
Homework 10

Due: 7:00pm, Friday, **April 26** via Gradescope

Problem 10.1 (**Video 9.1, 9.2, 9.3, 9.4**, **Lecture Problem**)

You measure the sulfate concentration in the local water reservoir over 9 consecutive days and obtain values $X_1, \ldots, X_9$, which are assumed to be i.i.d. Gaussian. (The units are mg/L and omitted below.) The sample mean is $M_9 = 6.1$ and the sample variance is $V_9 = 0.36$.

Let W have a t-distribution with 8 degrees-of-freedom. You can assume the following values are available:

- $F_W(-1.4) = \Phi(-1.3) = Q(1.3) = 0.1, \;\; F_W(1.4) = \Phi(1.3) = 0.9$
- $F_W(-1.9) = \Phi(-1.6) = Q(1.6) = 0.05, \;\; F_W(1.9) = \Phi(1.6) = 0.95$
- $F_W(-2.3) = \Phi(-2.0) = Q(2.0) = 0.025, \;\; F_W(2.3) = \Phi(2.0) = 0.975$

(a) Construct a confidence interval for the mean with confidence level 0.9.

(b) Is your sample significantly different from the baseline concentration $\mu = 5.4$ at a significance level of 0.05? Justify your approach and support your answer numerically.

(c) Say you also go out on the 10^{th} day and collect measurement $X_{10} = 5$. What is the new sample mean M_{10}?

Problem 10.2 (**Video 9.1, 9.2, 9.3, 9.4**)

In an unnamed course in the College of Engineering, there are two sections each with 128 students. The exam grades for Exam 1 for Section A averaged 80 points, with a sample variance of 60 points squared. For Section B, the exam average was 83 points, with a sample variance of 68 points squared. Our null hypothesis is that the grades in each section are i.i.d. samples from a Gaussian(μ, σ^2) distribution, with unknown mean and unknown variance, so that there is no statistical difference in performance among students across sections. Thus, the null hypothesis is that the students in both sections have grades with the same average and variance. We want to examine the validity of this null hypothesis, and see whether the exam scores suggest rejecting the null hypothesis as valid with a given significance level.

Possibly useful data:

$$2\Phi(-1.645) = 0.1, \;\; 2\Phi(-1.960) = 0.05; \;\; 2\Phi(-2.576) = 0.01; \;\; 2\Phi(-3) = 0.00269$$

$$2F_{T_{254}}(-1.651) = 0.1, \;\; 2F_{T_{254}}(-1.969) = 0.05; \;\; 2F_{T_{254}}(-2.595) = 0.01; \;\; 2F_{T_{254}}(-3) = 0.002968$$

(a) What type of statistical test should you use to reject or not reject H_0?

(b) Compute the average covariance to use in the test.

(c) Write an expression for the test statistic to use for the test in (a).

(d) Do you reject H_0 at significance level 0.01? Do not merely answer yes or no; clearly state the comparison that you are making for your conclusion.

Problem 10.3 (**Video 9.1, 9.2, 9.3, 9.4**)

The weights in ounces of coffee in bags produced by Rhett's Roastery $X_1, X_2, \ldots, X_{25}$ can be assumed to be i.i.d. from some single underlying Gaussian distribution. The null hypothesis H_0 is that the mean weight of coffee in a bag is 12 ounces.

Suppose you observe sample mean $M_{25} = 11.92$ and sample variance $V_{25} = 0.04$.

Possibly useful:

$$2\Phi(-1.64) = 0.1, \qquad 2\Phi(-1.96) = 0.05, \qquad 2\Phi(-2.57) = 0.01,$$

$$2F_{T_{24}}(-1.71) = 0.1, \qquad 2F_{T_{24}}(-2.06) = 0.05, \qquad 2F_{T_{24}}(-2.80) = 0.01,$$

(a) What type of statistical test should you use to reject or not reject H_0?

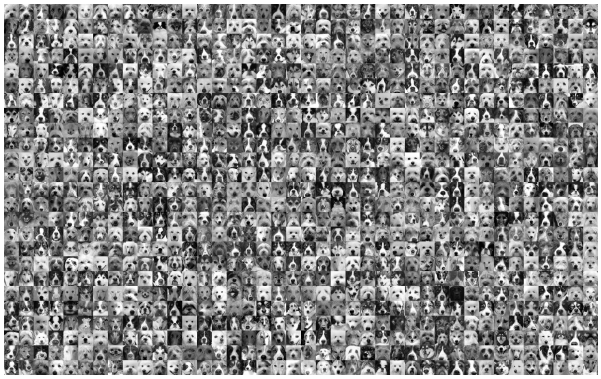
(b) Write an expression for the test statistic to use for the test in (a).

(c) Do you reject or not reject H_0 at significance level 0.05? Do not merely answer yes or no; clearly state the comparison that you are making for your conclusion.

(d) Find a 90% confidence interval for the mean.

Problem 10.4 (**Video 10.1 - 10.3**, **Programming Problem**)

In this problem, we are going to implement a machine learning application to classify images into one of two classes. We will be using some of the code that we have developed in past programming problems. The data we will be using is a collection of 64×64, grayscale images, with 1000 examples of cats and 1000 examples of dogs. The data sets are shown below (somewhat compressed).



We have provided two scripts, `HW10_matlab.m` and `HW10_python.ipynb`, that you can use to build functions and solve this problem. To get started, download and decompress `petdataset.zip`,

and place the resulting the resulting `catsfolder` and `dogsfolder` into your working MATLAB or Python directory. The provided scripts will look for the folders, not the files in the folders, so be careful with this step. Be particularly careful if your working directory is in cloud storage.

The provided function `read_cats_dogs` will scan through `catsfolder` and return a data matrix $\mathbf{X}$ of dimension 2000×4096, where each length-4096 row consists of a single cat or dog image image "unwrapped" from a 64×64 square into a 1×4096 row vector. It also returns a vector `Y` of dimension 1×2000 with the labels of each image, where $Y_i = 0$ indicates that row i of $\mathbf{X}$ is a cat, and $Y_i = 1$ indicates that the row is a dog.

The provided function `show_image(X,i)` can be used to display the i^{th} row of `X` as a 64×64 image. Use it to explore the dataset if you like.

(a) Your first task is to read the data from the cats and dogs folder, and to fill in the function `labeled_mean_cov(X,Y,desiredlabel)`. This function takes in a data matrix `X` and a label vector `Y`, as well as a desired label (either 0 or 1). It returns the number of rows in the data matrix with the desired label, the mean vector of the data rows with the desired label, and the covariance matrix of the data rows for the desired label. The script will use this function to compute the average cat and the average dog, and print pictures of both the average cat and the average dog.

To turn in: a pdf of your function ***`labeled_mean_cov`****, and pictures of the average cat and the average dog.*

(b) The next task is to extend the `closest_average` function of HW 8 to take in a full data matrix `X`, with two average vectors `mu0, mu1` (the average cat and the average dog computed in (a)), and classify each of the rows in this data matrix as 0 if `mu0` is closer, as 1 if `mu1` is closer. It outputs a 1-vector `guess` that contains the decision for each row of the data matrix. In the case of a tie, it should make the decision 1.

Complete this `closest_average` function, and use it in the script with the provided `error_rate` function to compute the error rate for the closest average classifier on the full data set.

To turn in: a pdf of your function ***`closest_average`****, and the error rate for the closest average classifier.*

(c) We are now going to design classifiers based on Gaussian probability models, similar to what we did in HW 8. However, the data is has too many dimensions (4096), so we are going to use Principal Component Analysis to represent the data in fewer dimensions. To save computation time later, we first compute the mean and covariance of the data matrix, and then compute the eigenvalues and eigenvectors of the covariance matrix. When using Python, we use the command `np.linalg.eigh` to obtain the eigenvalues and eigenvectors, in order to avoid issues with complex numbers.

Your next task is to fill in the code for the function `dimensionality_reduction` that takes in a data matrix `DataMatrix`, the mean of that data matrix `mu0`, the matrix of eigenvectors and the vector of eigenvalues for the covariance of the data matrix `V, D`, and the desired reduced dimension k. The function returns the k-dimensional PCA-reduced data matrix $\mathbf{X}_{\text{reduced}}$. Note that this task is nearly identical to the 2D data visualization task from Homework 7, except there you only retained the top 2 eigenvectors, rather than the top k. You should be able to adapt the `visualize` Homework 7 code to this problem.

To turn in: a pdf of your function ***`dimensionality_reduction`****.*

(d) We are now going to develop a pair of Gaussian classifiers. In Homework 8, we examined the maximum likelihood (ML) rule for vector Gaussian data. One of the special cases assumed that the mean vectors $\underline{\mu}_0$ and $\underline{\mu}_1$ were different and that the covariance matrices were the same $\mathbf{\Sigma}_0 = \mathbf{\Sigma}_1 = \mathbf{\Sigma}$. We estimated the shared covariance matrix by pooling our estimates under each label:

$$\hat{\mathbf{\Sigma}} = \frac{1}{n_{\text{train}} - 2}\big((n_{\text{train},0} - 1)\hat{\mathbf{\Sigma}}_0 + (n_{\text{train},1} - 1)\hat{\mathbf{\Sigma}}_1\big) \ .$$

In the machine learning literature, the resulting decision is known as Linear Discriminant Analysis (LDA), and uses the log-likelihood decision rule to avoid underflow. The decision rule is:

$$D_{\text{ML}}(\underline{x}) = \begin{cases} 1 & (\underline{x} - \underline{\mu}_1)^{\mathsf{T}}\hat{\mathbf{\Sigma}}^{-1}(\underline{x} - \underline{\mu}_1) \leq (\underline{x} - \underline{\mu}_0)^{\mathsf{T}}\hat{\mathbf{\Sigma}}^{-1}(\underline{x} - \underline{\mu}_0) \\ 0 & \text{otherwise.} \end{cases}$$

Fill in the code for the `LDA` function which takes as input a data matrix `Xrun`, the two mean vectors μ_0, μ_1, and the pooled covariance matrix $\hat{\mathbf{\Sigma}}$ and classifies each row of the data matrix as 1 or 0, and returns the classifications in a vector `guesses`. You will call this function using the reduced dimension data with 30 dimensions, and compute the error rate for this reduced dimension classifier.

To turn in: a pdf printout of your LDA function, and the reported error rate for the LDA classifier on the reduced dimension data.

(e) For this part, we are going to use the log-likelihood decision rule for Gaussian random variables with different means and different covariances. The log-likelihood decision rule is

$$D_{\text{ML}}(\underline{x}) = \begin{cases} 1 & \log(\det(\mathbf{\Sigma}_1)) + (\underline{x} - \underline{\mu}_1)^{\mathsf{T}}\mathbf{\Sigma}_1^{-1}(\underline{x} - \underline{\mu}_1) \leq \log(\det(\mathbf{\Sigma}_0)) + (\underline{x} - \underline{\mu}_0)^{\mathsf{T}}\mathbf{\Sigma}_0^{-1}(\underline{x} - \underline{\mu}_0) \\ 0 & \text{otherwise.} \end{cases}$$

This decision rule is known as Quadratic Discriminant Analysis in machine learning. Fill out the code for the `QDA` function which takes as input a data matrix `Xrun`, the two mean vectors μ_0, μ_1, and the two covariance matrices $\hat{\mathbf{\Sigma}}_\mathbf{0}, \hat{\mathbf{\Sigma}}_\mathbf{1}$ and classifies each row of the data matrix as 1 or 0, and returns the classifications in a vector `guesses`. To avoid underflow, you need one more numerical trick to get this to work, even after dimensionality reduction. The determinant of a matrix is equal to the product of its eigenvalues. This fact, coupled with the fact that $\log a \cdot b = \log a + \log b$, allows us to avoid computing the determinant directly, and instead calculate

$$\log(\det(\mathbf{\Sigma}_0)) = \sum_i \log(\lambda_{0,i}) \qquad \log(\det(\mathbf{\Sigma}_1)) = \sum_i \log(\lambda_{1,i})$$

where the $\lambda_{0,i}$ are the eigenvalues of $\mathbf{\Sigma}_0$ and the $\lambda_{1,i}$ are the eigenvalues of $\mathbf{\Sigma}_1$. We provide the start of this computation in the provided function `QDA`, which you will complete.

You will call this function using the reduced dimension data with 30 dimensions, and compute the resulting error.

To submit: A pdf of your completed QDA function, and the reported error rate for the QDA classifier on the reduced dimension data.

(f) We are going to implement one additional classifier: a nearest-neighbor classifier. The nearest neighbor classifier takes as inputs a training data matrix `Xtrain` with data labels

`Ytrain`, and a test data matrix `Xrun` with each row representing a data vector to be classified. Given an input vector $\underline{x}$ which is a row of `Xrun`, it searches through the training data to find the index of the closest training vector

$$i_{\text{closest}} = \underset{i=1,\ldots,n_{\text{train}}}{\arg\min} \|\underline{x} - \underline{X}_{\text{train},i}\|$$

and then returns as its guess the label of this vector,

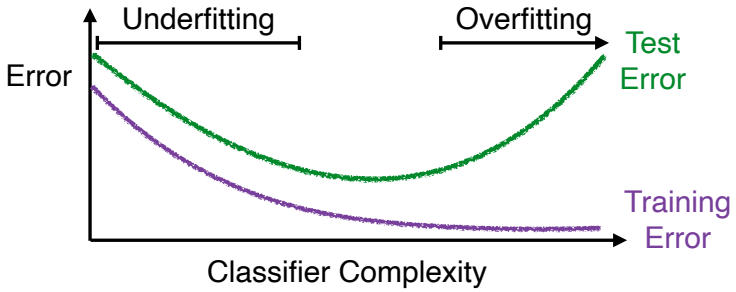
$$D_{\text{NN}}(\underline{x}) = Y_{\text{train},i_{\text{closest}}}.$$

Fill in the code for the `nearest_neighbor` function that returns the guesses for the data in the rows of `Xrun`.

The script will partition the `Xreduced` data into two data matrices `Xrun, Xtrain` and with appropriate data labes `Yrun, Ytrain`. The training data will be 80% of the data, and the test data willl be 20%. It will then call the nearest neighbor function to obtain the estimated data labels, and compute the error rate.

To submit: A pdf of your completed ***`nearest_neighbor`*** *function, and the reported error rate for the nearest neighbor classifier on the reduced dimension data.*

(g) We have now completed and tested four different classification algorithms. However, we have evaluated their performance on the same data they were trained in. The provided scripts will run a sequence of tests to evaluate the classifier performance under different random partitions of the data into training sets and test sets. We will also evaluate the classifiers under different levels of data aggregation, to observe the phenomenon of overfitting. At a high level, we expect to see something like the following illustration. As the complexity of the classifier increases, both the training and test error decrease. However, at a certain point, the test error starts to increase again while the training error continues to decrease. This is the regime where overfitting occurs. In our setting, the classifier complexity will be represented by the dimension k used in the dimensionality reduction algorithm.



Run the overall script to obtain a plot of the training and test error rates for the LDA, QDA and Nearest Neighbor classifiers at dimensions $k = 10, 25, 50, 100, 250, 500$. **This will take about 2 minutes per fold in Python, perhaps more depending on how many loops you wrote. Given 5 folds, do something else while this runs.**

What happens to training error rate for the different algorithms as the dimension increases? What happens to the test error rate as the dimension increases? For each algorithm, estimate the number of parameters used as a function of k, and determine the value of k that yields the lowest test error. For instance, the closest average classifier with $k = 30$ features has 60 parameters, corresponding to the specification of the two centers $\mu 0, \mu 1$.

Record the best values of k and the estimates of the parameters needed as a function of k for each of the algorithms.

To submit: A plot of the test error and training error for all the algorithms, as generated by the script, an estimate of the number of parameters in each algorithm as a function of k, and a brief discussion of the values of k that yield the lowest test error for each algorithm, as well as the test and training errors for those values.