P556 Homework 2

Fall 2022

Due on Oct 21th, 11:59pm

Please follow the instruction for how to submit your homework to GitHub (in the module additional resources). For HW1, there are students not following the instructions, which dramatically delay our grading process.

Question 1: K-Means Clustering (20 points)

In this question you will use K-means clustering to try to diagnose breast cancer based solely on a Fine Needle Aspiration (FNA), which as the name suggests, takes a very small tissue sample using a syringe (Figure 1).

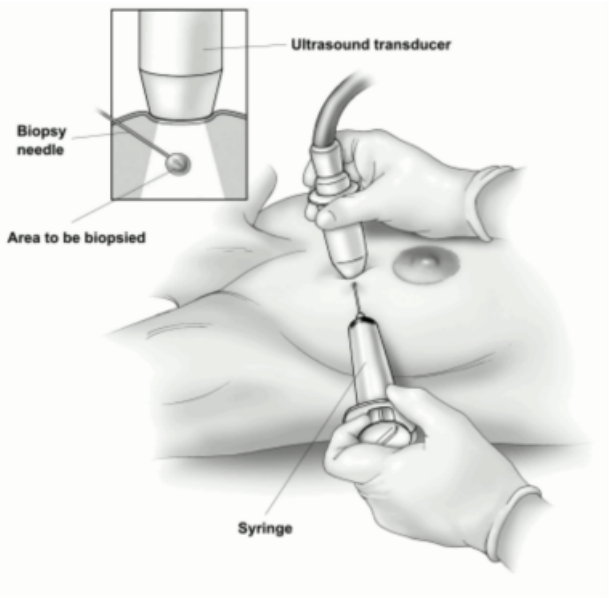

Figure 1: Fine Needle Aspiration using ultrasound. © Sam and Amy Collins.

To this end we will use the Wisconsin Diagnostic Breast Cancer dataset, containing information about 569 FNA breast samples [1]. Each FNA produces an image as in Figure 2. Then a clinician isolates individual cells in each image, to obtain 30 characteristics (features), like size, shape, and texture. You will use these 30 features to cluster benign from malign FNA samples.

Questions:

- Implement a function that performs K-means clustering by yourself from scratch.
- Load the Wisconsin Diagnostic Breast Cancer dataset (see the link below). You should obtain a data matrix with D = 10 features and N = 699 samples. Attribute information in Table 1. (Please refer to the data description on the webpage if there is anything updated or changed.)
- First without the class label, run your algorithm. Then compare with the class labels, what is the accuracy of your algorithm?
- Run your algorithm several times, starting with different centers. Do your results change depending on this? Explain.

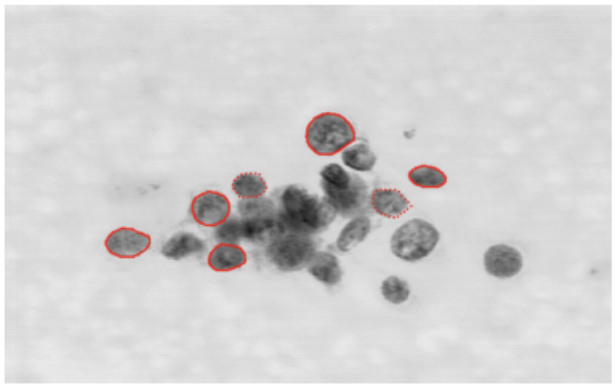

Figure 2: Breast sample obtained by FNA.

Attribute	Domain
Sample Code Number	id number
Clump Thickness	1-10
Uniformity of Cell Size	1-10
Uniformity of Cell Shape	1-10
Marginal Adhesion	1-10
Single Epithelial Cell Size	1-10
Bare Nuclei	1-10
Bland Chromatin	1-10
Normal Nucleoli	1-10
Mitoses	1-10
Class	2 for benign, 4 for malignant

Table 1: Caption

- Can you get a better result using a supervised method? (You don't need to implement the supervised method from scratch. You can use a package or function provided by Python).

[1] O. Mangasarian, W. Street and W. Wolberg, Breast cancer diagnosis and prognosis via linear programming, Operations Research, 1995. Dataset available at https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic)

Question 2: Fisher Linear Discriminant (40 points)

In this question, we will build up the Fisher Linear Discriminant step by step and generalize it for multiple classes. We will first recap the simplest case, $K = 2$. In general, we can take any D-dimensional input vector and project it down to D'-dimensions. Here, D represents the original input dimensions while D' is the projected space dimensions. For our purpose, consider D' less than D.

If we want to project the data into $D' = 1$ dimension, we can pick a threshold t to separate the classes in the new space. Given an input vector x:

- x belongs to class C1 (class 1), if predicted $y >= t$, where $y = w^T x$
- otherwise, it is classified as C2 (class 2).

In figure below, the original data with $D = 2$ and we want to reduce the original data dimensions from $D = 2$ to $D' = 1$. First we can compute the mean vectors of the two classes and consider using the class means as a measure of separation. In other words, we want to project the data onto the vector w joining

the 2 class means.

Note that during such projection, there can be information loss. In our case, clearly in the original space, the two classes can be seperated by a line. However, after projection, the yellow ellipse demonstrates overlapping of the two classes. For FLD, it looks for a large variance among the dataset classes and a small

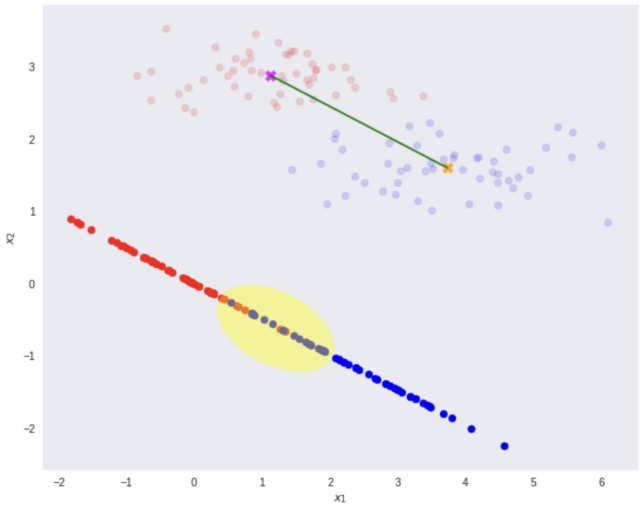

Figure 3: FLD projection

variance within each of the dataset classes. The project w could be: $w \propto S_w^{-1}(m_2 - m_1)$ and please refer to our slides for the meaning of each symbol and index here.

The above is a short summary of what we discussed during our lecture for 2 classes case. Now we will generalize the model to $K > 2$ classes. Here, we need generalization forms for the within-class and between-class covariance matrices, which is similar to the definitions for the two classes case:

- Mean vector: $m_k = \frac{1}{N_k} \sum_{i \in C_k} x_i$
- Within class variance: $S_k = \sum_{i \in C_k} (x_i - m_k)(x_i - m_k)^T$
- Total within class variance: $S_W = \sum_{k=1}^{K} S_k$
- Between class variance: $S_B = \sum_{k=1}^{K} N_k (m_k - m)(m_k - m)^T$

Then we can have the project vector:

$$w = max_{D'}(eig(S_W^{-1} S_B)) \tag{1}$$

For between-class covariance S_B estimation, for each class $k = 1, 2, 3, ..., K$, take the outer product of the local class mean m_k and the global mean m, and then scale it by the number of instances in class k.

The maximization of the criterion of FLD, we need to solve w with an eigendecomposition of the matrix multiplication between S_W^{-1} and S_B. To find the projection vector w, we then need to take the D' eigenvectors that correspond to their largest eigenvalues. For example, if we want to reduce our input dimension from $D = 784$ to $D' = 2$.

Now we have all the necessary settings for building a FLD for multiple classes. To create a discriminant, we first can use a multivariate Gaussian distribution over a D-dimensional input vector x for each class K as:

$$N(x|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{\frac{1}{2}}} exp - \frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu) \tag{2}$$

Here μ (the mean) is a $D-$dimensional vector. Σ (sigma) is a $D \times D$ covariance matrix. $|\Sigma|$ is the determinant of the covariance.

Using the projected input data, the parameters of the Gaussian distribution μ and Σ can be computed for each of the class $k = 1, 2, 3, ..., k$. And the class priors $P(C_k)$ probabilities can be calculated simply by the fractions from the training dataset for each class.

Once we have the Gaussian parameters and also the priors for each class, then we can estimate the class-conditional densities $P(x|C_k, \mu_k, \Sigma_k)$ for each class k: For each data point x, we first project it from D space to D' then we evaluate $N(x|\mu, \Sigma)$. By Bayes's rule, then we can get the posterior class probabilities $P(C_k|x)$ for each class $k = 1, 2, 3, ..., k$ using

$$p(C_k|x) = p(x|C_k)P(C_k) \tag{3}$$

We then can assign the input data x to the class $k \in K$ with the largest posterior.

Given all the instructions above, we can try on a dataset called MNIST, which has $D = 784$ dimensions. We want to do a projection to $D' = 2$ or $D' = 3$.

Questions:

- Implement the FLD for high-dimension to low-dimension projection with multivariant Gaussian. (Here are some of the implementations of functions that you can use if necessary).
- Split the MNIST dataset into training and testing. And report the accuracy on test set with $D = 2$ and $D' = 3$.
- Make two plots for your testing data with a 2D and 3D plots. You can use different colors on the data points to indicate different classes. See if your results make sense to you or not.

Question 3: Bias Variance Decomposition (40 points)

Recall that the squared error can be decomposed into bias, variance and noise:

$$\underbrace{\mathbb{E}[(h_D(x) - y)^2]}_{\text{Error}} = \underbrace{\mathbb{E}[(h_D(x) - \bar{h}(x))^2]}_{\text{Variance}} + \underbrace{\mathbb{E}[(\bar{h}(x) - \bar{y}(x))^2]}_{\text{Bias}} + \underbrace{\mathbb{E}[(\bar{y}(x) - y(x))^2]}_{\text{Noise}}$$

We will now create a data set for which we can approximately compute this decomposition. The function toydata generates a binary data set with class 1 and 2. Both are sampled from Gaussian distributions:

$$p(\vec{x}|y = 1) \sim \mathcal{N}(0, I) \text{ and } p(\vec{x}|y = 2) \sim \mathcal{N}(\mu_2, I), \tag{4}$$

where $\mu_2 = [2; 2]^\top$ (the global variable OFFSET $=2$ regulates these values: $\mu_2 = [\text{OFFSET} ; \text{OFFSET}]^\top$).

You will need to implement four functions: computeybar, computehbar, computevariance and biasvariancedemo.

(a) Noise (computeybar): First we focus on the noise. For this, you need to compute $\bar{y}(\vec{x})$ in computeybar. With the equations, $p(\vec{x}|y = 1) \sim \mathcal{N}(0, I)$ and $p(\vec{x}|y = 2) \sim \mathcal{N}(\mu_2, I)$, you can compute the probability $p(\vec{x}|y)$. Then use Bayes rule to compute $p(y|\vec{x})$.

Hint: You may want to use the norm probability density function, which you can directly use some package to call this function if you find some. With the help of computeybar you can now compute the "noise" variable within biasvariancedemo. Here is a plot that what your data is supposed to be like in Figure 1:

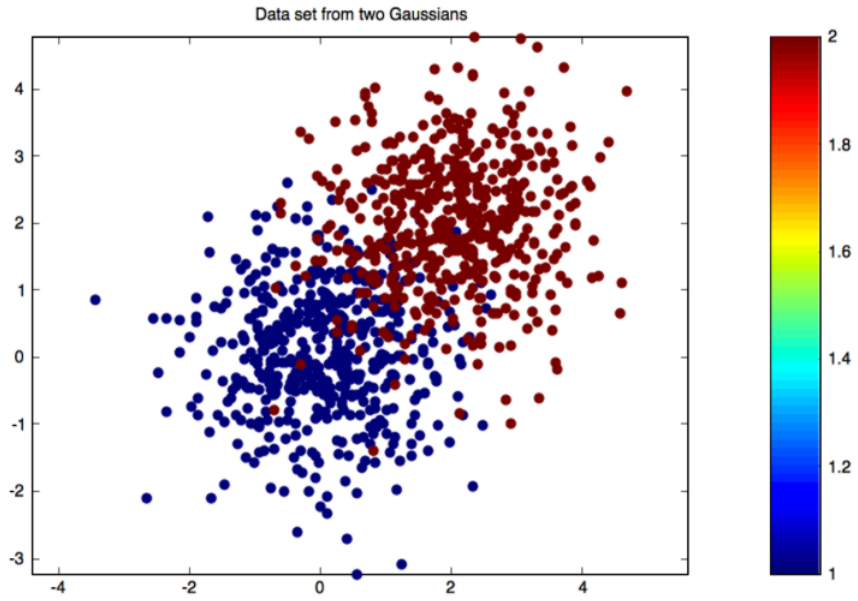

Figure 4: Data Distribution

(b) Bias (computehbar): For the bias, you will need $\bar{h}$. Although we cannot compute the expected value $\bar{h} = \mathbb{E}[h]$, we can approximate it by training many h_D and averaging their predictions. Edit the function computehbar: average over *n_models* different h_D, each trained on a different data set of n inputs drawn from the same distribution. Feel free to call toydata to obtain more data sets.

Hint: You can use ridge regression for h_D. With the help of computehbar you can now compute the "bias" variable within biasvariancedemo.

(c) Variance (computevariance): Finally, to compute the variance, we need to compute the term $\mathbb{E}[(h_D - \bar{h})^2]$. Once again, we can approximate this term by averaging over *n_models* models. Edit the file computevariance.

With the help of computevariance you can now compute the "variance" variable within biasvariancedemo.

(d) Demo (biasvariancedemo): In this function, you need to implement a plotting function that how the error decomposes (roughly) into bias, variance and noise when regularization constant λ increases. You can see the trend if you did everything correctly.

Hint: You can set a training dataset with a size of 500, for a really big dataset you can set the size as 100000. You can try average over 25 models. But all these parameters you can change freely.

The bigger number for the number of models and/or the training dataset, the better your approximation will be for $\mathbb{E}[h]$ and $\mathbb{E}[(h_D - \bar{h})^2]$).

If you get everything correct, you should get some plot like this:

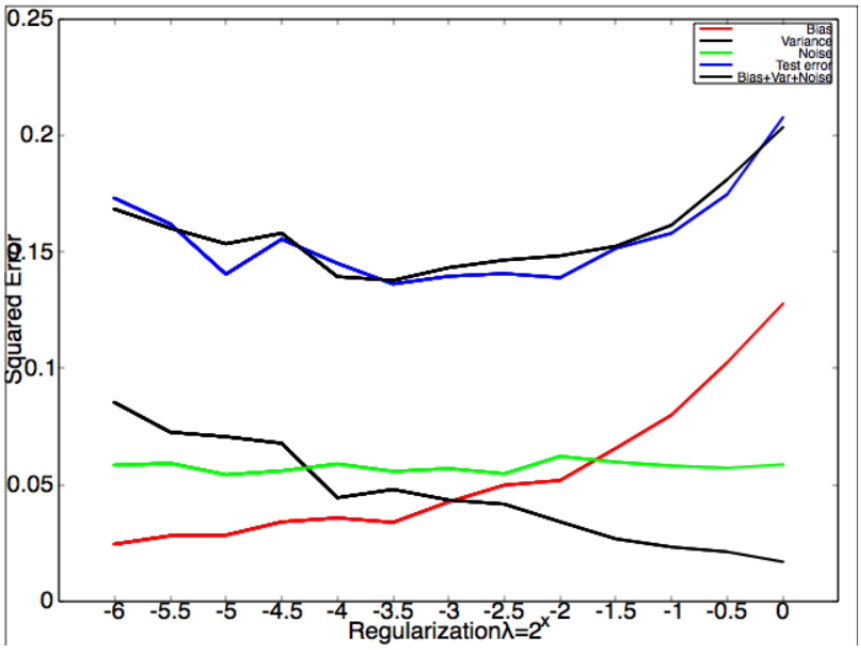

Figure 5: Error decomposition