Lab 7: Supervised classifier algorithms

Introduction

In the last lab, we learned how to use *unsupervised* algorithms to automatically sort samples from large datasets into different groups. In this lab, we will use *supervised* algorithms to classify data into groups. These algorithms are 'supervised' because they need to be trained on labeled data. (Note that none of the Lab 5 data needed to be labeled for the clustering algorithms to work.) This lab introduces three supervised classifier algorithms: Naïve Bayes, linear discriminate analysis, and support-vector machines. By the end of this lab, you should be familiar with how to use all three and have an understanding of how they are related.

<u>Note:</u> For calculating overall classification accuracy, take the sum of accurate classifications divided by the number of predictions.

Software

This lab should be completed using MATLAB. (+2 pts if it is completed in Python, without syntax support).

Part 1) Naïve Bayes

Naïve Bayes classifiers are a family of classifiers that are natively able to decode a set of input features into one of several different output classes, given some assumptions about the distribution of features. For example, a Naïve Bayes classifier could take firing rates from different neurons and predict whether a monkey is moving his hand up, down, left, or right. (Note that output classes are distinct, discrete, and exclusive.) These classifiers take the form

$$\hat{f}(x) = \arg\max_{k}\left[\prod_{\mathrm{j}=1}^{\mathrm{D}} \hat{g}_k^{(j)}(x)\right]$$

where x is a vector of features (e.g., firing rates of different neurons), D is the number of features in x, and $\hat{g}_k^{(j)}$ is the estimated probability distribution of feature j for class k (e.g., the distribution of neuron #14's firing rates when the monkey is pointing left). (If you're unfamiliar with this notation, the output $\hat{f}(x)$ takes the value of k associated with the largest value in the $\arg\max_{k}()$ operation.)

Note that Naïve Bayes classifiers assume that the different features of sample x are independent from one another, for a given class. For example, if firing rates are recorded from 30 different neurons on a Utah array of a monkey performing a grasp, the algorithm assumes that each of the neurons' firing rates are completely independent of one another. This is usually untrue, but turns out not to matter too much for decoder performance.

Here, we're going to build a Naïve Bayes classifier from scratch and measure its performance. We'll be using a set of spike data recorded from a monkey performing reach tasks. In order to train and test our classifier, we need to establish training and test datasets.

1. Load data from firingrate.mat. The dataset contains firing rates recorded from 95 different neurons and 8 different reach directions, with 182 samples for each neuron-direction combination.
2. Split the data in half, with the first half labeled for training and the second reserved for testing.

We begin by *training* our classifier. Using the training data, we will estimate key parameters of the model that the classifier will use to make predictions.

3. Use the training dataset to estimate the mean, λ, for each feature-class (neuron-direction) pair. We will use this to estimate the distribution of spike rates for each feature-class pair, $\hat{g}_k^{(j)}$, by assuming a Poisson distribution. The Poisson distribution is a reasonable assumption for spike data, as count variables very often fall under Poisson distributions.

Now that we've estimated the necessary parameters for the classifier, we can *test* the classifier using the test data. We do this to make sure we're not overfitting the classifier to the training data.

4. To predict the reach direction of a given sample i in the test dataset:
    a. For each feature, $X_i^{(j)}$ (the firing rate of neuron j in sample i), use $\hat{g}_{k=1}^{(j)}$ to find the probability density of observing $X_i^{(j)}$ when the monkey is reaching in direction #1. The MATLAB function `poisspdf()` may be useful here.
    b. Once you've calculated $\hat{g}_{k=1}^{(j)}(X_i)$ for every feature, we would normally multiply them together get $\prod_{\mathrm{j=1}}^{\mathrm{D}} \hat{g}_{k=1}^{(j)}(X_i)$. Note that this is the argument that needs to be evaluated for each class k in the equation shown in the introduction for Part 1. However, this ends up being a very, very small value that can cause floating point errors. To avoid that issue, we'll take the log of each probability and add them instead to get

    $$\sum_{j=1}^{D} \log \hat{g}_{k=1}^{(j)}(X_i)$$

    This creates much more reasonable values.
    c. Calculate this argument for each class k for sample i.
    d. Assign sample X_i to class k, where k is the class for which $\sum_{j=1}^{D} \log \hat{g}_k^{(j)}(X_i)$ is the largest.
5. Repeat this to predict the reach direction of every sample in the test dataset.
6. Calculate the accuracy of the decoder. How does its performance compare to chance?

To make doubly sure that we're not fooling ourselves with the classifier's performance, we can also apply it to a spoofed dataset that looks very similar to the original, but contains no meaningful information.

7. Find the overall mean of the full dataset's spike rates, across the different movement directions.
8. Create a second set of data with the same size and overall distribution (assume Poisson) as the original. I.e. generate a set of Poisson distributed firing rates using those means, drawing from a distribution that has the same mean for every target.
9. Test your classifier on this fabricated dataset. What is the performance? Is this what you expect?

Part 2) Linear discriminant analysis

Linear discriminant analysis (LDA) is a supervised classifier algorithm that can be thought of as a hyperplane separating two clusters of data. Everything that falls on one side of the hyperplane is in one class, and everything falling on the other side is in the other class. Instead of writing an LDA algorithm from scratch, we'll use the MATLAB built-in `classify()`. For this part of the lab, we'll use a set of ECoG data from a finger movement task.

1. Load data from ecogclassifydata.mat. The dataset contains average power values from 0.5 sec prior to movement to 1.5 sec after movement in the 60-120 Hz band. There are 38 trials with data recorded from 27 electrode pairs. The 'group' variable indicates which finger was squeezing (1 is rest, 2 is thumb, 3 is index, 4 is middle, 5 is ring and pinkie).
2. Since we don't have a lot of trials, creating independent training and test sets is impractical. Instead, we will use leave-one-out cross-validation. To start, designate the first trial as the test data and the rest of the data as training data.
3. Use the MATLAB built-in `testClass = classify(testData, trainingData, trainingClasses, 'linear')` to make an LDA prediction of the test sample's class.
4. Repeat this process such that every sample takes a turn at being the test data, with all other samples acting as training data. Keep track of the prediction for each test case.
5. What is the overall percentage accuracy of this algorithm?
6. In addition to overall accuracy, it is also important to know where errors occur. Create a confusion matrix showing what percentage of the time a finger is classified as itself (on diagonal entries) and what percentage of the time it was classified as another finger (on off-diagonal entries). See Figure 3 of Chestek et al. 2013, 'Hand posture classification using electrocorticography signals in the gamma band over human sensorimotor', for reference on what this data should contain. MATLAB's `pcolor()` may be useful here.

Guidelines for Lab Report (on Labs 7, 8, and 9 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 7, there should be methods paragraphs (and diagrams where necessary) on:

1. Assumptions of the classifiers used (NB, LDA)
2. How the classifiers 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 classifier (including 1 plot from part 2 and 4 plots from part 4).
2. The limitations of each classifier.
3. How the classifiers 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 you could use these tools for in the future.

This report will be combined with Labs 8 and 9 to create one cohesive report. The report (not including Appendix) should be no longer than **5 pages**. Use 12 pt. font and 1.15-1.5 line spacing. If your text is over the 5-page limit with figures, you can move your figures to an appendix section that goes beyond the 5-page limit. Please upload your report to Canvas.