ECE/CPSC 352
Fall 2018
Software Design Exercise #2

Canvas submission only

Assigned 10-3-2018 (early); Due 10-31-2018 11:59 PM

Contents

1 Preface

1.1 Objectives

The objective of SDE 2 is to implement an `ocaml` version of the c-means algorithm. Of extreme significance is the restriction of the paradigm to pure functional programming, i.e., **no imperative constructs are allowed.** We will specify a number of functions which **must** be developed as part of the effort.

This effort is straightforward, and 4 weeks are allocated for completion. The motivation is to:

- Learn the paradigm of (pure) functional programming;
- Implement a (purely) functional version of an interesting algorithm;
- Deliver working functional programming-based software based upon specifications; and
- Learn `ocaml`.

1.2 Resources

As discussed in class, it would be foolish to attempt this SDE without carefully exploring:

1. The text, especially the many ocaml examples in Chapter 11;
2. The `ocaml` lectures;
3. The background provided in this document;
4. **The background provided in** *Background on the c-means Algorithm for CPSC/ECE 3520***, available on Canvas**;
5. **Class discussions and examples**; and
6. The `ocaml` reference manual.

1.3 Standard Remark

Please note:

1. **This assignment (which counts as a quiz) tests *your* effort (not mine)**. I will not debug or design your code, nor will I install software for you. You (and only you) need to do these things.
2. It is never too early to get started on this effort.
3. We will continue to discuss this in class.

2 Preliminary Considerations

2.1 From Vectors to (`ocaml`) Lists

The companion document described the c-means algorithm using (column) vectors. In this SDE, all vectors will be represented as rows of `ocaml` lists. For example, the vector

$$\underline{x} = \begin{pmatrix} x1 \\ x2 \\ x3 \\ x4 \\ ... \\ xN \end{pmatrix}$$

becomes, in `ocaml` list form:

```
[x1; x2; x3; x4; ... xN]
```

Notice the dimension of the vector is the length of the list.

2.2 The Training Set, H

Based upon the algorithm description, we first consider the necessary, basic list-based data structures. The first is that of the training set.

Assume that the list-based data structure for H and the initial mean vectors, $\underline{\mu}_i(0)$, have been chosen. Below is a 2-D example[1] of H:

```
let h =[
[47.698002; 62.480000];
[-49.005001; -41.327999];
[45.958000; 29.403000];
[-60.546001; -50.702000];
[45.403000; 52.994999];
[-49.021000; -52.053001];
[29.788000; 58.993000];
[-40.433998; -36.362999];
...
```

Notice h is a 'list-of-float-lists' structure[2]. **In SDE2, H may consist of any number of vectors of arbitrary (but fixed) dimension. Be sure to keep this in mind as you develop functions**. The list-of-list form of the set of vectors comprising $\underline{\mu}_i(0)$ is similar. Sample `h` (with various names, dimensions and numbers of vectors) are given on the course Canvas page.

[1] Notice H is used to designate the training set, but in `ocaml` the symbol `h` is used. Can you figure out why?

[2] In fact, `ocaml` says the signature is `val h : float list list`

2.3 Initial Means

There are many ways to generate the set of initial means for the algorithm. They include:

1. **Random initialization**. In this technique, each of the c $\underline{\mu}_i(0)$ is a vector of randomly generated floating point components. The dimension of the vectors is determined by the dimension of the vectors in H. Notice this technique only uses `h` to determine the dimension of the vectors. The later 2 techniques actually return elements of `h` as the initial means.

2. Assuming the cardinality of H is n, another random approach is to use a RNG to generate c (integer) indices from 0 to $n-1$ and then use the corresponding vectors from `h` as the initial means. A list of lists is returned.

3. A third approach is to simply use existing vectors at n/c intervals in `h`, where `h` consists of n vectors.

Now the good news: we will generate the initial means used for SDE 2 evaluation. The bad news is you will need to generate your own $\underline{\mu}_i(0)$ for your development testing.

3 Prototypes, Signatures and Examples of Elementary Functions to be Developed

Notes:

- The use of `let` in the following examples is only for illustration. It should not appear in your `ocaml` source.

- **Carefully observe the function naming convention. Case matters. We will not rename any of the functions you submit. Reread the preceding three sentences at least 3 times.**

- You may develop additional functions to assist the required functions in Sections 3 and 4.

- **Note the argument interface on all multiple-argument functions you will design, implement and test is tupled**.
- You should work through all the samples by hand to get a better idea of the computation prior to function design, implementation and testing.

3.1 distanceR2

```
(**
Prototype:

distanceR2(v1,v2)

Inputs: vectors v1 and v2
Returned Value: Distance (squared) between 2 vectors of arbitrary (but same)
dimension
Side Effects: none
Signature: val distanceR2 : float list * float list -> float = <fun>
Notes:
A necessary capability is, given a vector, to be able to find the closest
vector in another set of vectors.
The distance between any two vectors is computed by forming the difference vector
and then taking the square root of the inner product of this difference vector
with itself. It is also the square root of the sum of the squared elements of the
difference vector. However, since we are interested in minimum distances
(which correspond to minima of distances squared), we leave out the square root computation.
*)
```

Sample Use.

```
# let v1 =[1.0;2.0;3.0];;
val v1 : float list = [1.; 2.; 3.]
# let v2=[3.0;2.0;1.0];;
val v2 : float list = [3.; 2.; 1.]
# distanceR2(v1,v2);;
- : float = 8.
# let v3=[1.0;1.0;1.0];;
val v3 : float list = [1.; 1.; 1.]
# distanceR2(v1,v3);;
- : float = 5.
```

3.2 distanceSqAllMeans

```
(**
```

```
Prototype:

distanceSqAllMeans(v,vset)

Inputs: a vector and a set of vectors (represented as lists).
Returned Value: a vector of the distances from v to each element of vset.
Side Effects: none
Signature: val distanceSqAllMeans : float list * float list list -> float list = <fun>
Notes: The objective is to take a single vector, v, and a set of vectors (to be the
current means in list-of-lists form) and compute the distance squared from v to each
of the vectors in the given set. The result is returned as a list of squared distances.
*)
```

Sample Use. See next function.

3.3 listMinPos

```
(**
Prototype:

listMinPos (alist)

Inputs: alist
Returned Value: position (0-based indexing) of the minimum in the list
Side Effects: none
Signature: val listMinPos : 'a list -> int = <fun>
Notes:
*)
```

Sample Use.

```
# h;;
- : float list list =
[[47.698002; 62.48]; [-49.005001; -41.327999]; [45.958; 29.403];
 [-60.546001; -50.702]; [45.403; 52.994999]; [-49.021; -52.053001];
 [29.788; 58.993]; [-40.433998; -36.362999]; [30.247; 46.307999]; ...]

# let testv = nth h 2;;
val testv : float list = [45.958; 29.403]

# # distanceSqAllMeans(testv,h);;
- : float list =
[1097.11553596000385; 14020.8457784640013; 0.; 17759.9132540079954;
 556.890441816001157; 15656.090539912002; 1137.03700000000026;
```

```
 11788.7439429000042; 532.614512190001; 20030.1838764119966;
 717.18468876400425; 15354.0278442339986; 701.036916022000923;
 18818.1324653960037; 623.331983706001097; 14951.582188770004;...]

# listMinPos(distanceSqAllMeans(testv,h));;
- : int = 2
```

3.4 elsum

```
(**
Prototype:

elsum(l1,l2)

Inputs: lists l1 and l2
Returned Value: vector addition of l1 and l2
Side Effects: none
Signature: val elsum : float list * float list -> float list = <fun>
Notes:
*)
```

Sample Use.

```
# elsum ([1.0; 2.0; 3.0; 4.0;],[6.0; 7.0; 8.0; 9.0;]);;
- : float list = [7.; 9.; 11.; 13.]
```

3.5 scaleList

```
(**
Prototype:

scaleList(l1,scale)

Inputs: list l1, scale factor
Returned Value: l1 with each element divided by scale
Side Effects: none
Signature: val scaleList : float list * int -> float list = <fun>
Notes: For use in forming next set of means. Must handle empty lists and division by zero.
*)
```

Sample Use.

```
# scaleList([1.0;2.0;3.0],10);;
- : float list = [0.1; 0.2; 0.3]
```

```
# scaleList([1.0;2.0;3.0],0);;
- : float list = [1.; 2.; 3.]
```

3.6 zeroes

```
(**
Prototype:

zeroes(size)

Inputs: size
Returned Value: list of zeroes of length size
Side Effects: none
Signature: val zeroes : int -> float list = <fun>
Notes: To create a list of zeroes (0.0) of length size (either c or mean vector dimension.
*)
```

Sample Use.

```
# zeroes(10);;
- : float list = [0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.; 0.]
# zeroes(2);;
- : float list = [0.; 0.]
```

3.7 zeroMeansSet

```
(**
Prototype:

zeroMeansSet(c,d)

Inputs: c,d
Returned Value: A list of c lists (means set) all zeros each with dim=d
Side Effects: none
Signature: val zeroMeansSet : int * int -> float list list = <fun>
Notes: Creates a list of c lists (means set) all zeros and dim=d
*)
```

Sample Use.

```
# zeroMeansSet(4,5);;
- : float list list =
[[0.; 0.; 0.; 0.; 0.]; [0.; 0.; 0.; 0.; 0.]; [0.; 0.; 0.; 0.; 0.];
 [0.; 0.; 0.; 0.; 0.]]
```

3.8 zeroVdiff

```
(**
Prototype:

zeroVdiff(v1,v2)

Inputs: vectors v1 and v2
Returned Value: true if v1 and v2 are the same,; false otherwise
Side Effects: none
Signature: val zeroVdiff : 'a list * 'a list -> bool = <fun>
Notes:
1. Used to tell when done (solution achieved) ->
All elements of all vectors in the list of means  are unchanging
so for all vectors new class is the same as the old class.
2. Be careful of = vs. == in ocaml.
*)
```

Sample Use.

```
# zeroVdiff([1.0;2.0;3.0],[1.0;2.0;3.0]);;
- : bool = true
# zeroVdiff([1.0;2.0;3.0],[1.0;0.0;3.0]);;
- : bool = false
```

3.9 zeroSetDiff

```
(**
Prototype:

zeroSetDiff(s1,s2)

Inputs: list-of-lists s1 and s2
Returned Value: true if s1 and s2 are equal; false otherwise
Side Effects: none
Signature: val zeroSetDiff : 'a list list * 'a list list -> bool = <fun>
Notes: To tell if two list-of-list structures are equal
*)
```

Sample Use.

```
# zeroSetDiff([[1;2;3];[1;2;4]],[[1;2;3];[1;2;3]]);;
- : bool = false
# zeroSetDiff([[1;2;3];[1;2;3]],[[1;2;3];[1;2;3]]);;
- : bool = true
```

3.10 zeroCounts

```
(**
Prototype:

zeroCounts(c)

Inputs: c
Returned Value: list of c elements, each 0
Side Effects: none
Signature: val zeroCounts : int -> int list = <fun>
Notes:
*)
```

Sample Use.

```
# zeroCounts(6);;
- : int list = [0; 0; 0; 0; 0; 0]
```

3.11 updateCounts

```
(**
Prototype:

updateCounts(p,counts)

Inputs: p, counts
Returned Value: updated counts list with element p incremented by 1
Side Effects: none
Signature: val updateCounts : int * int list -> int list = <fun>
Notes: Function to keep track of # elements in a cluster.
Records # of vectors closest to mean p as integer --
   for eventual use in computing new cluster mean.
*)
```

Sample Use.

```
# updateCounts(3,zeroCounts(6));;
- : int list = [0; 0; 0; 1; 0; 0]
# updateCounts(3,updateCounts(3,zeroCounts(6)));;
- : int list = [0; 0; 0; 2; 0; 0]
# updateCounts(1,updateCounts(3,updateCounts(3,zeroCounts(6))));;
- : int list = [0; 1; 0; 2; 0; 0]
```

4 Prototypes, Signatures and Examples of Higher-Level Functions to be Developed

These functions have the higher-level functionality involved in recomputation of the c means, and ultimately the top-level function.

4.1 updateMeansSum

```
(**
Prototype:

updateMeansSum(v,x,means)

Inputs: v,x,means (the current c cluster sums)
Returned Value: means with v added to the vector in means at position x.
Side Effects: none
Signature: val updateMeansSum : float list * int * float list list -> float list list =
  <fun>
Notes: It is not necessary to explicitly form the new cluster sets prior to forming
their new means. Instead,  simply keep a running sum of the vectors added to a cluster
and the number of vectors in the cluster.
This function adds a vector to a vector at index x in another set
of vectors and is a key part of the computation.
*)
```

Sample Use.

```
# updateMeansSum([1.0;2.0;3.0],2,zeroMeansSet(4,3));;
- : float list list =
[[0.; 0.; 0.]; [0.; 0.; 0.]; [1.; 2.; 3.]; [0.; 0.; 0.]]
# updateMeansSum([1.0;2.0;3.0],0,zeroMeansSet(4,3));;
- : float list list =
[[1.; 2.; 3.]; [0.; 0.; 0.]; [0.; 0.; 0.]; [0.; 0.; 0.]]

(* composite uses *)
# updateMeansSum([3.0;2.0;1.0],3,updateMeansSum([1.0;2.0;3.0],0,zeroMeansSet(4,3)));;
- : float list list =
[[1.; 2.; 3.]; [0.; 0.; 0.]; [0.; 0.; 0.]; [3.; 2.; 1.]]

(* show successive addition to single element *)

# updateMeansSum([1.0;2.0;3.0],2,updateMeansSum([1.0;2.0;3.0],2,zeroMeansSet(4,3)));;
- : float list list =
[[0.; 0.; 0.]; [0.; 0.; 0.]; [2.; 4.; 6.]; [0.; 0.; 0.]]
```

4.2 formNewMeans

```
(**
Prototype:

formNewMeans(newmeanssum, newcounts)

Inputs: list of newcluster sums, list of corresponding new cluster member counts
Returned Value: list of new means
Side Effects: none
Signature: formNewMeans : float list list * int list -> float list list = <fun>
Notes: Function to take newmeanssum and updated counts and form new means set.
Note: We do not want to update the means until all vectors in h have been (re-)classified.
      This is done in function reclassify.
*)
```

Sample Use.

```
# formNewMeans([[100.5];[-29.1]],[3;5]);;
- : float list list = [[33.5]; [-5.82]]
```

4.3 reclassify

```
(**
Prototype:

reclassify(h, currmeans, newmeanssum, newcounts)

Inputs: h, currmeans, newmeanssum, newcounts
Returned Value: new (updated) means from reclassification of h using current means
Side Effects: none
Signature: val reclassify :
  float list list * float list list * float list list * int list ->
  float list list = <fun>
Notes/hints:
1. The strategy is to use this function recursively to reclassify each element of H.
  (using previously developed functions updateCounts and updateMeans).
2. Once this is done, the new or updated means are recomputed from newmeanssum and newcounts
3. newmeanssum, newcounts in reclassify are initialized with zeroes.
*)
```

Sample Use. Here h is given above. To really understand the computation, I suggest you study this example.

```
# let muzero=create_muzero_3(2,h);;

val muzero : float list list =
  [[-57.244999; -43.969002]; [-68.746002; -55.521999]]


# let mu1=reclassify(h,muzero,zeroMeansSet(2,2),zeroCounts(2));;
val mu1 : float list list =
  [[7.09669729142857; 6.97822274285714084];
   [-59.7644003600000104; -62.4527998400000044]]

# let mu2=reclassify(h,mu1,zeroMeansSet(2,2),zeroCounts(2));;
val mu2 : float list list =
  [[48.5825602000000174; 49.1188998900000087]; [-51.10444003; -52.52021005]]

# let mu3=reclassify(h,mu2,zeroMeansSet(2,2),zeroCounts(2));;
val mu3 : float list list =
  [[48.5825602000000174; 49.1188998900000087]; [-51.10444003; -52.52021005]]
```

4.4 cmeans

The top-level function is named `cmeans`. This function should work for any c ($< |H|$) and any dimension of input vectors.

Part of this function design relies on determining when to stop. We adopt the philosophy: When done (solution achieved) all elements of all vectors in the list of means are unchanging.

```
Prototype:

cmeans(c,h,mucurrent)

Inputs: c,h,mucurrent
Returned Value: final c-means (as a list of means lists)
Side Effects: none
Signature: val cmeans : int * float list list * float list list -> float list list =
<fun>
Notes: mucurrent starts as muzero (implies recursion).
Stops when means not changing.
*)
```

Sample Use.

```
(* 2D test; c=2 *)
```

```
# let muzero=create_muzero_3(2,h);;
val muzero : float list list =
  [[-57.244999; -43.969002]; [-68.746002; -55.521999]]
# cmeans(2,h,muzero);;
- : float list list =
[[48.5825602000000174; 49.1188998900000087]; [-51.10444003; -52.52021005]]
#

(* 4D test; c=2 *)

# #use"test4d.caml";; (* on Canvas *)
val v : float list list =
  [[-3.682251; -3.86501; -26.724585; 9.971295];
   [0.675595; -5.314048; 12.171517; 4.948048];
   [6.538969; 0.796109; 6.885519; -0.277439];
   [6.904036; -6.858591; 3.88697; -1.947017];
...]; ...]
# let muzero4=create_muzero_3(2,v);;
val muzero4 : float list list =
  [[11.739382; 11.130896; 5.769114; 11.800967];
   [0.27744; 7.263812; 4.742097; 6.796983]]
# cmeans(2,v,muzero4);;
- : float list list =
[[5.91223877311895052; 0.987439540450160469; 6.27431081118970546;
  1.23034376655948985];
 [-1.28225301480968668; -6.19357348581312905; -1.56818392290657327;
  -6.55956124332180668]]

(* 3rd check using inputVectors2 *)

# #use"inputVectors2.caml";;  (* This file contains training set vset2 *)
val vset2 : float list list =
  [[57.415001; 55.627998]; [-33.544998; -45.639]; [40.626999; 53.868999];
   [-51.181; -56.695999]; [36.652; 53.931999]; [-38.917; -61.306];
   [55.441002; 59.639999]; [-55.047001; -73.101997]; [56.948002; 52.292];
   ; ...]; ...]

# let muzero3=create_muzero_3(2,vset2);;
val muzero3 : float list list = [[-26.32; -38.094002]; [-48.57; -54.43]]

# cmeans(2,vset2,muzero3);;
- : float list list =
[[50.8551899300000088; 50.86648994];
 [-50.6787099199999815; -51.6003100600000195]]
```

5 How We Will Grade Your Solution

The script below will be used with varying input files and parameters.

```
#use "cmeans.caml";;              (* YOUR ocaml source -- all the required functions *)
#use "inputs.caml";;              (* OUR TEST H inputs in list-of-list form *)
#use "initial.caml";;               (* OUR initial means *)
(* test individual functions, especially *)
cmeans(2,h,muzero);;              (* sample invocation *)
```

The grade is based upon a correctly working solution.

6 `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.

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 single `ocaml` source file for your function implementations. The file is to be named `cmeans.caml`. Note this file must include all the functions defined in this document. It may also contain other 'helper' or auxiliary functions you developed.

3. A log of 2 sample uses of each of the required functions and a log of 5 sample uses of the resulting main function (`cmeans`). Name this log file `<yourname>-sde2.log`.

The use of `ocaml` should not generate any errors or warnings. The grader will attempt to build your executable. Recall the grade is based upon a correctly working solution.