ECE/CPSC 3520
SS II 2019
Software Design Exercise #2
Implementing The 1NNR Algorithm in Prolog Using a Euclidean Distance (Squared) Measure

Canvas submission only

Assigned 7/19/2019

Due 7/29/2019 11:59PM

Contents

1 Preface

The overall objective of SDE2 is to implement the 1-NNR algorithm in `Prolog`, using a specified distance measure. Your grade is based upon a correctly working implementation which is submitted by the deadline.

A number of predicates to support the 1-NNR computation in Prolog are to be designed, implemented and tested. I have designed this SDE to force (or at least encourage) incremental development of the predicates. This means in the order they are presented in Section 5.1. Also, to be successful, you should deal with the training set recursively (by vector or sublist) and vectors recursively (by element).

2 Resources

It would be foolish to attempt this SDE without carefully exploring:

1. The text, especially the many examples in Chapter 3;
2. `http://www.swi-prolog.org/`;
3. The `Prolog` manual;
4. The course Prolog lectures and corresponding videos on Canvas; and
5. SDE1.

3 Background

3.1 Brief Description of 1-NNR

The k-NNR approach to pattern classification is straightforward and previously described in SDE 1. Basically, it assumes that 'vectors that are close in distance are close in class'. Using 1-NNR (k=1), a given vector, $\underline{x}$, is classified by finding, in a **labeled**[1] **training set of vectors**, H, , the closest vector to $\underline{x}$ and assigning the class of the closest vector to $\underline{x}$. An extension is to find the 3 and 5-nearest neighbors of $\underline{x}$ and then assign $\underline{x}$ to the class **most**

[1]Labeled means where the class of each vector is known.

represented in these neighbors, **but that extension is not considered here**.

Our goal is to implement a very unsophisticated version of 1-NNR using declarative programming and `Prolog`. We find the 1 nearest vector of $\underline{x}$ by brute force, i.e., by exhaustive computation of distance to all vectors in H. As noted in SDE 1, there are more elegant and computationally efficient approaches.

3.2 How Do You Define (and Implement) 'Close'?

In this effort, we will use the Euclidean distance measure (squared) as the measure of vector closeness. This was described in SDE1 and is done by computing the difference (vector) between two given vectors, squaring each element of the difference vector and summing the result.

4 Prolog Data Representation ('Where are the Lists?')

The desired `Prolog` input format for H is that of a `Prolog` list of lists format. **Each list in H represents a vector**, i.e., $\underline{x}$, with known class information appended (i.e., the last element). There are up to 10 possible classes; **class values are integers in the interval from 1 to 10**.

Unlike `ocaml`, it is not necessary for elements of Prolog lists to have a common type. Here is an example of a simple database with some test vectors[2]:

```
/* data for simple 4 class case */

simpleH([[5,5,1],[5,-5,2],[-5,-5,3],[-5,5,4]]).

/* some test vectors */

t1([3,3,0]).
t2([-1,-1,0]).
t3([0,0,0]).
t4([-1,1,0]).
t5([0,-1,0]).
```

[2]It should look familiar.

Here is an example of a simple use of this database:

```
?- simpleH(What).
What = [[5, 5, 1], [5, -5, 2], [-5, -5, 3], [-5, 5, 4]].

?- t5(What).
What = [0, -1, 0].
```

Like SDE1, the training set, H, is just a list of lists, where each list represents a vector **and the last element of each vector is the corresponding class. Thus, the distance between two vectors computation does not use this last element of each.** The number of lists, or the size of each list, i.e., the vector dimension plus 1 for the class information, is variable. **Your implementation must accommodate this**.

5 Implementing 1-NNR in `Prolog`

The overall objective and major part of your effort is to implement the 1-NNR algorithm in `Prolog`. This is via a predicate named `nnr1`. First, however, we digress to specify a set of predicates which must be developed **and which will be tested along with `nnr1`**.

We identify and structure of the declarative programming solution and specify prerequisite predicates and their prototypes below. Note that **all prototypes should be capable of working with arbitrary-length lists**. As indicated, do not hard-code your solution for a specific vector dimension (list length) or size of H.

5.1 1NNR Predicates, Prototypes and Sample Use

I've broken up some of the design into guided steps, as shown below. **You must design and implement all of these predicates with the prototypes and behavior shown.** Of course, there may also be other predicates to be developed.

5.2 Predicate Specification

The predicate prototype specification follows Prolog convention and the book. First, the arity of the predicate is specified by a forward slash (/) and an integer. The arguments are specified by:

+Argument means the argument is bound before invoking the predicate;

-Argument means the argument is bound to a value, if possible, in a solution; and

?Argument means either (The argument functions in both roles).

Notes: **This notation is for specification only**; the +,-, or ? designation or the arity specification is not used when the predicate is actually invoked. See the examples.

5.3 Individual Prototypes

Your Prolog implementation must include all the predicates in the following sections. It is probably redundant, but nevertheless worthwhile, to note the names (case matters!) and arity must be as indicated.

5.3.1 `printList/1`

Prototype.

```
printList(+H)

Notes:
A supporting predicate for training set (H) display.

Display the list-of-lists in H by printing the vectors (lists)
one per line. The examples help visualization of this.
This predicate has a side effect.
```

Sample Use.

```
?- simpleH(List),printList(List).

[5,5,1]
[5,-5,2]
[-5,-5,3]
[-5,5,4]
List = [[5, 5, 1], [5, -5, 2], [-5, -5, 3], [-5, 5, 4]].
```

5.3.2 `theClass/2`

Prototype.

```
theClass(+Avect,-C).

Given a vector, Avect in list format, with the known class as the last element,
C is the class.
Notes: Does not need to check of class is in the interval [1,10].
```

Sample Use.

```
?- theClass([1,2,3,4,5], What).
What = 5.

?- theClass([-5,-5,4],WhatClass).
WhatClass = 4.
```

5.3.3 `distanceR2/3`

Digression. Recall our Prolog representation of 2 lists consists of n elements (n is arbitrary, but ≥ 1) plus a $n+1^{st}$ element representing the class (or 0 if the class is not known). For example, given 2 lists (or vectors) V1 and V2 with structure as follows:

```
V1 = [a1, a2, ..., an, classV1]

and

V2 = [b1, b2, ..., bn, classV2]
```

the Euclidean distance (squared) we compute and use (the value of `DistSq` in predicate `distanceR2`, below, would be (recursively) computed as:

$$DistSq = \Sigma_{i=1}^{i=n}(a_i - b_i)^2 \tag{1}$$

Notice the class is not used in the distance computation.

Prototype.

```
distanceR2(+V1,+V2,-DistSq)

   This is the vector distance computation --
   returns Euclidean distance squared (^2) between given vectors V1 and V2 as value of DistSq.
   REMEMBER: only uses n-1 elements of vector in computation; last element denotes class.
```

Sample Use. You should validate these responses, by hand and with your Prolog implementation.

```
/* this is from the video */

?- distanceR2([1,2,0],[5,5,0],What).
What = 25.0.

/* here's some others */

?- t1(V1),t2(V2),distanceR2(V1,V2,D2).
V1 = [3, 3, 0],
V2 = [-1, -1, 0],
D2 = 32.0.

?- t1(V1),t3(V2),distanceR2(V1,V2,D2).
V1 = [3, 3, 0],
V2 = [0, 0, 0],
D2 = 18.0.
```

5.3.4 `distanceAllVectors2/3`

Prototype. Now we're getting closer to 1-NNR.

```
distanceAllVectors2(+V,+Vset,-Dlist)

/* Dlist is distance^2 of V to all vectors in Vset */
/* Dlist is list of distances^2 indexed the same as Vset */
```

Sample Use. You should also validate these responses, by hand and with your Prolog implementation.

```
?- simpleH(List),distanceAllVectors2([5,5,0],List,Distances2).
List = [[5, 5, 1], [5, -5, 2], [-5, -5, 3], [-5, 5, 4]],
Distances2 = [0.0, 100.0, 200.0, 100.0].

?- simpleH(List),distanceAllVectors2([0,0,0],List,Distances2).
List = [[5, 5, 1], [5, -5, 2], [-5, -5, 3], [-5, 5, 4]],
Distances2 = [50.0, 50.0, 50.0, 50.0].

?- simpleH(H),t1(V),distanceAllVectors2(V,H,Dists).
H = [[5, 5, 1], [5, -5, 2], [-5, -5, 3], [-5, 5, 4]],
V = [3, 3, 0],
Dists = [8.0, 68.0, 128.0, 68.0].
```

5.3.5 `nnr1/3`

Prototype. This is the top-level prototype, and probably the most difficult. Again, you should validate these responses.

```
nnr1(+Test,+H,-Class)
```

Sample Use.

```
?- simpleH(H),nnr1([-25,1,0],H,TheClass).
H = [[5, 5, 1], [5, -5, 2], [-5, -5, 3], [-5, 5, 4]],
TheClass = 4.

?- simpleH(H),nnr1([-25,-1,0],H,TheClass).
H = [[5, 5, 1], [5, -5, 2], [-5, -5, 3], [-5, 5, 4]],
TheClass = 3.

?- simpleH(H),nnr1([25,-1,0],H,TheClass).
H = [[5, 5, 1], [5, -5, 2], [-5, -5, 3], [-5, 5, 4]],
TheClass = 2.
```

5.4 Additional Notes

1. Pay particular attention to the names of predicates shown, their arity (and argument order) and the data structures used.
2. **The input vector dimension (list length) and number of vectors is variable, but fixed for any given data set. Your predicates must accommodate this**.
3. Recall **Prolog is case sensitive.**
4. Notice you must work with the specified list data structures, i.e., you cannot redefine the formats of these predicates to suit your needs or desires.
5. Finally, recall that all vectors (in addition to training set vectors) have class information appended. For vectors of unknown class (to be classified with `nnr1`, we set the class to be 0. *The classification of input vectors is done via 1-NNR, not simply reading the last element.*

5.5 A Second Set of Samples

Note the change in the size of H and the dimension of the vectors. Your Prolog implementation must accommodate this.

The Database.

```
/* larger dataset for testing */

wcdata([
[3.26495e+02, 6.00000e+00, 2.82447e+02, 1.10000e+01, -3.61804e+01, -1.17557e+01, 1],
[5.19186e+02, 1.00000e+00, 4.57228e+01, 4.70000e+01, -3.61804e+01, -1.17557e+01, 1],
[7.03244e+01, 5.00000e+00, 2.15593e+01, 2.80000e+01, -2.23607e+01, 3.07768e+01, 1],
[1.87247e+02, 1.00000e+00, 7.03097e+01, 1.80000e+01, -2.23607e+01, 3.07768e+01, 1],
[4.26179e+02, 6.00000e+00, 1.33301e+02, 2.20000e+01, 3.61804e+01, 1.17557e+01, 7],
[1.97036e+03, 5.00000e+00, 7.51209e+02, 1.90000e+01, 3.11803e+01, 1.90212e+01, 7],
[1.28191e+03, 1.00000e+00, 6.08078e+02, 2.20000e+01, 1.78214e+01, 1.08487e+01, 7],
[4.82068e+02, 4.00000e+00, 4.71010e+02, 2.00000e+01, -2.23607e+01, 3.07768e+01, 1],
[9.24921e+02, 7.00000e+00, 3.40959e+02, 1.60000e+01, 3.61804e+01, 1.17557e+01, 1],
[4.85993e+02, 7.00000e+00, 3.91866e+02, 2.10000e+01, -2.23607e+01, 3.07768e+01, 1],
[1.85204e+02, 4.00000e+00, 5.22527e+01, 2.40000e+01, -2.23607e+01, 3.07768e+01, 5],
[2.54153e+02, 2.40000e+01, 1.86337e+02, 1.00000e+00, 1.40018e+01, 2.98699e+01, 5],
[3.08718e+02, 1.00000e+01, 1.56700e+02, 2.00000e+01, -2.23607e+01, -3.07768e+01, 5],
[1.28083e+02, 8.00000e+00, 1.33937e+01, 5.40000e+01, 0.00000e+00, -1.00000e+01, 4],
[1.44025e+02, 6.00000e+00, 2.34110e+01, 1.60000e+01, -2.23607e+01, 3.07768e+01, 4],
[1.90704e+02, 6.00000e+00, 2.35136e+01, 1.80000e+01, 1.40018e+01, -2.98699e+01, 4],
[1.74306e+02, 1.20000e+01, 5.55195e+01, 2.20000e+01, -1.40018e+01, -2.98699e+01, 5],
[2.01648e+02, 1.00000e+00, 1.35576e+02, 4.90000e+01, 3.61804e+01, -1.17557e+01, 5],
[5.38501e+01, 6.00000e+00, 4.88364e+01, 2.50000e+01, 0.00000e+00, -1.00000e+01, 5],
[1.50294e+02, 7.00000e+00, 2.42179e+01, 2.40000e+01, 0.00000e+00, -1.00000e+01, 5]]).

tvc([98.641683, 17.0, 75.387416, 9.0, -36.18035, 11.7557, 0]).

tv2c([1869.2331, 5.0, 184.885, 27.0, 17.82145, 10.8487, 0]).
```

Selected Queries.

```
?- wcdata(What),printList(What).

[326.495,6.0,282.447,11.0,-36.1804,-11.7557,1]
[519.186,1.0,45.7228,47.0,-36.1804,-11.7557,1]
[70.3244,5.0,21.5593,28.0,-22.3607,30.7768,1]
[187.247,1.0,70.3097,18.0,-22.3607,30.7768,1]
[426.179,6.0,133.301,22.0,36.1804,11.7557,7]
[1970.36,5.0,751.209,19.0,31.1803,19.0212,7]
```

```
[1281.91,1.0,608.078,22.0,17.8214,10.8487,7]
[482.068,4.0,471.01,20.0,-22.3607,30.7768,1]
[924.921,7.0,340.959,16.0,36.1804,11.7557,1]
[485.993,7.0,391.866,21.0,-22.3607,30.7768,1]
[185.204,4.0,52.2527,24.0,-22.3607,30.7768,5]
[254.153,24.0,186.337,1.0,14.0018,29.8699,5]
[308.718,10.0,156.7,20.0,-22.3607,-30.7768,5]
[128.083,8.0,13.3937,54.0,0.0,-10.0,4]
[144.025,6.0,23.411,16.0,-22.3607,30.7768,4]
[190.704,6.0,23.5136,18.0,14.0018,-29.8699,4]
[174.306,12.0,55.5195,22.0,-14.0018,-29.8699,5]
[201.648,1.0,135.576,49.0,36.1804,-11.7557,5]
[53.8501,6.0,48.8364,25.0,0.0,-10.0,5]
[150.294,7.0,24.2179,24.0,0.0,-10.0,5]
What = [[326.495, 6.0, 282.447, 11.0, -36.1804|...], [...|...]|...].

?- tvc(V1), tv2c(V2), distanceR2(V1,V2,D).
V1 = [98.641683, 17.0, 75.387416, 9.0, -36.18035, 11.7557, 0],
V2 = [1869.2331, 5.0, 184.885, 27.0, 17.82145, 10.8487, 0],
D = 3150368.703908144 .

?- tvc(V1), theClass(V1,Huh).
V1 = [98.641683, 17.0, 75.387416, 9.0, -36.18035, 11.7557, 0],
Huh = 0.

?- wcdata(H), distanceAllVectors2([1.90704e+02, 6.00000e+00, 2.35136e+01,
               1.80000e+01, 1.40018e+01, -2.98699e+01, 0],H,Dlist),printList(Dlist).

88381.07875504001
112106.05032712001
19596.32101179
7227.079451350001
69742.32969707999
3699403.5140386196
1534159.6659134803
290154.2621361
642075.7659434801
227889.33773389994
5896.4394959500005
34719.0788926
33008.97521482
5914.665346260001
7183.193194900001
0.0
2129.4736517699994
```

```
14483.767169359999
20010.097474300004
2261.32746774
H = [[326.495, 6.0, 282.447, 11.0, -36.1804, -11.7557, 1], [519.186, 1.0, 45.7228, 47.0,
-36.1804, -11.7557, 1], [70.3244, 5.0, 21.5593, 28.0, -22.3607, 30.7768|...],
[187.247, 1.0, 70.3097, 18.0,-22.3607|...], [426.179, 6.0, 133.301, 22.0|...],
[1970.36, 5.0, 751.209|...], [1281.91, 1.0|...], [482.068|...], [...|...]|...],
Dlist = [88381.07875504001, 112106.05032712001, 19596.32101179, 7227.079451350001,
69742.32969707999, 3699403.5140386196, 1534159.6659134803, 290154.2621361,
642075.7659434801|...].

?- wcdata(H), nnr1([1.28083e+02, 8.00000e+00, 1.33937e+01, 5.40000e+01, 0.00000e+00,
-1.00000e+01, 0],H,TheClass).
H = [[326.495, 6.0, 282.447, 11.0, -36.1804, -11.7557, 1], [519.186, 1.0, 45.7228, 47.0,
 -36.1804,-11.7557, 1], [70.3244, 5.0, 21.5593, 28.0, -22.3607, 30.7768|...], [187.247,
1.0, 70.3097, 18.0, -22.3607|...], [426.179, 6.0, 133.301, 22.0|...], [1970.36, 5.0,
751.209|...], [1281.91, 1.0|...], [482.068|...], [...|...]|...],
TheClass = 4.

?- wcdata(H), nnr1([1.28191e+03, 1.00000e+00, 6.08078e+02, 2.20000e+01, 1.78214e+01,
                    1.08487e+01, 0],H,TheClass).
H = [[326.495, 6.0, 282.447, 11.0, -36.1804, -11.7557, 1], [519.186, 1.0, 45.7228, 47.0,
 -36.1804, -11.7557, 1], [70.3244, 5.0, 21.5593, 28.0, -22.3607, 30.7768|...], [187.247,
1.0, 70.3097, 18.0, -22.3607|...], [426.179, 6.0, 133.301, 22.0|...], [1970.36, 5.0,
751.209|...], [1281.91, 1.0|...], [482.068|...], [...|...]|...],
TheClass = 7.

?- wcdata(H), nnr1([9.24921e+02, 7.00000e+00, 3.40959e+02, 1.60000e+01, 3.61804e+01,
                    1.17557e+01, 0],H,TheClass).
H = [[326.495, 6.0, 282.447, 11.0, -36.1804, -11.7557, 1], [519.186, 1.0, 45.7228, 47.0,
 -36.1804, -11.7557, 1], [70.3244, 5.0, 21.5593, 28.0, -22.3607, 30.7768|...], [187.247,
1.0, 70.3097, 18.0, -22.3607|...], [426.179, 6.0, 133.301, 22.0|...], [1970.36, 5.0,
751.209|...], [1281.91, 1.0|...], [482.068|...], [...|...]|...],
TheClass = 1.

?- wcdata(H), nnr1([98.641683, 17.0, 75.387416, 9.0, -36.18035, 11.7557, 0],H,TheClass).
H = [[326.495, 6.0, 282.447, 11.0, -36.1804, -11.7557, 1], [519.186, 1.0, 45.7228, 47.0,
 -36.1804, -11.7557, 1], [70.3244, 5.0, 21.5593, 28.0, -22.3607, 30.7768|...], [187.247,
1.0, 70.3097, 18.0, -22.3607|...], [426.179, 6.0, 133.301, 22.0|...], [1970.36, 5.0,
751.209|...], [1281.91, 1.0|...], [482.068|...], [...|...]|...],
TheClass = 1.
```

6 How We Will Build and Evaluate Your Prolog Solution

First, we'll consult your Prolog source file to identify any warnings or errors. Then, a Prolog script will be used with varying input files and test vectors to test your predicates.

BE SURE YOUR PREDICATES CONFORM TO THE SPECIFIED PROTOTYPES!

We will not rewrite or revise your submission to facilitate grading. The grade is based upon a correctly working solution.

7 Format of the Electronic Submission

The final **zipped** archive is to be named `<yourname>`**-sde2.zip**, where `<yourname>` is your (CU) assigned user name. You will upload this to the Canvas assignment prior to the deadline.

The minimal contents of this archive are as follows:

1. A `readme.txt` file listing the contents of the archive and a brief description of each file. Include 'the pledge' here. Here's the pledge:

   > **Pledge:**
   > On my honor I have neither given nor received aid on this exam.

   This means, among other things, that the code you submit is **your** code.

2. The `Prolog` source file for your implementation (named `nnr1.pro`). Note this file must include the predicates defined in Section 5.1, as well as any additional predicates you design and implement. This is the file our Prolog scripts will consult.

3. An ASCII log file showing 2 sample uses of each of the predicates required in Section 5.1. Name this log file `nnr1.log`.

The use of `Prolog` should not generate any errors or warnings. The grader will attempt to interpret your Prolog source and check for correct behavior. Recall the grade is based upon a correctly working solution.

8 Final Remark: Deadlines Matter

This remark was included in the course syllabus and SDE1. Since multiple submissions to Canvas are allowed[3], if you have not completed all functions, you should submit a freestanding archive of your current success before the deadline. This will allow the possibility of partial credit. **Do not attach any late submissions to email and send them to either me or the graders.**

[3]But we will only download and grade the latest (or most recent) one, and it must be submitted by the deadline.