CS273a Homework #5

Introduction to Machine Learning: Winter 2015

Due: Tuesday March 10th, 2015

Write neatly (or type) and show all your work!

Please remember to turn in at most two documents, one with any handwritten solutions, and one PDF file with any electronic solutions.

Problem 1: Basics of Clustering

The code this week provides the three clustering algorithms we discussed: k-means, agglomerative clustering, and EM for Gaussian mixture models. (These functions are also provided in various Matlab toolboxes; you are free to use those if you prefer.) In this problem, you'll do some basic exploration of the clustering techniques.

(a) Load the usual Iris data restricted to the first two features, and ignore the class / target variable. Plot the data and see for yourself how "clustered" you think it looks. (You don't have to report on this.)

(b) Run k-means on the data, for $k = 5$ and $k = 20$. For each, turn in a plot with the data, colored by assignment, and the cluster centers. (You can do this yourself manually, using `plotClassify2D([],X,z)`, or just use the plotting options in the k-means code.) Try a few different initializations and check to see whether they find the same solution; if not, pick the one with the best score.

(c) Run agglomerative clustering on the data, using *single linkage* and then again using *complete linkage*, each with 5 and then 20 clusters. Again, plot with color the final assignment of the clusters, and describe their similarities and differences from each other and k-means. (This algorithm has no initialization issues; so you do not have to try multiple initializations.)

(d) Run the EM Gaussian mixture model with 5 and 20 components. Using the internal plotting functions will show you the evolution of the mixture components' locations and shapes (the magnitude coefficient isn't shown). As with k-means, you may want to try several initializations. Again, compare / discuss differences with the other clusterings. Which do you think is most reasonable?

As a side note: Clustering is often a useful element of other predictive tasks, like supervised learning. To be used properly, you need to be able to define the "out of sample" cluster assignments, but this is very easy for k-means and EM (a bit less so for agglomerative); for k-means, say:
`crule = knnClassify( clusters, (1:K)', 1 ); z = predict( crule, X );`
Then, you can then use these cluster assignments as a feature in a classifier:
`Phi = @(x) to1ofK( predict(crule, x) , 1:K );` will create K new binary features indicating which of the clusters is closest to a new point x.

Problem 2: k-means Clustering on Text

The zip file provided with your homework contains a collection of text documents (a "corpus") from the New York Times on January 1, 2000, in the "text/example1" subdirectory, along with several

scripts (courtesy of Dr. David Newman) to process them into a form more amenable to machine learning algorithms. In particular, they construct the "bag of words" representation we discussed in class, in which each document is a data point, each word in the dictionary (`vocab.txt`) is represented by a feature, and that feature's value is the number of times it appeared in the document. **You do not need to re-run the pre-processing**, but for completeness here is what I did:

```
cd text
d = dir('example1');           % for this directory of text files
fh = fopen('docs.txt','w');    % make a file listing all the file names
for i=3:length(d), fprintf(fh,'example1/%s\n',d(i).name); end;
fclose(fh);
                               % now, run Dr. Newman's perl scripts:
!perl Makewordstream.pl < docs.txt       > wordstream.txt
!perl Makevocab.pl      < wordstream.txt > vocab.txt
!perl Makedocword.pl    < wordstream.txt > docword.txt
% This extracts all the words, creates the vocabulary, and puts the data in our form
```

To load the data into Matlab, from the `text` directory use

```
% Read in vocabulary and data (word counts per document)
[vocab] = textread('vocab.txt','%s');
[did,wid,cnt] = textread('docword.txt','%d%d%d','headerlines',3);

X = sparse(did,wid,cnt);  % convert to a matlab sparse matrix
D = max(did);             % number of docs
W = max(wid);             % size of vocab
N = sum(cnt);             % total number of words

% It is often helpful to normalize by the document length:
Xn= X. / repmat(sum(X,2),[1,W]) ; % divide word counts by doc length
```

Since most words do not appear in any given document, we use Matlab's `sparse` matrix representation, which reduces the storage needed. However, some functions do not like sparse matrices; if you run into trouble you can convert it to a standard matrix with a lot of zeros with `full(Xn)`; our data are small enough that this should be possible, although not for a large text collection.

To interpret: row i of Xn corresponds to document i (here, text file `20000101.`i`.txt`). Column j corresponds to word `vocab{j}`, and indicates the fraction of the article represented by that word. It is often helpful to be able to output the most common words in an article based on its features:

```
[sorted,order] = sort( Xn(i,:), 2, 'descend');
fprintf('Doc %d: ',i); fprintf('%s ',vocab{order(1:10)}); fprintf('\n');
```

or to be able to view a snippet of the article itself:

```
fname = sprintf('example1/20000101.%04d.txt',i);
txt = textread(fname,'%s',10,'whitespace','\r\n'); fprintf('%s\n',txt{:});
```

Note that, due to lack of pre-processing, a few articles appear multiple times in the corpus, so you may see some repeated data points. Just try to ignore this effect, if possible.

In this problem you will use k-means clustering to try to understand and summarize the collection of documents.

(a) Use either the `kmeans` function from the stats toolbox or my provided version to perform k-means clustering on the data. Use about 20 clusters, and also compute the kmeans cost function (sum of squared distances) associated with your final clustering.

(b) Since k-means is sensitive to initialization, re-run your k-means clustering at least four more times. Report the value of the cost function for each run, and pick the best one to use for the rest of the problem.

(c) How many articles (documents) are associated with each cluster in your final clustering? Each cluster is described by a vector in the feature space (word frequencies) – for each cluster, print out the cluster center's ten "most likely" words. Do they form interpretable sets?

(d) Look at the cluster assignments for the documents, and find the cluster(s) associated with documents 1, 15, and 30. For each of these clusters, find all the documents in that cluster (or at least 12 of them, if there are a lot) and print out the first several lines of each document. Do the documents form coherent groups? Based on the words you saw in the previous part, is this the best cluster for those documents?

(e) Repeat the clustering process, this time using twice as many clusters. Compare the resulting cost of the clustering and discuss briefly. Again, find the clusters associated with documents 1, 15, and 30, and look at their other assigned documents – are the clusters significantly different? If so, how, and do those clusters seem better or worse?

Problem 3: EigenFaces

In class I mentioned that PCA has been applied to faces, and showed some of the results. Here, you'll explore this representation yourself. First, load the data and display a few faces to make sure you understand the data format:

```
X = load('data/faces.txt');          % load face dataset
img = reshape(X(i,:),[24 24]);  % convert vectorized datum to 24x24 image patch
imagesc(img); axis square; colormap gray;    % display an image patch; you may have to squint
```

(a) Subtract the mean of the face images ($X_0 = X - \mu$) to make your data zero-mean.

(b) Take the SVD of the data, so that

$$X_0 \approx U \cdot S \cdot V^T$$

Note that since the number of data is larger than the number of dimensions, S will be zero on its lower part; I suggest taking $X_0 \approx W \cdot V$ where $W = U \cdot S$. You may also prefer to use `svds`, which can return only the top K singular values and their associated columns of U and V.

(c) For $K = 1 \ldots 10$, compute the approximation to X_0 given by $\hat{X}_0 = W(:, 1:K) \cdot V(:, 1:K)^T$, and compute the mean squared error in the SVD's approximation, `mean(mean( (`$X_0 - \hat{X}_0$`).^2 ))`. Plot these values as a function of K.

(d) Display the first few principal directions of the data, by computing $\mu + \alpha$ V(:,j)' and $\mu - \alpha$ V(:,j)', where α is a scale factor (I suggest, for example, `2*median(abs(W(:,j)))`, to get a sense of the scale found in the data). These should be vectors of length 24^2, so you can reshape them and view them as "face images" similar to the original data.

(e) These are often called "latent space" methods, as the coefficients can be interpreted as a new geometric space in which the data are being described. To visualize this, choose a few faces at random (say, about 15–25), and display them as images with the coordinates given by their coefficients on the first two principal components:

```
idx = ...      % pick some data at random or otherwise
figure; hold on; axis ij; colormap(gray);
range = max(W(idx,1:2)) - min(W(idx,1:2)); % find range of coordinates to be plotted
scale = [200 200]./range;                  % want 24x24 to be visible but not large on new scale
for i=idx, imagesc(W(i,1)*scale(1),W(i,2)*scale(2), reshape(X(i,:),24,24)); end;
```

This can often help you get a "feel" for what the latent representation is capturing.

(f) Choose two faces and reconstruct them using only K principal directions, for $K = 5, 10, 50$.

Problem 4: Go work on your project