EE 486E: Homework 7

Spring 2015

42 points

Due: MON 20 APR

Directions for submission

Turn in a paper copy of your writeup at the beginning of class on the due date. Your writeup should consist of typed, numbered answers to each question followed by an addendum containing only the **PLOTS** requested in all-caps bold red font, followed by your code. Include two plots per page, each with a title indicating which problem it answers and axes labels that have proper units. In addition, upload your *Matlab code only* to Google Drive (`EE486E_CLASS\EE486E_DROPBOX\<YOURNAME>_EE486E_DROPBOX`). Your code may be split into more than one m-file but it must have a single main file, `RunHw<X>_<YOURNAME>.m`, that calls all of the others.

Spike Sorting

Although the dogma that "all spikes are created equal" is true at least to first approximation for intracellular recordings from a single neuron, multi-unit extracellular recordings often contain action potentials of variable wave-shapes and amplitudes. Apart from biophysical explanations for this, one easy-to-appreciate contributing factor is that the neurons represented in a multi-unit recording have different relative distances and orientations with respect to the electrode. Spike sorting is the task of assigning the action potentials in such a multi-unit recording to specific groups based upon a similarity measure. The underlying hypothesis is typically that the identified groups will each reflect the activity of one or a very small number of individual neurons. Reliable spike sorting can play a critical role in brain machine interface applications, particularly when only a small proportion of the recorded neurons are related to the output variable. Spike sorting is by nature an unsupervised learning task. In this homework assignment, you will explore some methods for doing it using intracranial EEG (iEEG) data recorded from a human subject.

1 Spike Detection and Feature Extraction

Start by loading the file `human_ieeg.mat`, a recording from the same patient whose data you worked with in the first homework assignment. The file contains two variables: `data`, which is the iEEG recording itself, and the structure `info`, which contains the sampling rate, among other things. Note that the recording has already been appropriately bandpass filtered to facilitate visualizing and detecting spikes, so there is no need for a filtering step.

1.1 Perhaps building on the detector you wrote in homework 1, write a function that detects all spikes exceeding a threshold of of six standard deviations above the mean of the iEEG signal. This time, however, instead of storing just spike peak-times, store 1 millisecond's worth of samples on either side of each spike's peak.[1] The resultant matrix (call it `spikes`) should be $m \times n$, where m is the number of detected spikes and n is the number of samples occurring in two milliseconds. **PLOT** the 68^{th} spike, putting the x-axis in milliseconds relative to the beginning of the 2 ms spike. (4 pts)

1.2 Make a time-domain **PLOT** of all detected spikes overlaid. In addition, compute the average spike waveform and overlay it on this same plot using a thick black line. (2 pts)

1.3 Compute an $m \times 2$ feature matrix with the first and second columns being the maximum and minimum voltage values, respectively, for each spike snippet. Report these two feature values for the 123^{rd} spike. (2 pts)

1.4 Now normalize the feature matrix by subtracting from each feature its mean and dividing by its standard deviation. Report these normalized feature values for the 205^{th} spike. (2 pts)

2 K-means Clustering

K-means clustering is one of the most widely-adopted techniques for unsupervised learning. The problem in K-means clustering[2] is to find a partitioning of n data points into K clusters that minimizes the total (across clusters) "within-cluster scatter." Within-cluster scatter is defined as the sum of squared Euclidean distances between all pairs of points in a given cluster, normalized by the number of points in the cluster. This is equivalent to minimizing the total (across clusters) "within-cluster variation": the sum of squared Euclidean distances between each point in a cluster and its centroid. In this section you will explore using K-means clustering to sort spikes.

2.1 Write your own code to implement K-means clustering. Do not use Matlab's `kmeans` function. Use squared Euclidean distance as your distance metric and set the number of clusters, k, to 3. Seed the algorithm by assigning rows 25, 116, and 262 (a random choice) of your normalized features matrix to be the initial cluster centers. Terminate the algorithm when the cluster assignments have stopped changing or 100 iterations have been reached, whichever comes first. After termination, make a scatter **PLOT** of the spikes in 2-dimensional feature space, coloring the spikes from classes 1, 2, and 3 green, blue, and red, respectively. (6 pts)

2.2 As you know from lecture, the algorithm you implemented in 2.1 is guaranteed to converge only to a *local* minimum of the objective function. Repeat the clustering you did in 2.1, this time using rows 124, 138, and 283 (another random choice) of the normalized features matrix as your initial cluster centers. Did you converge to the same local minimum as in 2.1? Again, make a scatter **PLOT** of the spikes in 2-dimensional feature space, coloring the spikes from classes 1, 2, and 3 green, blue, and red, respectively.[3] (2 pts)

[1] Ignore any spikes occurring too close to the beginning or end of the record to collect the requisite number of flanking samples.

[2] (in its strictly-defined standard form, which uses squared Euclidean distance as its similarity metric)

[3] Note that even if the cluster memberships were identical between 2.1 and 2.2, the colors (which reflect the numerical class labels, which in turn depend on the initial centers) need not correspond.

2.3 A common way to address the issue of initialization-dependent solutions is to run the algorithm multiple times using different random initializations and then choose the one that produces the smallest value of the objective function (i.e., the lowest of the local minima). Use Matlab's `kmeans` with $k = 3$ and the `'Replicates'` parameter set to 10 to run K-means with ten random initializations. Report the value of the objective function returned for the best solution (`sum(sumd)`) and **PLOT** the final clustering just as you did in 2.1 and 2.2. (3 pts)

2.4 We often don't know a priori how many clusters are contained within a data set.[4] Indeed, one of the main goals of clustering is sometimes to get a sense for this very thing, as a launching point for further analyses. Typical approaches to estimating k involve running the algorithm for a number of values of k and then using some criterion to select the "optimum." But since the local minimum of the objective function found at each k will generally decrease as k increases (until it becomes trivially zero when k reaches n and each observation is its own cluster), this is not a straightforward task. One common way to select k is to use the "elbow heuristic." To do this, **PLOT** the value of the local minimum found at each k as a function of k, and then choose the k corresponding to the most prominent "bend" (elbow) in the curve – the point at which the most abrupt change in the rate of decrease is observed. Run `kmeans` with values of k ranging between 1 and 10, and ten replicates per k value. Report the value of k you selected using the elbow heuristic.[5] (4 pts)

2.5 **PLOT** the "optimal" clustering you found in 2.4 in 2-dimensional feature space, with the clusters color-coded. (2 pts)

2.6 As mentioned in class, the data-partitioning arrived at via K-means-type algorithms is in general dependent on the chosen distance metric (among other things). To see this clearly, load the simulated feature matrix `'outlierData.mat'` into the workspace. In a 2×1 subplot, **PLOT** the color-coded clusterings you obtain with $k = 2$ and 10 replicates, for two cases: one that squared Euclidean distance (Matlab's default) and the other that uses Manhattan (a.k.a. L1 or city-block) distance as the distance measure. Explain the clustering results you obtained in view of the nature of the data set and the different distance metrics. (6 pts)

3 Principal Components Analysis

Principal Components Analysis (PCA) is a popular technique that can be used to reduce the dimensionality of a data set in a principled (no pun intended) way. It finds the directions in input space along which the *projected* data are maximally variable, subject to the constraint that successive directions are orthogonal to one another. These are the principal component directions.[6] The principal component directions are linear combinations of the axes that define the original

[4] In fact, this question so broadly framed is inherently ill-posed: the answer depends on the measured features, the chosen similarity metric and objective function, and perhaps subjective factors that can at best be captured by heuristics.

[5] A bit more formalism and automation can be brought to the notion of the elbow heuristic by, for example, using the "gap statistic" (Tibshirani R, Walther G, Hastie T. 2001. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society, Series B.* 63(2):411-423) to choose k, but for simplicity's sake we use a visual method in this assignment.

[6] They are the eigenvectors of the data covariance matrix; the corresponding eigenvalues are the variances of the projected data.

feature space, and the projections onto them are linear combinations of the original features. Dimensionality reduction is achieved if some number of the principal component directions (logically, ones whose projections have relatively small variance) is ignored. A new, smaller (i.e., lossily compressed) representation of a given observation can then be obtained by taking the projections onto the remaining principal component directions. In this section, rather than explicitly computing features such as the peaks and troughs of the spikes, you will generate features by performing PCA on the spikes matrix. You will then examine the performance of K-means using the PCA-derived feature-set.

3.1 Use Matlab's `pca` to perform principal components analysis on the original $m \times n$ spike data matrix. The syntax `[directions,projections,variances] = pca(spikes);` will return the principal component direction vectors, the coordinates of the projections of each spike onto each of the principal component directions, as well as the variances of the projections of all spikes onto each of the principal component directions. The values returned by Matlab are sorted in order from largest to smallest variance. **PLOT** the cumulative proportion of the variance accounted for as a function of number of principal components retained. (3 pts)

3.2 What is the minimum number, M, of principal components you need to retain to preserve at least 90% of the data variance? (2 pts)

3.3 Derive a new features matrix that consists of only the first M principal component projections for each spike, per your analysis in 3.2. Use `kmeans` as in 2.4 to perform K-means clustering and select the optimal number of clusters, k, using the elbow heuristic. **PLOT** your results in the 2-dimensional feature space defined by the first two principal component projections, with clusters color-coded. (3 pts)

3.4 **PLOT** Finally, make a time-domain plot of all spikes overlaid, as you did in 1.2, only now coloring each spike green or blue according to its cluster membership. Compute an average spike waveform for each cluster, and overlay these two average traces on the same plot using thick black lines. (3 pts)