Assignment #1

SYDE – 675, Winter 2019
Tushar Chopra
tushar.chopra@uwaterloo.ca

---

1. Consider two classes described by the covariance matrices below (assume zero mean)

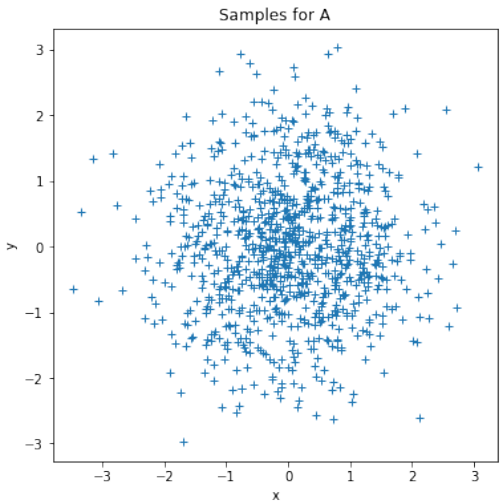
$$a.\sum=[\ [1,0]\ [0,1]\ ]\quad b.\sum=[\ [1,0.9]\ [0.9,1]\ ]$$

a) For each matrix generate 1000 data samples and plot them on separate figures.
b) For each case calculate first standard deviation contour as a function of the mean, eigenvalues, and eigenvectors. Show your calculation (Hint: consider distribution whitening from the tutorial). You may use pre-existing functions for Eigen computation. Plot each contour on the respective plots from part (a).
c) Calculate sample covariance matrices for each class using the data generated in part (a). Do not use a Python/Matlab function for computing the covariance.
d) Compare the given covariance matrix for each class with the corresponding sample covariance matrix generated in (b).

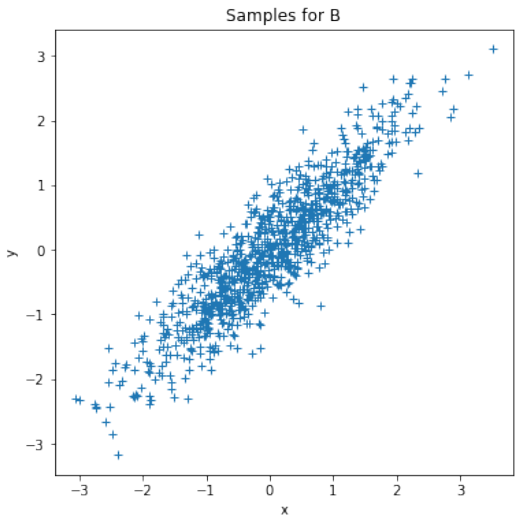
Explanations: -

a)

Results:-



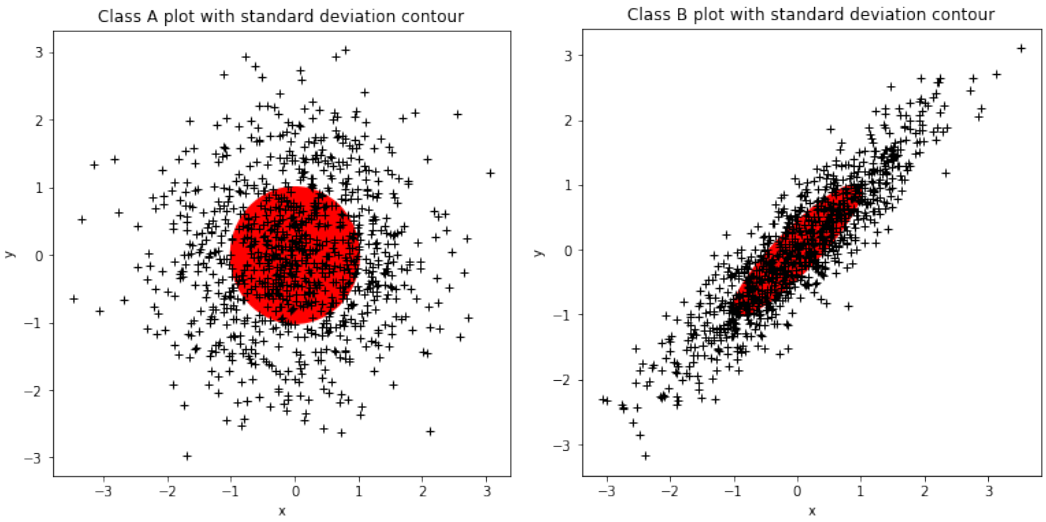
Distribution for class a



Distribution for class b.

➔ As, the covariance and mean of both the classes are given. We used np.random.multivariate_normal function which is used to generate random samples with Gaussian distribution of mean 0 and covariance of a and b.

b) Results:-



➔ Standard deviation contour is plotted above for both the classes (Class A and Class B).First, Eigen Values and Eigen Vectors are calculated using Covariance Matrix. Then to plot the contour, Mean is used as center, Eigen Values are used to calculate standard deviation (which is used for 2 axis) and Eigen Vectors are used to get angle of the ellipse.

c) Results:-

Cov(A):
[[1.01610862 0.04286161]
[0.04286161 1.03927416]]

Cov(B):
[[1.04891803 0.93509886]
[0.93509886 1.01803137]]

➔ Equation for Covariance Matrix:
- Let x is your data
- Move your data to center (X = x – mean(x))
- Covariance of x = $X^T X$ / len(X)

d) Results:

RMSE for Cov A: 0.001369045544623301
RMSE for Cov B: 0.0012954910153811984

➔ The RMSE of original and calculated covariance matrix are given above for both the classes A and B. We can see the error is negligible but it exists. One possible reason could be reconstru cted covariance is generated using random samples and subtracting mean from them will no t have same effect on all elements (Like if we have one extreme point).

➔ One possible way to decrease RMSE is that to reduce step size and generate more points.

2. Consider a 2D problem with 3 classes where each class is described by the following priors, mean vectors, and covariance matrices.

P(C1) = 0.2

μ1 = [3 2] TT

∑1= [[1 −1] [−1 2]]

P(C2) = 0.3

μ2 = [5 4] T

∑2= [[1 −1] [−1 7]]

P(C3) = 0.5

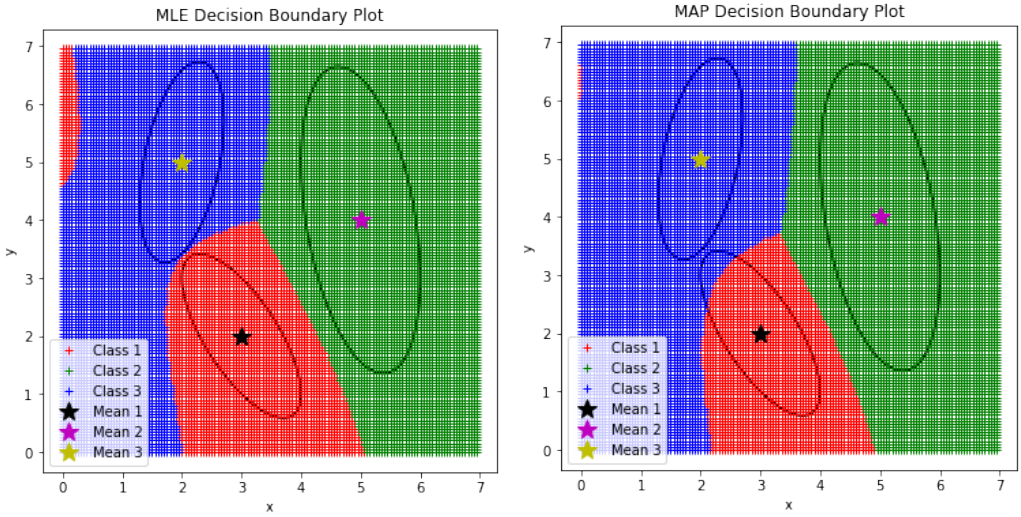
μ3 = [2 5] TT

∑3= [[0.5 0.5] [0.5 3]]

a) Create a program to plot the decision boundaries for a ML and MAP classifier. Plot the means and first standard deviation contours for each class. Discuss the differences between the decision boundaries.
b) Generate a 3000 sample dataset using the prior probabilities of each class. For both the ML and MAP classifiers: classify the generated dataset, calculate a confusion matrix, and calculate the experimental P(ε). Discuss the results.

Explanations: -

a) Results:-



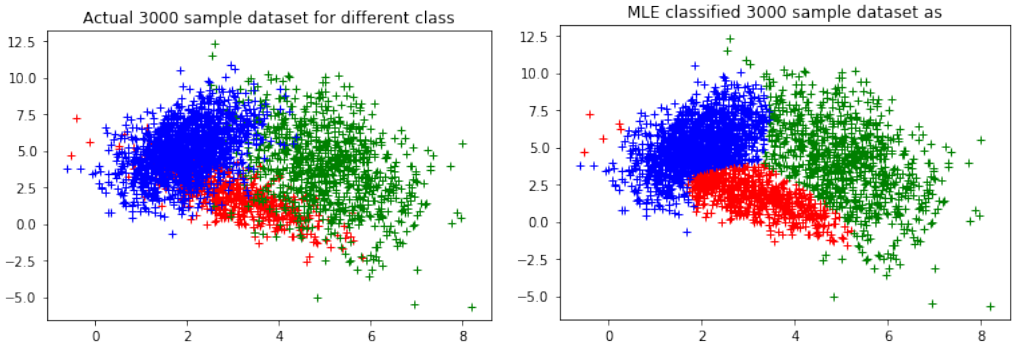
➔ The prior probability, mean and covariance matrix were given for 3 classes (C1, C2, C3). To get the decision boundary we generated numpy mesh-grid, and for every point(x) on mesh-grid, we calculated likelihood (Conditional probability of observation $\boldsymbol{x}$ given class C i.e P(x/Ck)). The equation of likelihood is given below.

$$\text{Likelihood or P(x/Ck)} = \frac{1}{(2*pi)^{l/2}|S|^{1/2}} * \exp(-(0.5)(x-m)^T S^{-1}(x-m))$$

➔ Now, the boundary of MAP classifier is defined as “i”, if P(x|Ci)*P(Ci) > P(x|Cj)*P(Cj) and vise verse, that is, the point with the higher value of P(x|Ck)*P(Ck) is classified in k class. Similarly, the boundary of ML classifier is defined as “i”, if P(x|Ci) > P(x|Cj), that is, the point with the higher value of P(x|Ck) is classified in k class. For all the classes we have given P(Ck) and by using the equation of Likelihood, we get the class of new point(x) using MAP and ML classifiers. Also, Standard deviation contour is plotted in the graph as well. Mean, eigen values and eigen vectors are used for this.

b) Results:-

For MLE

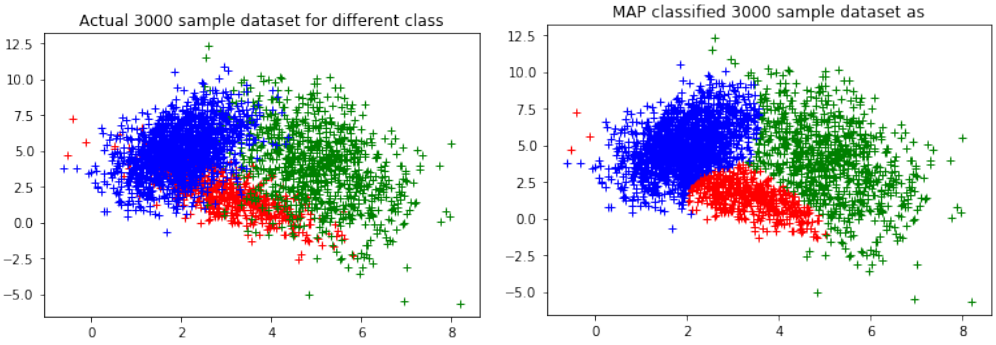


Confusion Matrix

	Class 1	Class 2	Class 3	Accuracy
Class 1	490	24	86	490/600
Class 2	70	800	30	800/900
Class 3	125	38	1337	1337/1500

P(e) = 1 – sum(Accuracy) = 1 - (490/600 + 800/900 + 1337/1500) = 0.124 (approx)

For MAP



Confusion Matrix

	Class 1	Class 2	Class 3	Accuracy
Class 1	419	32	149	419/600
Class 2	58	799	43	799/900
Class 3	45	26	1429	1429/1500

P(e) = 1 – sum(Accuracy) = 1 - (419/600 + 799/900 + 1429/1500) = 0.117 (approx)

- ➔ The confusion matrix describes everything thing behind the scenes, that is, how many samples of class 1 are classified as class 1, class 2 and class 3. Similarly, for class 2 and class 3. It can be used to indicate that which dataset(classes) need to be cleaned before projecting it to any classification algorithm.
- ➔ The accuracy for both the classifiers are shown above. We can see MAP classifier has performed marginally better accuracy over MLE classifier and that's because we are taking prior probabilities into account.
- ➔ Probability of error can be found as 1 – Accuracy of classifier. That is being used above and P(e) for MLE is 0.124 and MAP is 0.117.

3. The MNIST dataset contains a set of images containing the digits 0 to 9. Each image in the data set is a 28x28 image. The data is divided into two sets of images: a training set and a testing set. The MNIST dataset can be downloaded from http://yann.lecun.com/exdb/mnist/. Use only the training set to perform this part.

a) Program PCA that takes X(DxN) and returns Y(dxN) where N is the number of samples, D is the number of input features, and d is the number of features selected by the PCA algorithm. Note that you must compute the PCA computation method by yourself. You may use pre-existing functions for Eigen computation.
b) Propose a suitable d using proportion of variance (POV) =95%.
c) Program PCA reconstruction that takes Y (dxN) and returns X (DxN) (i.e., a reconstructed image). For different values of d= {1, 2, 3, 4, ..., 784} reconstruct all samples and calculate the average mean square error (MSE). Plot MSE (y-axis) versus d (x-axis). Discuss the results.
d) Reconstruct a sample from the class of number '5' and show it as a 'png' image for d= {1, 10, 50, 250, 784}. Discuss the results.
e) For the values of d= {1, 2, 3, 4, ..., 784} plot eigenvalues (y-axis) versus d (x-axis). Discuss the results.

3. Explanations: -

a) Results:-

Original Data Shape: (784, 60000)
Lower Dimension Data Shape: (60000, 10)

- ➔ Algorithm for PCA:-
  1. $\boldsymbol{\mu}$ = sample mean of X, Σ = sample covariance of
  2. X_hat = subtract sample mean $\boldsymbol{\mu}$ from each column sample (X_hat has zero mean)
  3. Find eigenvectors and eigenvalues of Σ
  4. $W =$ Using d $(d < D)$ eigenvectors with largest eigenvalues form the mapping function as a $D \times d$ matrix, each column corresponds to an eigenvector
  5. The transformed samples will be $Y = W^{\mathrm{T}} X\_\mathrm{hat}$

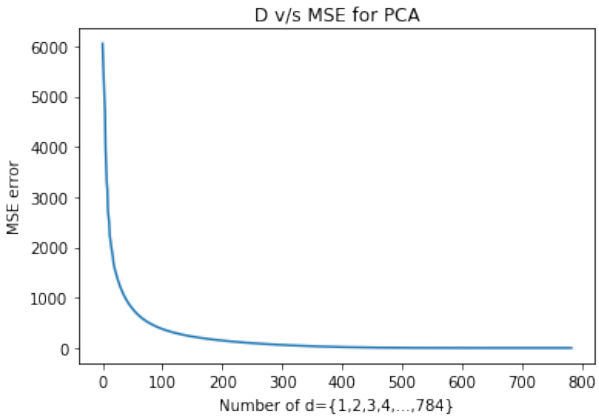
- ➔ The MNIST data is used from the link given in the question(http://yann.lecun.com/exdb/mnist/). In code, "pca_from_no_dimentions()" is used to calculate PCA which gives data in lower number of dimensions. It takes 2 arguments, one is dataset ("data") and other is the required number of dimentions ("dimentions"). After execution, it returns lower dimention data, eigen vectors of data which have largest eigen values, eigen values and mean

b) Results:-

Number of d for 95% POV: 154

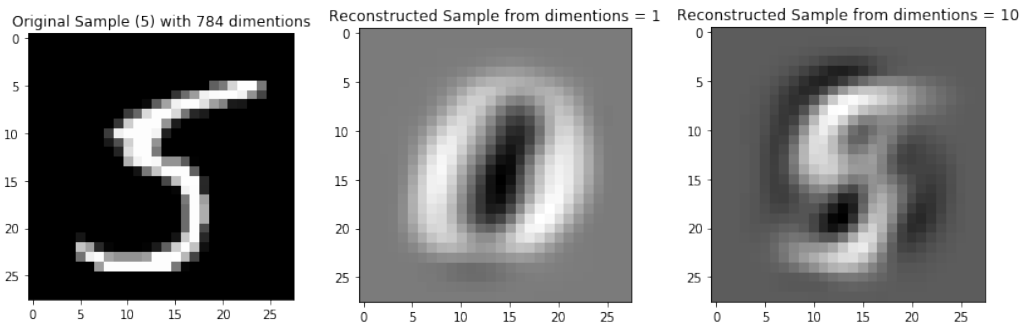
➔ For calculating suitable d for 95% POV, we calculated eigen values of the dataset, sort them from largest to smallest and sum them up until we get 95% eigen values. Next step, is to use corresponding eigen values to get lower dimension data. For MNIST dataset of 60,000 images, we have 784 dimensions per image, after applying PCA we got 154 dimensions which has 95% of POV.

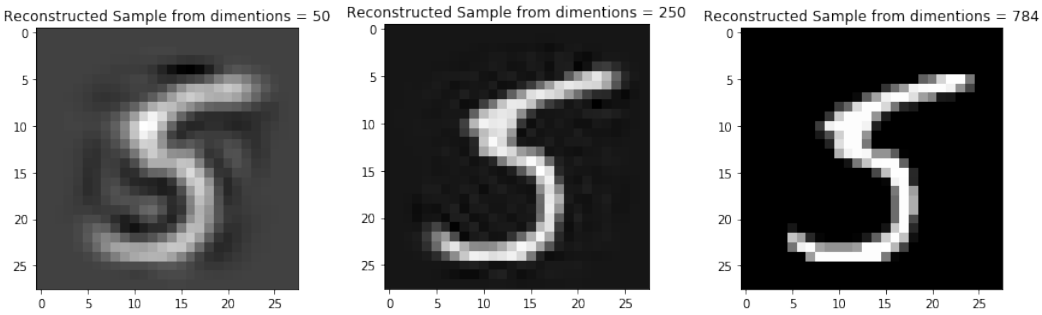
c) Results:-



➔ We applied PCA for every value of d = {1 to 784} values and then reconstructed the data. We can see from the above plot that as the number of dimensions increases the MSE errors decreases. This is expected behaviour as the reconstruction from lower dimensions will have greater error as in lower dimensions we lost most of the variance, but as we increase the number of dimensions in PCA it will preserve more variance and hence lower MSE error. We, can see after 154 dimensions the MSE error is much lower, POV greater than 95 percent after d equals 154. After 300 it is almost near to 0, as oppose to 6000 to 4000 for first 10 dimensions.

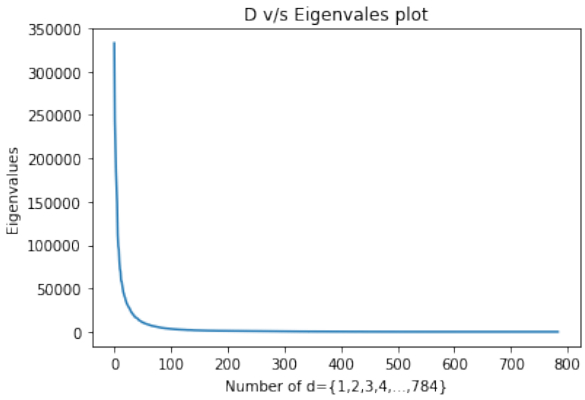
d) Results:-





➔ The output confirms that the image generated after reconstruction from the lower dimension (lower value of d) has more noise than the image generated after reconstruction from the higher dimension (higher value of d). We can see that image reconstructed from 784 dimensions is identical to original "5" image, but as we reduce the dimensions from 784 to 250,50,10,0, the noise is started to increase. And at d=1, we can see visually that the image is not even classified as "5" instead it looks like "0".

e) Results:-



➔ The graph implies that roughly 80 eigen values out of 784 are contributing in the representation of the information or they preserve more variance than remaining dimensions, which are more significant ones to classify the data. We can see the plot till 80 dimensions is quite high, and the importance of remaining dimension is quite less for classification. As the dimension increases its significance decreases, and approaches to 0 after 120 dimensions.

4. Once again, consider the MNIST dataset. Use the training set as your training data and the test set as your test data.

a) Classify the test data using a kNN classifier. Report the accuracy for k = {1,3,5,11}. Justify and compare the reported accuracies for the different values of k. Do not use kNN implemented function in Python/Matlab and implement it by yourself.
b) Apply PCA to MNIST to create a new dataset MNIST-d. Classify the test samples in MNIST-d using a kNN classifier. For each d = {5, 50, 100, 500} use k = {1,3,5,11}. Calculate sand display the classification accuracy for each of the 16 combinations of d and k in a table. Discuss the results.
c) Compare the reported accuracies in part (a) and part (b)

Explanations: -

a) Results:-

k=1 : 96.91
k=3 : 97.04
k=5 : 96.86
k=11 : 96.64

➔ The accuracy of different values of "k" is given above. We can see that the results are be st at k=3. In KNN we have to find best value of "k" for better accuracy. This process is call ed parameter tuning. For different values of k, we have different results, its depend upon the neighbours of each test sample. If k is small we can have more noise, that is, KNN ca n pick just next data sample for your test sample, which may be wrong. On the other han d if our k is big we need more processing time. Based on our results we say that k=3 is be st value of k for our MNIST dataset.

b) Results:-

Executing for d = 5
For d = 5 k=1 : 71.76 k=3 : 75.0 k=5 : 76.81 k=11 : 78.42

Executing for d = 50
For d = 50 k=1 : 97.44 k=3 : 97.61 k=5 : 97.54 k=11 : 97.38

Executing for d = 100
For d = 100 k=1 : 97.11 k=3 : 97.26 k=5 : 97.33000000000001 k=11 : 97.02

Executing for d = 500
For d = 500 k=1 : 96.93 k=3 : 97.0 k=5 : 96.89999999999999 k=11 : 96.67

➔ We used different lower dimension data for KNN that is for d = {5,50,100,500}. As the d= 5 its accuracy is the lowest because it doesn't have sufficient variance of data to make pr edictions. On the other hand its performance when d=50 is better and almost same for r emaining value of d = {100,500} as well. It seems like at d equal 50, we have sufficient va riance of data to make good predictions. But When we increase the value of d to 100 or 500 its accuracy decreases marginally.

c) Results:-

Table of accuracies for different D and K values

	D=5	**D=50**	**D=100**	**D=500**	**D=784**
K=1	71.76	75.0	76.81	78.42	96.91
K=3	97.44	97.61	97.54	97.38	97.04

K=5	97.11	97.26	97.33(approx)	97.02	96.86
K=11	96.93	97.0	96.89(approx)	96.67	96.64

- ➔ The table is given for 4a and 4b. For different values of k and d, we found that the accuracy is best at d=50 and k=5. The possible reason for this is, at d=50 we have sufficient information/variance of the data that can contribute in decision making. Also among all 4 values of k, we found the optimal value at k=3 (Parameter tuning among 4 k values).
- ➔ Another possible reason for best value of d=50 could be that remaining dimensions are not contributing to predictions but only adding noise. As when we explore an image, we can see most of the pixels are zero, that is having no information at all, one the pixels in the centre are significant to participate in classification.
- ➔ We know that depending upon the value of the d, the variance varies. That is, lower "d" have less variance than the higher value of "d". So, at d=5 and k=1, we got worst results. Because, neither we have sufficient dimensions to get maximum spread of data, nor the best value of k. This may be observed as under fitting.