

050.680 — Learning Theory
Problem Set 4
Due 26 March 2007

In this problem set you will apply the clustering techniques we discussed in class to a pair of data sets. In some places, you are asked to do certain very specific things, while in others, the assignment is quite open ended. Don't feel as though you need to pursue every part of every question exhaustively, but do enough to get a sense of the lay of the land.

Problem 1

The Peterson/Barney vowel data consist of measurements of F0, F1, F2 and F3 values for two repetitions of 10 different vowels by 76 speakers of British English (33 adult males, 28 adult females, and 15 children). After downloading the file `voweldata.m` from the website and executing the command `voweldata` in Matlab, you will have a variable `vowels` that is a 1520×7 matrix, each of whose rows represents a vowel as follows:

<i>column</i>	<i>information</i>	<i>key</i>
1	Speaker type	1 = Male, 2 = Female, 3 = Child
2	Speaker number	a unique ID for each speaker (ranges from 1-76)
3	Vowel identity	1 = i, 2 = ɪ, 3 = e, 4 = æ, 5 = ʌ, 6 = ɑ, 7 = ɔ, 8 = ʊ, 9 = u, 10 = ɜ
4	F0	fundamental frequency
5	F1	first formant
6	F2	second formant
7	F3	third formant

Visualizing the data: Matlab will happily display scatterplots of your data, with each vowel measurement colored distinctively based on whatever property you would like. To do this, you will need to use the command `scatter`. You will want to use this command with 4 arguments (though other options are possible – try executing `doc scatter` to find out more). The first two arguments will be vectors that include the x and y coordinates (respectively) of the points that you want to plot. The third argument is scalar that represents the size of each point (I find 3 to work well). The fourth argument is a vector that gives the target value for each point (and therefore should be of the same length as the first two arguments). Make a scatter plot for the vowels, where the two axes are the F0 and F1 values for the vowels, and label the points by their vowel identity. Do the distinct vowels form identifiable clusters? What happens if we label these points by speaker type? Why? Now produce scatter plots where the two values are F1 and F2, with labeling determined by vowel identity. Do these produce identifiable clusters? What about for speaker type?

k-means clustering: The first clustering technique you will try out is k-means clustering. Matlab provides a function `kmeans` which takes two obligatory arguments: the data you want to cluster in the form of a matrix (each row of values is taken to be a data point), and the number of clusters you want. This function returns a vector of length equal to the number of data points, and each value is the cluster to which that point is assigned. Perform a k-means clustering over the F1 and F2 values of each vowel with 1 target clusters, and make a scatter plot of the results.

Quantitative evaluation: One way to evaluate the goodness of the your clustering is by comparing its same vs. different judgments to those of the target. That is, for any pair of vowels v_1 and v_2 , we can

check whether or not the clustering puts them in the same cluster, and whether or not this is correct (by looking at the true vowel identities). This will give us four kinds of cases:

- *true positives*: v_1 and v_2 are in the same cluster and are the same vowel
- *false positives*: v_1 and v_2 are in the same cluster but are not the same vowel
- *true negatives*: v_1 and v_2 are in different clusters and are not the same vowel
- *false negatives*: v_1 and v_2 are in different clusters but are the same vowel

By calculating how many pairs of vowels fall into each of these cases, we can compute a trio of measures that are commonly used to evaluate performance on classification tasks:

- $\text{precision} = \frac{\# \text{ true positives}}{\# \text{ true positives} + \# \text{ false positives}}$
- $\text{recall} = \frac{\# \text{ true positives}}{\# \text{ true positives} + \# \text{ false negatives}}$
- $\text{f-score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$

Write a function `cluster_accuracy` that takes the result of clustering along with a true labeling of the data and returns precision, recall, and f-scores for same-difference judgments. (Hint: to do this, you will need to loop over all *pairs* of data points. The easiest way to do that is to nest one loop inside of another.) Use your function to evaluate the accuracy of your k-means clustering. What happens to the different measures of performance if we redo our k-means clustering with a larger number of clusters? Can you find an optimal number of clusters for each measure?

Hierarchical clustering: To perform hierarchical clustering over our vowel data, we proceed in three steps.

1. First, we use the Matlab command `pdist`, which takes two arguments: the data to be clustered and the distance measure (you may choose from `'euclidean'`, `'cosine'`, `'cityblock'`, among others – note that the distance measure must be surrounded by single quotes). Given n data points, `pdist` yields an $n \times n$ matrix where the value at cell (k, j) is the computed distance between data points k and j .
2. Next, we invoke the Matlab command `linkage`, which takes the result of `pdist` together with a metric for cluster linkage (chosen from `'single'`, `'complete'`, `'average'`, `'ward'`, or others). This command returns a hierarchical cluster tree. (This tree may be plotted using the command `dendrogram`.)
3. Finally, we select a fixed set of clusters from the tree using the command `cluster`, which takes as arguments the cluster tree produced by `linkage`, along with a specification of the number of clusters: `cluster(T, 'maxclust', n)`, where T is the cluster tree, and n is the desired number of clusters. The result of this command is a vector of the same sort produced by `kmeans`.

Perform hierarchical clustering over the F1 and F2 values for the vowels with 10 target clusters, experimenting with different distance measures and metrics for cluster linkages. Using both scatterplots and your `cluster_accuracy` function, compare your results to those you obtained from k-means clustering.

Problem 2

Now, you will apply the clustering techniques you have just learned to the task of induction of grammatical categories, as discussed in the Mintz, Newport and Bever paper. To do this, you will need to select a piece of text over which to perform clustering. It can be any kind of text you would like (not necessarily English, though it should be a language you know), and it should be at least 15,000 words in length. If you're having trouble finding something, you might check out the Project Gutenberg website (www.gutenberg.org), or if you're interested in child-directed speech, have a look at Childes (childes.psy.cmu.edu). Whatever text you choose, make sure to remove any extraneous formatting and markup. Once you have chosen your text, you can use the program `bigram` (available on the course webpage) to compute bigram statistics. `bigram('filename')` returns a vector `[text, list, marg_count, bigram_count]`, where `text` is a list containing the text of the entire file, `list` is a list of all of the k distinct words in the file, `marg_count` is a vector containing the number of occurrences of each word from `list`, and `bigram_count` is a $k \times k$ matrix with (i, j) being the number of occurrences of the bigram $\langle \text{word}_i, \text{word}_j \rangle$. Note that the `bigram` function is not written in an efficient manner, and may take a bit of time to run. However, you will only need to run it once for a given text.

Preparing the data: As was discussed on a previous problem set, Zipf's law tells us that most of the words in a text will be exceedingly rare, and therefore will not have meaningful bigram statistics. In order to avoid this problem, we will follow Mintz et al.'s lead and restrict our attention to the 200 most frequent words. Use the function `recompute_bigram_rank` which takes as input the four outputs of `bigram` together with an n indicating the number of words desired. This produces a new word list, vector of marginal counts, and matrix of bigram counts.

Row i of the `bigram_count` matrix gives counts only for the words that follow word w_i . However, we would like to define the context of each word to be the counts of the words that both precede and follow w_i . Use the `bigram_count` matrix you have computed to define a new matrix `bigram_context` of size $k \times 2k$, where the first k values of each row are the preceding bigram counts, and the final k values are the following bigram counts.

k-means clustering: Run k-means clustering over `bigram_context` with a range of number of clusters (start with 25). Use the function `list_clusters(WORDLIST, CLUSTERS)` to explore the placement of words into different clusters. Do you get better qualitative results with a smaller (around 10) or larger (around 30) number of clusters?

Hierarchical clustering: Apply hierarchical clustering to `bigram_context`, using different distance measures and cluster linkages. Judging by a qualitative assessment, which ones appear to work best?