Lab 6: Clustering algorithms

Introduction

Neural data sets are large, diverse, and high-dimensional. These properties make manual sorting of neural data difficult and impractical. In this lab, we will introduce two simple clustering algorithms that can be used to interpret neural data. Here, we will use k-means clustering and Gaussian mixture models to sort a data set of spikes into different units. We will then discuss some limitations of these algorithms and potential solutions.

Software

This lab must be completed using MATLAB. You will also need the principal components of the units in spikes.mat from Lab 5.

Part 1) k-means clustering

k-means is an iterative clustering algorithm based on the simple assumption that points from the same group are similar each other. The algorithm minimizes the objective function

$$J = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\left|\left|\vec{x}_n - \vec{\mu}_k\right|\right|^2, \quad where\ r_{nk} = \begin{cases} 1, & if\ x_n\ is\ in\ group\ k \\ 0, & if\ x_n\ is\ not\ in\ group\ k \end{cases}$$

Where N is the number of samples, K is the number of clusters, and $\vec{\mu}_k$ is the centroid of cluster k. Note that $\left|\left|\vec{x}_n - \vec{\mu}_k\right|\right|^2$ is the squared Euclidian distance between $\vec{x}_n$ and $\vec{\mu}_k$.

To start, we will implement the standard k-means algorithm from scratch. For Parts 1 and 2, do not use any of MATLAB's built-in clustering functions.

1. Calculate the principal components of the units in spikes.mat. Pull out the first two principal components of each sample. We will use this as our data set for this lab. That is, rather than using the original spike data, we will analyze just the first two principal components of the spikes and treat that as our data. (Your data matrix, X, should be an $N\times 2$ matrix.)
2. Plot the first two principal components against each other. How many discernable clusters do you see in the data? Set K equal to the number of clusters.

   Note: It may be easier to visualize the data if you only plot every third point or fourth point.

3. To initialize the algorithm, pick K random points from the data set as starting centroids for your clusters. Make sure the same point is not used multiple times.
4. For each point in your data set, calculate its distance from each centroid.
5. Assign each point to the cluster for which the centroid is closest. That is, if μ_k is closest to point x_n, then assign point n to cluster k.

6. Recalculate the centroids for each cluster by taking the mean of the points in each cluster, so that

$$\vec{\mu}_k = \frac{1}{N_k}\sum_{n}^{N_k}\vec{x}_n$$

   Where N_k is the number of points in cluster k.

   Note that the new centroids do not necessarily have to lie on any point in the data.

7. Repeat steps 4-6 until the clustering reaches convergence, i.e., none of the points change cluster assignment.
8. Create a plot like you did in Step 2, but with each point colored according to its cluster assignment.
9. Go back to the original spike data and plot the average spike from each cluster, with standard deviation visible as a shaded area. Use the shadedErrorBar function provided (make sure to add it to Matlab's path using addpath).
10. Rerun your k-means algorithm a few times with different starting points. Are the results the same each time?
11. Consider some limitations of the k-means algorithm. What assumptions does it require? What kinds of cluster shapes work or do not work well? How does it treat outliers?

Part 2) Extensions of k-means

k-means, while simple to implement and relatively easy to understand, has some significant limitations. Here, we will explore some extensions of the k-means algorithm.

1. Instead of calculating the cluster centroids with the mean, set the medoid of each cluster (the median coordinate) as the cluster centroid at each iteration. This is called the k-medoids algorithm. Plot your results and rerun the algorithm a few times to get a good representation of its performance. Compare to results from Part 1. What advantages does this offer over k-means?
2. Going back to the k-*means* algorithm, consider a different similarity metric. Rather than using the Euclidean distance between each point and the cluster centroids, use the Mahalanobis distance, defined as

$$d_M = \sqrt{(x-\mu_k)^T S_k^{-1}(x-\mu_k)}$$

   Where S_k is the covariance matrix of points in cluster k. The Mahalanobis distance normalizes distances from centroid k to the covariance of points within cluster k. For example, if cluster k spans a wide range of x values, distances from centroid k are normalized in the x dimension to account for the cluster's inherent variance. The new objective function will take the form

$$J = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk} d_M^2, \quad where\ r_{nk} = \begin{cases} 1, & if\ x_n\ is\ in\ group\ k \\ 0, & if\ x_n\ is\ not\ in\ group\ k \end{cases}$$

Feel free to use the MATLAB built-in: `dM = mahal(Xall,Xk)`, where `Xall` is all of your data, `Xk` is all of the points in cluster k of your current cluster assignments, and `dM` is an $N \times 1$ vector of Mahalanobis distances. Be sure to set a maximum number of iterations of 5 (otherwise, it likely won't converge with our data).

Note that you will need to initialize cluster assignments with the traditional k-means approach before applying the Mahalanobis distances. (Reasonable starting clusters are necessary for calculating a meaningful covariance.)

Rerun the algorithm a few times to get a good representation of its performance, and keep a plot that shows good separation of clusters. Compare to results from Part 1. What advantages does using the Mahalanobis distance offer over traditional k-means clustering?

Part 3) Gaussian mixture model

k-means and k-medoids are both partitional clustering algorithms that break up the dataset into distinct, non-overlapping groups without any underlying assumptions about the distribution of data. It is not unreasonable, however, to assume that spikes within each cluster are normally distributed about the cluster centroid. Gaussian mixture models (GMMs) are a probabilistic approach to clustering that makes this assumption. While k-means assigns each point to the *nearest* cluster, GMMs assign each point to the *most likely* cluster, based on the underlying distribution calculated from the data.

GMMs are much more complex to implement, since not only do we need to find the centroid for each cluster, we also need to find the covariance of each cluster. This turns out to be a fairly involved process involving optimization algorithms beyond the scope of this class, so we'll just use the MATLAB built-in for implementation.

Again, we'll use the first two principal components from spikes.mat as our data set.

1. First define some function options with the following command:

   ```
   options = statset('display','final');
   ```

2. To run the GMM algorithm, choose a number of clusters, K, and use the following command:

   ```
   obj = gmdistribution.fit(X,K,'options',options);
   ```

3. The cluster assignments can be retrieved using the following command:

   ```
   clusters = cluster(obj,X);
   ```

   Plot the data in a scatter plot, with different colors for each cluster. Again, it might help to plot only a fraction of the data for ease of visualization.

4. The cluster centroids can be retrieved using the command:

```
hold on     % this keeps MATLAB from overwriting the scatter plot
centroids = obj.mu;
```

   Mark the cluster centroids on your scatter plot with 'X' symbols.

5. Plot the contours of the Gaussian distributions with the following commands:

```
h = ezcontour(@(x,y)pdf(obj,[x y]),[-15 15],[-15 15]);
```

6. Run the GMM algorithm with different values of K until you feel that all clusters in the data have been captured by at least one cluster.
7. What are some limitations of GMMs? What advantages and disadvantages does it have when compared to k-means? What post processing steps are still required if this is your final automated output?

Guidelines for Lab Report (on Labs 5 and 6 together)

Introduction: The introduction should be one paragraph long summarizing the motivation for developing the tools used in this lab and what they can be used for, along with a brief summary of everything you will show in this lab report.

Methods: From Lab 6, there should be methods paragraphs (and diagrams where necessary) on:

1. Assumptions of the algorithms used
2. How the algorithms were implemented

Include the code as an Appendix to your report. Cite sources for any values used in your models.

Results: You should include the following in your Results:

1. The outcome of each algorithm (cluster plots for k-means, k-mediods, k-means with Mahalanobis distance, and Gaussian mixture model (4 total); plots of average spikes with standard deviation for k-means.
2. The limitations of each algorithm.
3. How the algorithms compare to each other.

   Include all figures produced by MATLAB that could help explain and illustrate your findings.

Discussion: Should be 2~ paragraphs long describing what postprocessing steps are still needed after the steps you implemented in class. Also, discuss what these tools are useful for.

This report will be combined with Lab 5, to create one cohesive report. The report (not including Appendix) should be no longer than 4 pages. Use 12 pt. font and 1.15-1.5 line spacing. If your text is over the 4-page limit with figures, you can move your figures to an appendix section that goes beyond the 4-page limit.

Please upload your report to Canvas.