γειτόνων. Όμως είναι και η πιο αργή μέθοδος, της οποίας ο χρόνος εκτέλεσης αυξάνεται γραμμικά με την αύξηση του αριθμού δειγμάτων.

Ταχύτερες μέθοδοι με παρόμοια απόδοση μεταξύ τους είναι αυτές των ελαχίστων τετραγώνων και της “naive bayes”, ενώ η “gradient descent” απαιτεί βέλτιστες επιλογές ρυθμού μάθησης και κατωφλίου σφάλματος, οι οποίες επιλέγονται δοκιμαστικά για κάθε σύνολο δεδομένων ώστε να συγκλίνουν σε πρακτικό χρόνο. Παρουσιάζει παρόμοια απόδοση με τις προηγούμενες 2 μεθόδους.

Τελικά, για τα εξεταζόμενα πειραματικά σύνολα δεδομένων, αν δεν μας ενδιαφέρει ο χρόνος εκτέλεσης ή αν το σύνολο δεδομένων είναι μικρό, επιλέγεται η μέθοδος των κοντινότερων γειτόνων, ενώ αν μας ενδιαφέρει ο χρόνος εκτέλεσης επιλέγεται κάποια από τις «ελαχίστων τετραγώνων» και «naive bayes».

ΠΑΡΑΡΤΗΜΑ Α (κώδικας)

```
// TSALESIS EVANGELOS
// AM: 1779

import java.util.Scanner;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.ArrayList;


public class PatternRecognition1 {
    // MAIN
    public static void main(String[] args) {
        // imports data into a string array (every element a line of data file)
        Scanner inputReader = null;
        ArrayList<String> lines = new ArrayList<>();
        try {
            inputReader = new Scanner(new
FileInputStream("C:/users/vagtsal/desktop/assignment1/data/clouds.dat"));
        }
        catch (FileNotFoundException e){
            System.out.println("File not found");
            System.exit(0);
        }
        while (inputReader.hasNextLine()){
            lines.add(inputReader.nextLine());
        }

        System.out.println("K-Nearest Neighbour Classifier");
        kNN(lines);
        System.out.println("---------------------------");
        System.out.println("Naive Bayes Classifier");
        NBC(lines);
        System.out.println("---------------------------");
        System.out.println("Least Squares Linear Classifier");
        LS_LC(lines);
        System.out.println("---------------------------");
        System.out.println("Gradient Descent Linear Classifier");
        GD_LC(lines);
        System.out.println("---------------------------");

    }



    // **********************************'Naive Bayes classifier'
method*****************************************
    public static void NBC(ArrayList<String> lines){
        long time = System.currentTimeMillis();

        String[] sample;
        int matrixSize = lines.size()-lines.size()/10;
        double sumAccuracy = 0.0;

        // folds loop
        for (int f=0; f<10; f++){
            int success = 0;

            double[][] X0 = new double[matrixSize][2];
            double[][] X1 = new double[matrixSize][2];

            // defines matrices X0 (0 class samples), X1 (1 class samples)
            int c0 = 0;
            int c1 = 0;
            int j = 0;
            if (j == f*lines.size()/10){
                j = j + lines.size()/10;
            }
```

```
while (j<lines.size()){
    sample  = lines.get(j).trim().split("\\s+");
    if (Byte.parseByte(sample[2]) == 0){
        X0[c0][0] = Double.parseDouble(sample[0]);
        X0[c0][1] = Double.parseDouble(sample[1]);
        c0++;
    }
    else {
        X1[c1][0] = Double.parseDouble(sample[0]);
        X1[c1][1] = Double.parseDouble(sample[1]);
        c1++;
    }

    j++;
    if (j == f*lines.size()/10){
        j = j + lines.size()/10;
    }
}

// calculates P(class 0), P(class 1)
double P0 = (c0+1)/(double)matrixSize;
double P1 = (c1+1)/(double)matrixSize;

// calculates means for x1|class0, x2|class0
double sum1 = 0;
double sum2 = 0;
for (int i=0; i<=c0; i++){
    sum1 = sum1 + X0[i][0];
    sum2 = sum2 + X0[i][1];
}
double m10 = sum1 / (c0+1);
double m20 = sum2 / (c0+1);

// calculates means for x1|class1, x2|class1
sum1 = 0;
sum2 = 0;
for (int i=0; i<=c1; i++){
    sum1 = sum1 + X1[i][0];
    sum2 = sum2 + X1[i][1];
}
double m11 = sum1 / (c1+1);
double m21 = sum2 / (c1+1);

// calculates variances for x1|class0, x2|class0
sum1 = 0;
sum2 = 0;
for (int i=0; i<=c0; i++){
    sum1 = sum1 + (X0[i][0] - m10)*(X0[i][0] - m10);
    sum2 = sum2 + (X0[i][1] - m20)*(X0[i][1] - m20);
}
double s10 = sum1 / (c0+1);
double s20 = sum2 / (c0+1);

// calculates variances for x1|class1, x2|class1
sum1 = 0;
sum2 = 0;
for (int i=0; i<=c1; i++){
    sum1 = sum1 + (X1[i][0] - m11)*(X1[i][0] - m11);
    sum2 = sum2 + (X1[i][1] - m21)*(X1[i][1] - m21);
}
double s11 = sum1 / (c1+1);
double s21 = sum2 / (c1+1);

// tests the estimator on the samples of the current fold
for (int k=f*lines.size()/10; k<(f+1)*lines.size()/10; k++){
    sample = lines.get(k).trim().split("\\s+");
    double x1 = Double.parseDouble(sample[0]);
    double x2 = Double.parseDouble(sample[1]);
    byte category = Byte.parseByte(sample[2]);
```

```
                double normalDist10 = ((1/(Math.sqrt(2*Math.PI*s10)))*Math.pow(Math.E,-
(x1-m10)*(x1-m10)/(2*s10)));
                double normalDist20 = ((1/(Math.sqrt(2*Math.PI*s20)))*Math.pow(Math.E,-
(x2-m20)*(x2-m20)/(2*s20)));
                double normalDist11 = ((1/(Math.sqrt(2*Math.PI*s11)))*Math.pow(Math.E,-
(x1-m11)*(x1-m11)/(2*s11)));
                double normalDist21 = ((1/(Math.sqrt(2*Math.PI*s21)))*Math.pow(Math.E,-
(x2-m21)*(x2-m21)/(2*s21)));

                boolean estimator = (P0*normalDist10*normalDist20) <
(P1*normalDist11*normalDist21);

                if ((estimator && category == 1) || ((!estimator && category == 0))){
                    success++;
                }
            }
            double accuracy = success/(lines.size()/10.00);
            sumAccuracy = sumAccuracy + accuracy;
        }
        System.out.println("Average accuracy: " + String.format("%.3f" ,
sumAccuracy/10.00) + "\t\tTime: " + (System.currentTimeMillis() - time)/1000.00 + " sec");
    }
    //
*****************************************************************************************
**************************


    // *********************************'Gradient-Descent Linear Classifier'
method*****************************************
    public static void GD_LC(ArrayList<String> lines){
        String[] sample;
        int matrixSize = lines.size()-lines.size()/10;

        double hList[] = {0.001, 0.0005, 0.0001};       // learning rate
        double e = 0.0001;                              // error threshold

        // learning rates h loop
        for (int l=0; l<hList.length; l++){
            long time = System.currentTimeMillis();

            double h = hList[l];
            double sumAccuracy = 0.0;

            // folds loop
            for (int f=0; f<10; f++){
                int success = 0;
                double newE,oldE;

                double[][] X = new double[matrixSize][3];
                byte[] Y = new byte[matrixSize];
                double W[] = {0,0,0};    // initial W

                // imports xi,yi
                int i = 0;
                int j = 0;
                if (j == f*lines.size()/10){
                    j = j + lines.size()/10;
                }
                while (j<lines.size()){
                    // defines X,Y
                    sample  = lines.get(j).trim().split("\\s+");
                    X[i][0] = 1;
                    X[i][1] = Double.parseDouble(sample[0]);
                    X[i][2] = Double.parseDouble(sample[1]);
                    Y[i]    = Byte.parseByte(sample[2]);

                    i++;
```

```
                    j++;
                    if (j == f*lines.size()/10){
                        j = j + lines.size()/10;
                    }
                }

                // estimates W weights
                double sum = 0;
                for (int k=0; k<matrixSize; k++){
                    sum = sum + Math.pow((1/(1+Math.pow(Math.E, -(W[0]+
W[1]*X[k][1]+W[2]*X[k][2])))) - Y[k], 2);
                }
                newE = 0.5*sum;
                do {
                    oldE = newE;

                    sum = 0;
                    for (int m=0; m<3; m++){
                        for (int k=0; k<matrixSize; k++){
                            double phi = 1/(1+Math.pow(Math.E,-(W[0]+
W[1]*X[k][1]+W[2]*X[k][2])));
                            sum = sum + ((phi-Y[k])*phi*(1-phi)*X[k][m]);
                        }
                        W[m] = W[m] - h*sum;
                    }

                    sum = 0;
                    for (int k=0; k<matrixSize; k++){
                        sum = sum + Math.pow((1/(1+Math.pow(Math.E, -(W[0]+
W[1]*X[k][1]+W[2]*X[k][2])))) - Y[k], 2);
                    }
                    newE = 0.5*sum;
                } while(Math.abs(oldE - newE) > e);


                // tests the estimator on the samples of the current fold
                for (int k=f*lines.size()/10; k<(f+1)*lines.size()/10; k++){
                    sample = lines.get(k).trim().split("\\s+");
                    double x1 = Double.parseDouble(sample[0]);
                    double x2 = Double.parseDouble(sample[1]);
                    byte category = Byte.parseByte(sample[2]);

                    double estimator = W[0]+ W[1]*x1+W[2]*x2;
                    if ((estimator >= 0 && category == 1) || (estimator < 0 && category ==
0)){
                        success++;
                    }
                }
                double accuracy = success/(lines.size()/10.00);
                sumAccuracy = sumAccuracy + accuracy;
            }
            System.out.println("Average accuracy for h = " + h + " : " +
String.format("%.3f", sumAccuracy/10.00) + "\t\tTime: " + (System.currentTimeMillis() -
time)/1000.00 + " sec");
        }
    }
    //
**************************************************************************************
*************************


    // *********************************'Least-Squares Linear Classifier'
method*****************************************
    public static void LS_LC(ArrayList<String> lines){
        long time = System.currentTimeMillis();

        String[] sample;
        int matrixSize = lines.size()-lines.size()/10;
        double sumAccuracy = 0.0;
```

```
// folds loop
for (int f=0; f<10; f++){
    int success = 0;

    double[][] X = new double[matrixSize][3];
    double[][] XT = new double[3][matrixSize];
    byte[] Y = new byte[matrixSize];

    // finds W matrix
    int i = 0;
    int j = 0;
    if (j == f*lines.size()/10){
        j = j + lines.size()/10;
    }
    while (j<lines.size()){
        // defines X,Y
        sample  = lines.get(j).trim().split("\\s+");
        X[i][0] = 1;
        X[i][1] = Double.parseDouble(sample[0]);
        X[i][2] = Double.parseDouble(sample[1]);
        if (Byte.parseByte(sample[2]) ==  0){
            Y[i] = -1;
        }
        else{
            Y[i] = 1;
        }

        i++;
        j++;
        if (j == f*lines.size()/10){
            j = j + lines.size()/10;
        }
    }

    // calculates XT
    for (int m=0; m<matrixSize; m++){
        for (int n=0; n<3; n++){
            XT[n][m] = X[m][n];
        }
    }

    // calculates XT*X
    double[][] XTX = new double[3][3];
    for (int m=0; m<3; m++){
        for (int n=0; n<3; n++){
            double sum = 0;
            for (int k=0; k<matrixSize; k++){
                sum = sum + XT[m][k] * X[k][n];
            }
            XTX[m][n] = sum;
        }
    }

    // calculates inverse(XT*X)
    double det =    + XTX[0][0]*(XTX[1][1]*XTX[2][2]-XTX[1][2]*XTX[2][1])
                    - XTX[0][1]*(XTX[1][0]*XTX[2][2]-XTX[1][2]*XTX[2][0])
                    + XTX[0][2]*(XTX[1][0]*XTX[2][1]-XTX[1][1]*XTX[2][0]);
    double[][] invXTX = new double[3][3];
    if (Math.abs(det) > 0.00001){
        invXTX[0][0] = (1.00/det)*(XTX[1][1]*XTX[2][2] - XTX[2][1]*XTX[1][2]);
        invXTX[0][1] = (1.00/det)*(XTX[0][2]*XTX[2][1] - XTX[2][2]*XTX[0][1]);
        invXTX[0][2] = (1.00/det)*(XTX[0][1]*XTX[1][2] - XTX[1][1]*XTX[0][2]);
        invXTX[1][0] = (1.00/det)*(XTX[1][2]*XTX[2][0] - XTX[2][2]*XTX[1][0]);
        invXTX[1][1] = (1.00/det)*(XTX[0][0]*XTX[2][2] - XTX[2][0]*XTX[0][2]);
        invXTX[1][2] = (1.00/det)*(XTX[0][2]*XTX[1][0] - XTX[1][2]*XTX[0][0]);
        invXTX[2][0] = (1.00/det)*(XTX[1][0]*XTX[2][1] - XTX[2][0]*XTX[1][1]);
        invXTX[2][1] = (1.00/det)*(XTX[0][1]*XTX[2][0] - XTX[2][1]*XTX[0][0]);
        invXTX[2][2] = (1.00/det)*(XTX[0][0]*XTX[1][1] - XTX[1][0]*XTX[0][1]);
    }
```

```
            else {
                System.out.println("Cannot find (XTX)-1");
                System.exit(0);
            }

            // calculates inverse(XT*X)*XT
            double[][] invXTX_XT = new double[3][matrixSize];
            for (int m=0; m<3; m++){
                for (int n=0; n<matrixSize; n++){
                    double sum = 0;
                    for (int k=0; k<3; k++){
                        sum = sum + invXTX[m][k] * XT[k][n];
                    }
                    invXTX_XT[m][n] = sum;
                }
            }

            // calculates W = inverse(XT*X)*XT*Y
            double[] W = new double[matrixSize];
            for (int m=0; m<3; m++){
                double sum = 0;
                for (int k=0; k<matrixSize; k++){
                    sum = sum + invXTX_XT[m][k] * Y[k];
                }
                W[m] = sum;
            }

            // tests the estimator on the samples of the current fold
            for (int k=f*lines.size()/10; k<(f+1)*lines.size()/10; k++){
                sample = lines.get(k).trim().split("\\s+");
                double x1 = Double.parseDouble(sample[0]);
                double x2 = Double.parseDouble(sample[1]);
                byte category = Byte.parseByte(sample[2]);

                if ((W[0] + W[1]*x1 + W[2]*x2 >= 0 && category == 1) || ((W[0] + W[1]*x1 +
W[2]*x2 < 0 && category == 0))){
                    success++;
                }
            }
            double accuracy = success/(lines.size()/10.00);
            sumAccuracy = sumAccuracy + accuracy;
        }
        System.out.println("Average accuracy : " + String.format("%.3f" ,
sumAccuracy/10.00) + "\t\tTime: " + (System.currentTimeMillis() - time)/1000.00 + " sec");
    }
    //
**********************************************************************************
************************


    // **********************************'kNN-Nearest Neighbor classifier'
method*****************************************
    public static void kNN(ArrayList<String> lines){
        // k loop
        for (int k=1; k<10; k=k+2){
            long time = System.currentTimeMillis();

            double sumAccuracy = 0.00;
            // folds loop
            for (int f=0; f<10; f++){
                int success = 0;
                // 'new' samples loop
                for (int i=f*lines.size()/10; i<(f+1)*lines.size()/10; i++){
                    int listCounter = 0;
                    NeighbourSample head = null;

                    String sample[] = lines.get(i).trim().split("\\s+");
                    double x1 = Double.parseDouble(sample[0]);
                    double x2 = Double.parseDouble(sample[1]);
```

```
                    byte category = Byte.parseByte(sample[2]);

                    // finds the neighbour samples
                    int j = 0;
                    if (j == f*lines.size()/10){
                        j = j + lines.size()/10;
                    }
                    while (j<lines.size()){
                        sample  = lines.get(j).trim().split("\\s+");
                        double xn1 = Double.parseDouble(sample[0]);
                        double xn2 = Double.parseDouble(sample[1]);
                        byte categoryn = Byte.parseByte(sample[2]);

                        double distance = Math.sqrt((x1-xn1)*(x1-xn1) + (x2-xn2)*(x2-
xn2));

                        // makes a sorted linked list of k neighbour samples (keeps
relative distance and category)
                        NeighbourSample newSample = new NeighbourSample();
                        newSample.distance = distance;
                        newSample.category = categoryn;
                        newSample.next = null;
                        if (head == null){
                            head = newSample;
                        }
                        else {
                            if (newSample.distance >= head.distance){
                                newSample.next = head;
                                head = newSample;
                            }
                            else{
                                NeighbourSample currentSample = head;
                                while (currentSample.next != null){
                                    if ((newSample.distance <= currentSample.distance) &&
(newSample.distance >= currentSample.next.distance)){
                                        newSample.next = currentSample.next;
                                        currentSample.next = newSample;
                                        break;
                                    }
                                    currentSample = currentSample.next;
                                }
                                currentSample.next = newSample;
                            }
                        }
                        listCounter++;

                        if (listCounter > k){
                            head = head.next;
                            listCounter--;
                        }

                        j++;
                        if (j == f*lines.size()/10){
                            j = j + lines.size()/10;
                        }
                    }

                    // sums successes and measures accuracy
                    int sum = 0;
                    NeighbourSample currentSample = head;
                    while (currentSample != null){
                        sum = sum + currentSample.category;
                        currentSample = currentSample.next;
                    }
                    if ((sum > k/2 && category == 1) || (sum <= k/2 && category == 0)){
                        success++;
                    }
                }
                double accuracy = success/(lines.size()/10.00);
```

```
                sumAccuracy = sumAccuracy + accuracy;
            }
            System.out.println("Average accuracy for k = " + k + " : " +
String.format("%.3f" , sumAccuracy/10.00) + "\t\tTime: " + (System.currentTimeMillis() -
time)/1000.00 + " sec");
        }
    }
}

class NeighbourSample {
    public double distance;
    public byte category;

    public NeighbourSample next;
}

//*************************************************************************************
***********************************
```