CS273a Homework #4

Introduction to Machine Learning: Fall 2012

Due: Tuesday December 4th, 2012

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

Problem 1: Naïve Bayes

Consider the email and spam prediction problem from the previous homework. Again, we obtain the following data set of binary-valued features about each email, including whether I know the author or not, whether the email is long or short, and whether it has any of several key words, along with my final decision about whether to read it ($y = +1$ for "read", $y = -1$ for "discard").

x_1 know author?	x_2 is long?	x_3 has 'research'	x_4 has 'grade'	x_5 has 'lottery'	y $\Rightarrow$ read?
0	0	1	1	0	-1
1	1	0	1	0	-1
0	1	1	1	1	-1
1	1	1	1	0	-1
0	1	0	0	0	-1
1	0	1	1	1	1
0	0	1	0	0	1
1	0	0	0	0	1
1	0	1	1	0	1
1	1	1	1	1	-1

In the case of any ties, we will prefer to predict class +1.

(a) Compute all the probabilities necessary for a naïve Bayes classifier, i.e., $p(y)$ and all the $p(x_i|y)$.

(b) Which class would be predicted for $\underline{x} = (0\ 0\ 0\ 0\ 0)$? What about for $\underline{x} = (1\ 1\ 0\ 1\ 0)$?

(c) Compute the posterior probability that $y = +1$ given the observation $\underline{x} = (1\ 1\ 0\ 1\ 0)$.

(d) Why should we probably not use a Bayes classifier (as opposed to a naïve Bayes classifier) for these data?

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 the `kmeans` function from the stats toolbox (or modify the example kmeans code from the online notes) 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: Agglomerative Clustering

Here we will cluster the same text data as in Problem 3, but using agglomerative clustering. Use the built-in `linkage` function and associated functions (from Matlab's stats toolbox) for this problem.

(a) Use `linkage` to build an agglomerative clustering of Xn, using 'single' linkage and 'euclidean' distance. Visualize this clustering using `dendrogram`. (You will probably want to set parameter $p = 0$ to show all the leaf nodes of the dendrogram.)

(b) Repeat using 'complete' linkage instead – do you notice a significant difference? Why might this be the case? Just based on the dendrogram, which of these clusterings might you prefer and why?

(c) Use the `cluster` function with `maxclust=20` (or whatever you used in Problem 3) to produce a clustering comparable to the one you found in Problem 2. (Use either 'single' or 'complete', and tell me which.) How many documents are assigned to each cluster?

(d) Again, find the documents clustered with documents 1, 15, and 30. (If you need to, increase the number of clusters until you get at least 2-3 and at most around 15 documents in each of these clusters.) Look at the other documents assigned to these clusters – do they seem appropriate? Do they seem better or worse than k-means? (There is no absolutely right answer to these last questions – just comment based on your results.)

Problem 4: Go work on your project