ECE/CPSC 352
Fall 2018
Software Design Exercise #3

Canvas submission only

Assigned 11-7-2018; Due 12-5-2018 11:59 PM

Contents

1 Preface

1.1 Objectives

The objective of SDE 3 is to implement a `Prolog` version of the c-means algorithm. We will specify a number of predicates 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 declarative programming;
- Implement a declarative (Prolog) version of an interesting algorithm;
- Deliver working software based upon specifications; and
- Learn `Prolog`.

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 3;
2. The `Prolog` 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 `swipl` website and 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 Data Structures and Representation in `Prolog`

One of the most common questions which arise at this point is:

> *'How do I get the vectors, 'into' Prolog?'*

The following should help.

2.2 From Vectors to (Prolog) Lists

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

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

becomes, in Prolog list form:

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

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

Simple examples of vector representation include:

```
vector(v1, [1.0,2.0,3.0]).
vector(v2, [3.0,2.0,1.0]).
vector(v3, [1.0,1.0,1.0]).
```

2.3 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 are 2 2-D examples of H in Prolog:

```
h(ts1, [[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]]).

h(ts2, [[47.698002, 62.480000],
[-49.005001, -41.327999]]).
```

Notice h is a predicate, with the second argument bound to the actual data. **In SDE3, like SDE2, H may consist of any number of vectors of arbitrary (but fixed) dimension. Be sure to keep this in mind as you develop predicates**. The list-of-list form of the set of vectors comprising $\underline{\mu}_i(0)$ is similar.

2.4 Initial Means

To be brief, remarks about initialization parallel those of SDE2. One 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 3 evaluation. The bad news is you will need to generate your own $\underline{\mu}_i(0)$ for your development testing.

3 Prototypes and Examples of Predicates to be Developed

You will design, implement, test and deliver the predicates described in the following sections. Note:

1. Recall the '/n' in the predicate representation indicates the arity of the predicate is n. It is not part of the name but rather the Prolog convention for showing predicate name and arity.
2. Recall the +,- or ? symbol indicates the role of the argument.
3. Pay special attention to the predicate naming and argument specifications. Sine your submission will be graded by a Prolog script, if you deviate from this your tests will probably fail. **The graders will not fix or change anything in your submission.**
4. Pay attention to the file naming conventions specified.
5. The implementation of all predicates is to be included in a single file named `cmeans.pro`.
6. You may design and use other predicates to support any of these predicates, but they are not (directly) graded. They must also be included in your single SDE1 Prolog file.

3.1 distanceR2/3

```
(**
Prototype:

distanceR2(+V1,+V2,-Dsq)

Arguments: vectors V1 and V2 (as lists), Dsq is result
Notes:
1. 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.
2. This may be done recursively, element-by-element.
*)
```

Sample Use. You should compare this with your `ocaml` results.

```
?- vector(v1, V1), vector(v2,V2), distanceR2(V1,V2,D).
V1 = [1.0, 2.0, 3.0],
V2 = [3.0, 2.0, 1.0],
D = 8.0.

?- vector(v1, V1), vector(v3,V3), distanceR2(V1,V3,D).
V1 = [1.0, 2.0, 3.0],
V3 = [1.0, 1.0, 1.0],
D = 5.0.
```

3.2 distanceSqAllMeans/3

```
(**
Prototype:

distanceSqAllMeans(+V,+Vset,-Dsq)

Arguments: a vector and a set of vectors (represented as lists), and a vector of the
distances from v to each element of vset.
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 a list of squared distances.
*)
```

Sample Use. In the examples below, H.

```
?- h(ts1,H).
H = [[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|...]].

?- h(ts1,H),nth0(0,H,First).
H = [[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|...]],
First = [47.698002, 62.48].

?- h(ts1,H),nth0(0,H,First),distanceSqAllMeans(First,H,What).
H = [[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|...]],
First = [47.698002, 62.48],
What = [0.0, 20127.571445602007, 1097.1155359600039, 24526.929309464005, 95.23227815000496,
22472.373665942006, 332.92734064000405, 17537.187875314].

?- h(ts1,H),nth0(2,H,Third).
H = [[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|...]],
Third = [45.958, 29.403].

?- h(ts1,H),nth0(2,H,Third),distanceSqAllMeans(Third,H,What).
H = [[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|...]],
Third = [45.958, 29.403],
What = [1097.1155359600039, 14020.845778464001, 0.0, 17759.913254007995, 556.8904418160012,
15656.090539912002, 1137.0370000000003, 11788.743942900004].

?- h(ts1,H),last(H,Last),distanceSqAllMeans(Last,H,What).
H = [[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|...]],
Last = [-40.433998, -36.362999],
What = [17537.187875314, 98.11331742600892, 11788.743942900004, 610.0996143500097,
15352.842032220007, 319.912766108008, 14023.895548400005, 0.0].
```

3.3 listMinPos/2

```
Prototype: listMinPos(+Alist,-M)

Arguments: Alist, M is position (0-based indexing) of the minimum in the list
```

Sample Use.

```
?- h(ts1,H),nth0(0,H,First),distanceSqAllMeans(First,H,Dists),listMin(Dists,What).
H = [[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|...]],
First = [47.698002, 62.48],
Dists = [0.0, 20127.571445602007, 1097.1155359600039, 24526.929309464005, 95.23227815000496,
22472.373665942006, 332.92734064000405, 17537.187875314],
What = 0.0.

?- h(ts1,H),nth0(2,H,Third),distanceSqAllMeans(Third,H,Dists),listMin(Dists,What).
H = [[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|...]],
Third = [45.958, 29.403],
Dists = [1097.1155359600039, 14020.845778464001, 0.0, 17759.913254007995, 556.8904418160012,
15656.090539912002, 1137.0370000000003, 11788.743942900004],
What = 0.0.

?- h(ts1,H),nth0(0,H,First),distanceSqAllMeans(First,H,Dists),listMinPos(Dists,Where).
H = [[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|...]],
First = [47.698002, 62.48],
Dists = [0.0, 20127.571445602007, 1097.1155359600039, 24526.929309464005, 95.23227815000496,
22472.373665942006, 332.92734064000405, 17537.187875314],
Where = 0 .

?- h(ts1,H),nth0(2,H,Third),distanceSqAllMeans(Third,H,Dists),listMinPos(Dists,Where).
H = [[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|...]],
Third = [45.958, 29.403],
Dists = [1097.1155359600039, 14020.845778464001, 0.0, 17759.913254007995, 556.8904418160012,
15656.090539912002, 1137.0370000000003, 11788.743942900004],
Where = 2
```

3.4 elsum/3

```
Prototype: elsum(+L1,+L2,-S)
Arguments: L1,L2 and S are lists of same length
Notes:
1. Implement vector addition as list 'addition' of
   element by element sums of lists L1 and L2
2. Add corresponding elements recursively
3. DO NOT NEED LIST LENGTH.
```

Sample Use.

```
?- elsum([1.0, 2.0, 3.0, 4.0],[6.0, 7.0, 8.0, 9.0],Answer).
Answer = [7.0, 9.0, 11.0, 13.0].
```

3.5 scaleList/3

```
Prototype: scaleList(+List,+Scale,-Answer)
Arguments: List, Scale factor, Answer is List with each element divided by scale factor.
Notes:
1. Simple utility for use in forming next set of means.
2. Must handle empty lists and division by zero (see examples).
```

Sample Use.

```
?- scaleList([1.0,2.0,3.0],10,Result).
Result = [0.1, 0.2, 0.3].

?- scaleList([1.0,2.0,3.0],0,Answer).
Answer = [1.0, 2.0, 3.0].
```

3.6 zeroes/2

```
(**
Prototype: zeroes(+Size,-TheList)

Notes:
creates a list of zeroes (0.0) of length Size (see example)
```

Sample Use.

```
?- zeroes(6,Alist).
Alist = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0].
```

3.7 zeroMeansSet/3

```
Prototype: /* zeroMeansSet(+Cmeans,+Dim,-Set)

Note creates a list (Set) of Cmeans lists, each all zeros with length Dim
```

Sample Use.

```
?- zeroMeansSet(4,5,TheSet).
TheSet = [[0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0],
          [0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0]].
```

The next 2 predicates are used for stopping, i.e., how to tell when the algorithm is done (solution achieved). Stopping conditions might be articulated as:

1. All elements of all vectors in the list of means are unchanging; or
2. For all vectors in H, the new class is the same as the old class.

We will use the first method.

3.8 zeroVdiff/2

This predicate tests vector equivalence.

```
Prototype: zeroVdiff(+V1,+V2)

Succeeds if V1 and V2 are identical.
```

Sample Use.

```
?- zeroVdiff([1,2,3,4],[4,3,2,1]).
false.

?- zeroVdiff([1,2,3,4],[1,2,3,4]).
true.
```

3.9 zeroSetDiff/2

Here we test if two sets of means (2 list of lists) are equal. This is a termination condition.

```
Prototype: zeroSetDiff(+S1,+S2)

Arguments: 2 list-of-lists S1 and S2
Succeeds (true) if S1 and S2 are equal; false otherwise
```

Sample Use.

```
?- zeroSetDiff([[1,2,3],[1,2,4]],[[1,2,3],[1,2,3]]).
false.

?- zeroSetDiff([[1,2,3],[1,2,3]],[[1,2,3],[1,2,3]]).
true.
```

Now we develop predicates to keep track of the means by summing vectors and keeping track of their number.

3.10 zeroCounts/2

```
Prototype: zeroCounts(+C,-CountsList)
   like predicate zeroes, but creates integer values
```

Sample Use.

```
?- zeroCounts(8,Out).
Out = [0, 0, 0, 0, 0, 0, 0, 0].
```

3.11 updateCounts/3

```
(**
Prototype: updateCounts(+P,+Counts,-Updated)

Arguments: Updated Counts list with element P incremented by 1

Notes:
Predicate to keep track of # of elements in a cluster.
Records # of vectors closest to mean P as an integer.
For eventual use in computing new cluster mean.
Easy to see from examples.
Reminder: 0-based indexing
```

Sample Use.

```
?- zeroCounts(6,S),updateCounts(0,S,Updated).
S = [0, 0, 0, 0, 0, 0],
Updated = [1, 0, 0, 0, 0, 0].

?- zeroCounts(6,S),updateCounts(3,S,Updated).
S = [0, 0, 0, 0, 0, 0],
```

```
Updated = [0, 0, 0, 1, 0, 0].

?- zeroCounts(6,S),updateCounts(3,S,Updated),updateCounts(3,Updated,Overall).
S = [0, 0, 0, 0, 0, 0],
Updated = [0, 0, 0, 1, 0, 0],
Overall = [0, 0, 0, 2, 0, 0].

?- zeroCounts(6,S),updateCounts(3,S,Updated),updateCounts(3,Updated,Overall),
updateCounts(0,Overall,Overall2).
S = [0, 0, 0, 0, 0, 0],
Updated = [0, 0, 0, 1, 0, 0],
Overall = [0, 0, 0, 2, 0, 0],
Overall2 = [1, 0, 0, 2, 0, 0].
```

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

Here, the real work starts. These predicates have the higher-level functionality involved in recomputation of the c means, and ultimately the top-level predicate.

4.1 updateMeansSum/4

```
(**
Prototype: updateMeansSum(+V,+X,+Means,-NewMeansSum)

Add a vector, V, to a vector at index X in a set of vectors (Means)
with the result in NewMeansSum.

Notes: It is not necessary to explicitly form the new cluster sets prior to forming
the new means. In fact, this is inefficient.
Instead, we simply keep a running sum of the vectors summed in a cluster
and the number of vectors in the cluster. This predicate is a key part of cmeans computation.
```

Sample Use.

```
?- zeroMeansSet(4,3,M), updateMeansSum([1.0,2.0,3.0],0,M,Result).
M = [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]],
Result = [[1.0, 2.0, 3.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]].

?- zeroMeansSet(4,3,M), updateMeansSum([1.0,2.0,3.0],2,M,Result).
M = [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]],
```

```
Result = [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [1.0, 2.0, 3.0], [0.0, 0.0, 0.0]].
```

4.2 formNewMeans/3

```
(**
Prototype: formNewMeans(+Newmeanssum, +Newcounts,-NewMeans)

Recomputation of the means; this predicate uses Newmeanssum and Newcounts to form NewMeans.
Really just normalization of Newmeanssum for each class by dividing each vector
by the number of vectors in the cluster.
Now you should see the utility of the counting and summing as we classify.
```

Sample Use.

```
?- formNewMeans([[100.5],[-29.1]],[3,5],NewMeans).
NewMeans = [[33.5], [-5.82]].
```

4.3 reclassify/3

This predicate implements the heart of the algorithm.

```
Prototype: reclassify(+H, +Currmeans, -UpdatedMeans)

Arguments:

  1. H (used recursively, processing a single vector starting at head)
  2. Currmeans is current set of c means (required, of course, to allow reclassification of H)
  3. UpdatedMeans is 'new' means set.

Notes:
 0. The strategy is for this predicate to recursively reclassify each element of H.
    (using previously developed functions updateCounts and updateMeans).
 1. Other previously developed predicates (including formNewMeans) are used.
    An auxiliary predicate recommended (see note 3).
 2. We can determine c and vector dimension from Currmeans for count initialization.
 3. Notice also reclassify does not explicitly have arguments for Newmeanssum, Newcounts which a
    necessary to reclassify. These are initialized with zeroes.
```

Sample Use. To really understand the computation, I suggest you study these examples.

```
?- h(ts2,What).
What = [[47.698002, 62.48], [-49.005001, -41.327999]].
```

```
?- h(ts2,H), reclassify(H,[[1.0,1.0],[-1.0,-2.0]],NewMu).
H = NewMu, NewMu = [[47.698002, 62.48], [-49.005001, -41.327999]].

?- h(ts2,H), reclassify(H,[[1.0,1.0],[-1.0,-2.0]],NewMu),reclassify(H,NewMu,Mu2).
H = NewMu, NewMu = Mu2, Mu2 = [[47.698002, 62.48], [-49.005001, -41.327999]].

?- h(ts2,H), reclassify(H,[[1.0,1.0],[-1.0,-2.0]],NewMu),reclassify(H,NewMu,Mu2),
   reclassify(H,Mu2,Mu3).
H = NewMu, NewMu = Mu2, Mu2 = Mu3, Mu3 = [[47.698002, 62.48], [-49.005001, -41.327999]].
```

When you're done with the simple example, consider a full-fledged case (bigH data given on Canvas):

```
muzero([[-57.244999, -43.969002], [-68.746002, -55.521999]]). /* used in previous testing */

?- bigH(H), muzero(MZ0),reclassify(H,MZ0,Mu1).

?- bigH(H), muzero(MZ0),reclassify(H,MZ0,Mu1).
H = [[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|...], [...|...]|...],
MZ0 = [[-57.244999, -43.969002], [-68.746002, -55.521999]],
Mu1 = [[7.09669729142857, 6.978222742857141], [-59.76440036000001, -62.452799840000004]].

?- bigH(H), muzero(MZ0),reclassify(H,MZ0,Mu1), reclassify(H,Mu1,Mu2).
H = [[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|...], [...|...]|...],
MZ0 = [[-57.244999, -43.969002], [-68.746002, -55.521999]],
Mu1 = [[7.09669729142857, 6.978222742857141], [-59.76440036000001, -62.452799840000004]],
Mu2 = [[48.58256020000002, 49.11889989000001], [-51.10444003, -52.52021005]].

?- bigH(H), muzero(MZ0),reclassify(H,MZ0,Mu1), reclassify(H,Mu1,Mu2),reclassify(H,Mu2,Mu3).
H = [[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|...], [...|...]|...],
MZ0 = [[-57.244999, -43.969002], [-68.746002, -55.521999]],
Mu1 = [[7.09669729142857, 6.978222742857141], [-59.76440036000001, -62.452799840000004]],
Mu2 = Mu3, Mu3 = [[48.58256020000002, 49.11889989000001], [-51.10444003, -52.52021005]].

?- bigH(H), muzero(MZ0),reclassify(H,MZ0,Mu1), reclassify(H,Mu1,Mu2),reclassify(H,Mu2,Mu3),
reclassify(H,Mu3,Mu4).
H = [[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|...], [...|...]|...],
MZ0 = [[-57.244999, -43.969002], [-68.746002, -55.521999]],
Mu1 = [[7.09669729142857, 6.978222742857141], [-59.76440036000001, -62.452799840000004]],
Mu2 = Mu3, Mu3 = Mu4, Mu4 = [[48.58256020000002, 49.11889989000001], [-51.10444003, -52.52021005
```

```
THIS RESULT CHECKS WITH OCAML.
```

4.4 cmeans/3

The final, top-level predicate, `cmeans`. This predicate should work for any c ($< |H|$) and any dimension of input vectors. Part of this function design relies on determining when to stop. Recall we adopted the philosophy: *When done (solution achieved) all elements of all vectors in the list of means are unchanging.*

```
Prototype: cmeans(+H,+MuCurrent,-MuFinal)

MuCurrent starts as muzero; c is derivable from muzero
Stops when means not changing.
```

Sample Use.

```
?- h(ts1,H), cmeans(H,[[1.0,1.0],[-1.0,-2.0]],MuFinal).
H = [[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|...]],
MuFinal = [[42.2117505, 50.967749749999996], [-49.7515, -45.11149975]] .

?- h(ts2,H), cmeans(H,[[1.0,1.0],[-1.0,-2.0]],MuFinal).
H = MuFinal, MuFinal = [[47.698002, 62.48], [-49.005001, -41.327999]].

?- bigH(H), muzero(MZ0),cmeans(H,MZ0,MuFinal).
H = [[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|...], [...|...]|...],
MZ0 = [[-57.244999, -43.969002], [-68.746002, -55.521999]],
MuFinal = [[48.58256020000002, 49.11889989000001], [-51.10444003, -52.52021005]].
```

5 How We Will Grade Your Solution

A (Prolog) script will be used with varying input files and parameters. The grade is based upon a correctly working solution.

6 Prolog Use Limitations and Constructs Not Allowed

1. The entire solution must be in SWI-Prolog (Version 7.2 or newer.)

2. No use of the `if..then` construct anywhere in your Prolog source file. (Don't even bother looking for it). It has the syntax:

   ```
   condition -> then_clause ; else_clause
   ```

   and is useful for people who want an if-then capability, but don't understand the goal-satisfaction mechanism in Prolog.

3. No use of predicates `assert` or `retract`.

   If you are in doubt about any allowable predicates, ask and I'll provide a 'private-letter ruling'.

The objective is to obtain proficiency in declarative programming and Prolog, not to try to find built-in predicates which simplify or trivialize the effort, or to implement an imperative solution in a declarative programming paradigm.

7 Pragmatics

Use this document as a checklist to be sure you have responded to all parts of the SDE, and have included the required files (*.pro and *.log). Furthermore:

- The simulation uses **only** (SWI-Prolog) code you wrote.
- The final, single zipped archive which must be submitted (to Canvas) by the deadline must be named **<yourname>-sde3.zip**, where <yourname> is your (CU) assigned user name.

  The contents of this archive are:

  1. A `readme.txt` file listing the contents of the archive and a very 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
     SIGN _____

  2. Your single SWI-Prolog source file `cmeans.pro` containing the required predicates.

3. A log file named `sde3.log` showing 3 uses of each required predicate. Uses test cases other than those shown herein.

We will attempt to consult your Prolog source file and then query Prolog with relevant goals. Most of the grade will be based upon this evaluation.

8 Final Remark: Deadlines Matter

Since multiple submissions to Canvas are allowed[1], if you have not completed all predicates, 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 to either me or the graders.**

[1] But we will only download and grade the latest (or most recent).