ECE/CPSC 3520
SS II 2019
Software Design Exercise #1
Implementing The 1NNR Algorithm in Functional Programming Using `ocaml` and a Euclidean Distance (Squared) Measure

Canvas Submission Only

Assigned 7/5/2019

Due 7/16/2019 11:59PM

Contents

1 Preface **3**

2 Resources **3**

3 Implementing Basic 1NNR in `ocaml` **3**
- 3.1 Brief Description of 1-NNR 3
- 3.2 How Do You Define (and Implement) 'Close'? 4
- 3.3 `ocaml Data Structures` 4
- 3.4 The Distance (Squared) Measure of Closeness 5

4 Notes on Implementing 1NNR in `ocaml` **5**

5 1NNR Functions, Prototypes, Signatures, Semantics and Sample Use **6**
- 5.1 `printList` 6
- 5.2 `theClass` 6
- 5.3 `distanceR2` 6
- 5.4 `distanceAllVectors2` 7
- 5.5 `nnr1` 7

6 Sample Uses of the Required `ocaml` Functions **7**

7 `ocaml` Functions and Constructs Not Allowed **9**

8 How We Will Build and Evaluate Your ocaml Solution **10**

9 Format of the Electronic Submission **10**

10 Final Remark: Deadlines Matter **11**

1 Preface

The overall objective of SDE1 is to implement the 1NNR algorithm in `ocaml` using a purely functional programming paradigm.

2 Resources

As mentioned previously, it would be foolish to attempt this SDE without first carefully studying:

1. The text, especially the many examples in Chapter 11;
2. The ocaml course lectures (slides/videos);
3. The `ocaml` manual; and
4. The related SDE1 videos (2).

3 Implementing Basic 1NNR in `ocaml`

The overall objective and major part of your effort is to implement the 1NNR algorithm in `ocaml`. The a basic list structure and Euclidean distance measure (squared) from SDE1 is reused here. Initially, we identify and structure the functional programming solution and specify prerequisite functions.

3.1 Brief Description of 1-NNR

The k-NNR approach to pattern classification is straightforward. 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 represented** in these neighbors, **but that extension is not considered here**.

Our goal is to implement a very unsophisticated version of 1-NNR using functional programming and `ocaml`[2]. We find the 1 nearest neighbor of vector

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

[2]In SDE2, we will accomplish the same task using declarative programming and `Prolog`.

$\underline{x}$ by brute force, i.e., by exhaustive computation of distance to all vectors in H. 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 is done by computing the difference (vector) between two given vectors, squaring each element of the difference vector and summing the result.

3.3 `ocaml` Data Structures

The desired `ocaml` input format for training set H is that of an `ocaml` list of lists (of type float) format[3]. **Each (sub)list in H represents a vector** with known class information appended (i.e., the last element) as a single float. There are up to 10 possible classes. **Appended class values are members of the set of floating point numbers {1.0, 2.0, 3.0,... 10.0}**.

Below is an example of a simple database with some test vectors (used in the accompanying introductory SDE1 video):

```
(* data for simple 4 class case *)

let simpleH = [[5.;5.;1.];[5.;-5.;2.];[-5.;-5.;3.];[-5.;5.;4.]];;

(* some test vectors *)


let t1 = [3.;3.;0.];;
let t2 = [-1.;-1.;0.];;
let t3 = [0.;0.;0.];;
let t4 = [-1.;1.;0.];;
let t5 = [0.;-1.;0.];;
```

As noted, the last element of each vector is the corresponding class. Thus, the computation of distance (squared) between two vectors **does not use this last element**. **The number of lists in H, or the size of each list, is variable[4]. Your implementation must accommodate this**.

[3]`ocaml` shows this with the signature `float list list`.

[4]Although it will be the same for all elements of a given training set, H.

3.4 The Distance (Squared) Measure of Closeness

Recall our representation of 2 lists (vectors) consists of n elements (n is arbitrary, but ≥ 1) plus a $n+1^{st}$ element representing the class (or 0.0 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) between these lists or vectors would be (recursively) computed as:

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

Notice (again) the class data contained in the last element is not used in the distance computation. `DistSq` is the value returned by `ocaml` function `distanceR2`.

4 Notes on Implementing 1NNR in `ocaml`

I've broken up some of the design into guided steps, as shown below. **You must design and implement these functions with the behavior shown.** Of course, you may also develop other 'auxiliary' functions to support these.

Notes:

1. Pay particular attention to the names, prototypes and signatures of the functions shown and the `ocaml` data structures used.

2. Recall **ocaml is case sensitive.**

3. Notice you must work with the specified list data structures, i.e., you cannot redefine the input and output formats (or signatures) of these functions to suit your needs or desires.

4. Finally, recall that input vectors (in addition to training set vectors) have class information appended.

The overall objective and major part of your effort is to implement the 1-NNR algorithm in `ocaml`. This is via a top-level function named `nnr1`. First we specify a set of functions which must be developed and which will be tested **along with** `nnr1`.

Note that **all functions 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 1NNR Functions, Prototypes, Signatures, Semantics and Sample Use

Listed below are the functions you will design, implement and test. **Notice all required functions use tuples for argument interface.**

5.1 `printList`

Prototype and Signature.

```
printList(ListofLists)

val printList : float list list -> unit = <fun>

See the examples for the use of printList. It is the only function
you will develop with a side effect (printing).
```

5.2 `theClass`

Prototype and Signature.

```
theClass(Avect).

val theClass : 'a list -> 'a = <fun>

Given a vector, Avect, in list format, with the known class as the last element, function
theClass returns the class.
Notes: Does not need to check if class is  a member of the discrete set [1.,10.].
```

5.3 `distanceR2`

Prototype and Signature.

```
distanceR2(V1,V2)

val distanceR2 : float list * float list -> float = <fun>

This is the vector distance computation: it returns Euclidean distance
squared (^2) between given vectors V1 and V2.
REMEMBER: distanceR2 only uses n-1 elements of vector in computation;
the last element denotes class.
```

5.4 distanceAllVectors2

Prototype and Signature.

```
distanceAllVectors2(v, vset)

val distanceAllVectors2 : float list * float list list -> float list = <fun>

Returns a list of distances from v to each member of vset.
```

5.5 nnr1

Prototype and Signature.

```
nnr1(Test,H)

val nnr1 : float list * float list list -> float = <fun>

This is the top-level function, and probably the most challenging.
Given a vector, Test, and a training set, H, function nnr1 returns the class
of the 1-nearest neighbor of Test in H using the Euclidean (squared) distance measure.
You should validate the responses shown in this document.
```

6 Sample Uses of the Required ocaml Functions

```
(* data for simple 4 class case *)

let simpleH = [[5.;5.;1.];[5.;-5.;2.];[-5.;-5.;3.];[-5.;5.;4.]];;

(* some test vectors *)

let t1 = [3.;3.;0.];;
let t2 = [-1.;-1.;0.];;
let t3 = [0.;0.;0.];;

# printList simpleH;;
5.  5.  1.
5.  -5.  2.
-5.  -5.  3.
-5.  5.  4.

# theClass([1.;2.;3.;4.;5.]);;
- : float = 5.

# theClass([-5.;-5.;4.]);;
- : float = 4.

# distanceR2([1.;2.;0.],[5.;5.;0.]);;
- : float = 25.

# distanceR2(t1,t2);;
- : float = 32.

# distanceR2(t1,t3);;
- : float = 18.

# distanceAllVectors2([5.;5.;0.],simpleH);;
- : float list = [0.; 100.; 200.; 100.]

# distanceAllVectors2([0.;0.;0.],simpleH);;
- : float list = [50.; 50.; 50.; 50.]

# distanceAllVectors2(t1,simpleH);;
- : float list = [8.; 68.; 128.; 68.]

# distanceAllVectors2([-25.;1.;0.],simpleH);;
- : float list = [916.; 936.; 436.; 416.]

# nnr1([-25.;1.;0.],simpleH);;
- : float = 4.

# nnr1([-25.;-1.;0.],simpleH);;
- : float = 3.

# nnr1([25.;-1.;0.],simpleH);;
- : float = 2.
```

7 `ocaml` Functions and Constructs Not Allowed

> Of extreme significance is the restriction of the paradigm to pure functional programming (no side effects). **No `ocaml` imperative constructs are allowed.** Recursion must dominate the function design process.

So that we may gain experience with functional programming, *only the applicative (functional) features of* `ocaml` *are to be used.* **Please reread the previous sentence.** This rules out the use of `ocaml`'s imperative features. See Section 1.5 'Imperative Features' of the manual for examples of constructs not to be used. **To force you into a purely applicative style, `let` should be used only for function definition**. `let` cannot be used in a function body. Loops and local or global variables are prohibited.

The only module you may use is the List module. (To help you develop a recursive style of programming, I recommend against using list iterators). This means you may not use the Array Module.

Finally, **the use of sequence (6.7.2 in the ocaml manual) is not allowed**. Do not design your functions using sequential expressions or begin/end constructs. Here is an example of a sequence in a function body:

```
let print_assignment = function(student,course,section) ->
print_string student; (* first you evaluate this*)
print_string " is assigned to "; (* then this *)
print_string course;  (* then this *)
print_string " section " ; (* then this *)
print_int section;  (* then this *)
print_string "\n;; (* then this and return unit*)
```

> *If you are in doubt, ask and I'll provide a 'private-letter ruling'.*

The objective is to obtain proficiency in functional programming, not to try to find built-in `ocaml` functions or features which simplify or trivialize the effort.

8 How We Will Build and Evaluate Your ocaml Solution

An `ocaml` script with varying input files and test vectors are used to test the 5 required functions in Section 5. The grade is based upon a correctly working solution.

9 Format of the Electronic Submission

The final **zipped** archive is to be named `<yourname>`**-sde1.zip**, where `<yourname>` is your (CU) assigned user name. You will upload this archive to the Canvas assignment Content item 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 `ocaml` source for your implementation in a single file named `nnr1.caml`. Note this file must include all the required functions, as well as any additional functions you design and implement. We will supply the testing data.

3. An ASCII log file showing 2 sample uses of each of the functions required in Section 5. Name this log file `nnr1.log`.

The use of `ocaml` should not generate any errors or warnings. The grader will attempt to interpret your ocaml source and check for correct functionality. We will also look for offending `let` use and sequences. Recall the grade is based upon a correctly working solution.

10 Final Remark: Deadlines Matter

This remark is included in the course syllabus. Since multiple submissions to Canvas are allowed[5], 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.**

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