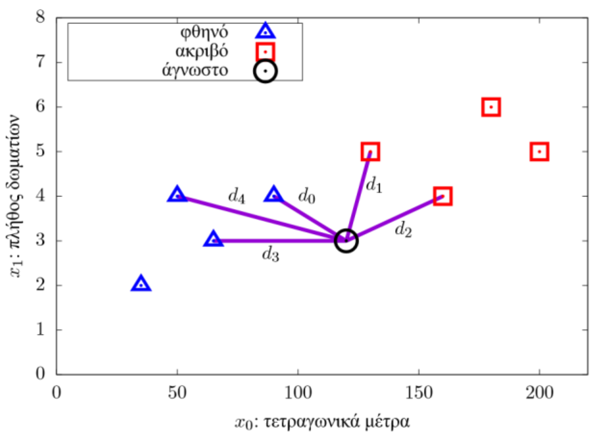
1 Description

The k-Nearest Neighbors (*k-NN)* method is used in Machine Learning for automatic categorization. It estimates the unknown category *y* (from a finite set of categories) corresponding to *n* measurements of *n* quantities, represented as an *n-dimensional* point

χμένα= (*nx-dimensional* measurement points0,...,*xn-1*). For the estimation, the method exploitsx(*i*), *i* = 0,...,*m* - 1. In the paper we will implement structure simplydata categories *y*(*i*) for *m data-*

linked list that supports *k-NN* method functions and experiment with it.

Η Μέθοδος k−NN με Παράδειγμα. Στο παρακάτω σχήμα απεικονίζονται 8 σημεία $x^{(i)} = (x_0^{(i)}, x_1^{(i)})$, *i* = 0,1,...,...,7, where x_0 measures the area of a house in square meters and x_1 measures the number of rooms. Houses marked with a 'triangle' are considered cheap, while those marked with a square are considered expensive.



Υπάρχοντα Δεδομένα

τ.μ. $x_0^{(i)}$	*Δωμάτια* $x_1^{(i)}$	*Κατηγορία* $y^{(i)}$
35	2	φθηνό
50	4	φθηνό
65	3	φθηνό
90	4	φθηνό
130	5	ακριβό
160	4	ακριβό
180	6	ακριβό
200	5	ακριβό

Από την περιγραφή $x = (x_0, x_1)$ ενός σπιτιού που είδαμε σε αγγελία, θέλουμε να εκτιμήσουμε αν κατηγοριοποιείται σε αυτά που θεωρούμε φθηνά, βάσει των δεδομένων μας, ή αν είναι ακριβό. Μία εκδοχή της μεθόδου k−NN εφαρμόζεται ως εξής: για δεδομένη τιμή του k, έστω $k = 5$, εντοπίζουμε τα k εγγύτερα σημεία $x^{(i)}$ στο x, ως προς τις Ευκλείδειες αποστάσεις τους. Έστω d_0, d_1, d_2, d_3, d_4 οι αποστάσεις των k εγγύτερων σημείων (δείτε το σχήμα). Μετράμε τη «βαρύτητα» κάθε κατηγορίας («φθηνό/ακριβό»), βάσει των αποστάσεων, ως εξής. Η βαρύτητα της κατηγορίας «φθηνό» υπολογίζεται ως το άθροισμα $\frac{1}{d_0} + \frac{1}{d_3} + \frac{1}{d_4}$, που προκύπτει από τις αποστάσεις των εγγύτερων γειτόνων που κατηγοριοποιούνται ως «φθηνοί». Αντίστοιχα, η βαρύτητα της κατηγορίας «ακριβό» υπολογίζεται ως το άθροισμα $\frac{1}{d_1} + \frac{1}{d_2}$. Με τον τρόπο αυτόν, καθένας από τους $k = 5$ εγγύτερους γείτονες συνεισφέρει «ψήφο» για το αγνώστου κατηγορίας σημείο, αντιστρόφως ανάλογη προς την απόστασή του από αυτό. Η κατηγορία του σημείου x ορίζεται ως εκείνη με τη μεγαλύτερη βαρύτητα.

Ευκλείδεια Απόσταση. Η Ευκλείδεια απόσταση $d(x, x')$, μεταξύ δύο n-διάστατων σημείων x και x' είναι:

$$d(x, x') = \sqrt{(x_0 - x'_0)^2 + (x_1 - x'_1)^2 + \cdots + (x_{n-1} - x'_{n-1})^2}$$

2 Job requests

2.1 Implementation

You will implement the `PointData`, `Node`, `PointList` and `KNN` classes, which are specified in the *given* `KNN.java` file. ***You should not use arrays in the*** implementation of the methods, except for representing *x* points where needed. You will make appropriate use of the `PointList` methods that you implement. ***You may only add*** `private` ***fields or methods***.

2.1.1 `PointData` and `Node` classes

An object of class `PointData` simply stores a point $x^{(i)}$, along with its categorization $y^{(i)}$, and provides a method of Euclidean distance `dist of` point $x^{(i)}$ from another given point x. To avoid errors, the `PointData` constructor should ***copy*** the input point `x` into the `this.x` field rather than assigning the reference with `this.x = x`.

```
class PointData {                              class Node {
  public double[] x;public double score; public double y;   public
  PointData ptd;
                                                 public Node next;
  public PointData(double[] x, double y)
  public double dist(double[] x)                 public Node(double[] x, double y)
}}
```

The constructor of the `Node` class creates a list node from a given point (and its class) by initializing the `ptd` field. The `Node` class contains a `score` field of type `double`, which can be used in the implementation of the `classify` method (described below).

2.1.2 Class `PointList`

It implements a simple linked list structure that provides the following methods.

`public PointList(int dim)`Constructs a list ofPointList classes that stores pairs(x,y),

where x is `dim`. `public static PointList readData(String filename)` Constructs and returns a new point list `readData(String filename)`.

in the `PointList` class, reading a set of data from a file named `filename` and inserting the data pairs into the list. This method is given implemented for your convenience. It is used as follows: `pointList list = PointList.readData("wine.dat");` `public int getDim()` Returns the dimension of the points stored in the list. `public int length()` Returns the number of nodes in the list. `public boolean append(double[] x, double y)` Inserts a copy of the input pair `x,y`, encapsulated in an object of the `PointData` class, at the end of the list and returns `true`. Returns `false` if `x` is of different length than the dimension of the points stored in the list. `public boolean append(PointList list)` Appends the contents of the input list `list`, to the

end of the list in which the method is called, *copying each* `PointData` class *point* contained in the `list`, without linking the list to the list in which the method is called[1]. Returns `false` if the two lists store points of different dimensions. Otherwise, it returns `true`. `public PointData rmFirst()` Deletes the first node of the list and returns a reference to

`PointData` class object that encapsulates the data. If the list is empty it returns `null`.

Note: ***All of the following methods can change the relative order in which the list data is stored.***

[1] Java hint: methods of a class have access to the `private` fields of all objects in the class.

`public void` *shuffle*`()` *"Shuffles"* randomly and uniformly the order in which they are stored

the `PointData` class data of the list. `public PointData rmNearest(double[] x)` Locates and deletes from the list the node that

stores the data of the nearest point to the entry point `x`. Returns a `PointData` class reference to this data, unless the list is empty, in which case it returns `null`.

`public PointList findKNearest(double[] x, int k)` For the given value of `k`, locates in λίστα τους `k` εγγύτερους γείτονες του σημείου εισόδου `x`. Εγγράφει σε νέα λίστα κλάσης `PointList` αυτούς τους *k* εγγύτερους γείτονες, την οποία επιστρέφει.

`public double classify(double[] x)` Classifies the entry point `x`, based on the points that αποθηκεύονται στη λίστα στην οποία καλείται η μέθοδος (θεωρώντας ότι *όλα* τα σημεία που περιέχει η λίστα είναι «εγγύτεροι γείτονες» του `x`). Η μέθοδος θα πρέπει να δουλεύει ορθά για *οσεσδήποτε* δια-φορετικές κατηγορίες εμφανίζονται στα περιεχόμενα της λίστας. Για την υλοποίησή της, θα σας φανεί useful the `public score` field, contained in the `Node` class. Like the other methods, it is expected to be implemented without the use of arrays. As long as you have properly implemented the `classify` and `findKNearest` methods, the *k-NN* algorithm for a point `x` and a given value of *k* (e.g., *k* = 5) is expressed in a single line: `list.findKNearest(x, 5). classify(x)?`

2.1.3 Class KNN

This will be the main class of your program, implementing the `main` method. The `main` method will perform the experimentation described in the following paragraph.

2.2 Experimentation and Technical Reference

Two (2) data files are given. For each file and for each value of $k \in \{1,2,\ldots,10\}$, repeat the following procedure 10 times. You will create two lists of the `PointList` class. One list ("small") will contain a randomly and uniformly selected 25% of the data in the file, and the other will contain the remaining 75% ("large"). You will then measure the categorization accuracy of the *k-NN* algorithm for the current value of *k* as the percentage of points in the "small" list that are correctly categorized by the method. More specifically, you will categorize each point in the "small" list with the algorithm, using the points in the "large" list, and measure the percentage of points in the "small" list that are correctly categorized. Your program will print the *average* categorization *accuracy* for the current data file and the current value of *k*, calculated over the 10 iterations.

You will write a technical report in which you will show and comment on the results of your experiment. In the report, you will include tables, one for each data file, recording the average categorisation accuracy of the algorithm for each different value of *k*. In addition, you will produce bar charts illustrating the results of the tables (one for each data file).

The technical report will include a brief implementation description, for each of the `shuffle`, `rmNearest`, `findKNearest`, `classify` methods of the `PointList` class - with pseudocode and/or explanations where appropriate. Based on your description, you will analyze the complexity of your implementation for each of these methods, in terms of the parameters: *list length m (or k, where appropriate), point dimension n.* In conclusion, you will derive and justify the complexity of your implementation of the *k-NN* algorithm.

Data Files. They are simply text files. On the 1st line they contain the number m of data lines and the dimension n of the points $x^{(i)}$. This is followed by one line for each pair ($x^{(i)}$,$y^{(i)}$), containing $n+1$ real numbers, separated by a blank character: The first n numbers correspond to the coordinates of $x^{(i)}$. The last number corresponds to the category y. $^{(i)}$The dataset in `wine.dat` concerns the categorisation of wine into three classes. The dataset in `housing.dat` concerns the categorisation of houses into 'cheap/expensive'.