L. Vandenberghe ECE133A Fall 2018

Homework 1

Due: Tuesday 10/9/2018.

- Homework is due at 5:00PM on the due date. It can be submitted in class, or using the submission box in the TA meeting room (67-112 Engineering 4). Late homework will not be accepted.
- You are allowed to discuss the homework with each other, but you must write up your own answers and, for programming assignments, code your own programs to hand in.
- Data files can be found at `www.seas.ucla.edu/~vandenbe/133A/data-files`.
- For exercises that involve programming, include the code with your results.

Reading assignment: Chapters 1–4 of the textbook *Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares*.

Homework problems. Exercises with prefix 'T' refer to exercises in the textbook. Exercises with prefix 'A' refer to the set of additional exercises on the class webpage `www.seas.ucla.edu/~vandenbe/133A/133A-exercises.pdf`.

1. Exercise T2.4.
2. Exercise T2.8.
3. Exercise T3.9.
4. Exercise T3.25.
5. Exercise A1.2.
6. *The k-means algorithm.* We apply the k-means algorithm to the example in §4.4.2 of the textbook. In this example we cluster the word histogram vectors of 500 Wikipedia articles, using a dictionary of 4423 words.

   We first explain how to obtain the data.

   - MATLAB users should download the binary file `wikipedia_m.mat` from the dataset directory on the course webpage, and import it in MATLAB using the command `load wikipedia_m`.

- Octave users should download the file `wikipedia_o.mat`, and import it in Octave using the command `load wikipedia_o`.
- Julia users should download the two files `wikipedia.jl` and `tdmatrix.txt`, and import the data as follows:

```
julia> include("wikipedia.jl");
julia> using DelimitedFiles
julia> tdmatrix = readdlm("tdmatrix.txt");
```

In each case, three variables will be defined: `tdmatrix` (which stands for term-document matrix), `articles`, `dictionary`. The variable `tdmatrix` is a 4423×500 matrix with the 500 word histogram vectors as its columns. The variable `articles` is an array of length 500 with the article titles: `articles(j)` (in MATLAB), `articles(j,:)` (in Octave), or `articles[j]` (in Julia) is the title of the article represented by column j in `tdmatrix`. The variable `dictionary` is an array of length 4423 with the words in the dictionary: `dictionary(i)` (in MATLAB), `dictionary(i,:)` (in Octave), or `dictionary[i]` (in Julia) is the word referred to by row i of `tdmatrix`.

You are asked to apply the k-means algorithm to this set of $N = 500$ vectors, with $k = 8$ groups and starting from a random initial group assignment (as opposed to starting from randomly generated group representatives, as in Algorithm 4.1 in the textbook). To evaluate the quality of the clustering we use the clustering objective

$$J = \frac{1}{N} \sum_{i=1}^{N} \min_{j=1,\ldots,k} \|x_i - z_j\|^2.$$

The algorithm is terminated when J is nearly equal in two successive iterations (e.g., we terminate when $|J - J_{\text{prev}}| \leq 10^{-8} J$, where J_{prev} is the value of J after the previous iteration).

Test your code with different random initial assignments, and summarize the results for the best clustering you find (with the lowest clustering objective). To summarize the result, give the five top terms for each cluster (the words associated with the highest five components of the cluster representative) and the titles of the five articles in the cluster closest to the representative.

MATLAB/Octave suggestions

- You can create an initial group assignment using the `randi` function: the command

```
group = randi(8, 1, 500);
```

  creates a pseudorandom 1×500 array `group` of integers between 1 and 8.

- Since we start from a random partition, the order of the two steps in Algorithm 4.1 is switched. The first step in each cycle is to compute the group representatives for the current assignment. Suppose we store the assignment in an array `group` of length 500, as initialized above. We can find the columns of `tdmatrix` (*i.e.*, the articles) that are assigned to group i using the function `find`. The command

  ```
  I = find(group == i);
  ```

  defines an index vector I with the indexes of the articles in group i, so that `tdmatrix(:,I)` is the submatrix of `tdmatrix` with the columns assigned to group i. To find the average of the columns in this matrix, you can use for-loops, or the functions `sum` or `mean`. (Be sure to look up what `sum` or `mean` do when applied to a matrix; see `help sum` and `help mean`.)
- The function `sort` with two output arguments is useful to find the top terms for each group representative, and the articles in each cluster closest to the representative.

Julia suggestions

- The code on page 34 in the *Julia Language Companion* on the textbook website can be followed as a template, and requires very few changes for this exercise. The main difference is that the input argument `x` in the `kmeans` function in the companion is an array of vectors, not a matrix. Also, if you use the for-loop on page 34 outside a function, you will need to add `global Jprevious` inside the loop.
- The Julia function `sortperm` is useful to find the top terms for each group representative, and the articles in each cluster closest to the representative.