Homework 2

Due date is **16th December 2022, 23:59!**

kamard@itu.edu.tr
korkmazmer@itu.edu.tr

Homework Policy

- Use comments whenever necessary to explain your code.
- You will use Python as the programming language. However, you are NOT allowed to use built-in functions from Python libraries for Bayes classifier model building. Following are libraries that are NOT allowed:
  - scipy, Bio, Theano, Tensorflow, Keras, PyTorch and any other similar machine learning modules.

  Following are libraries that are allowed:
  - pandas, numpy, math, random, matplotlib, csv, etc... Basically anything that is not related to machine learning. When you are in doubt, simply ask.
- IMPORTANT: This is an individual assignment! You are expected to act according to Student Code of Conduct, which forbids all ways of cheating and plagiarism. It is okay to discuss the homework with others, but it is strictly forbidden to use all or parts of code from other students' codes or online sources and let others do all or part of your homework.
- Only electronic submissions through Ninova will be accepted. You only need to submit your Jupyter Notebook file. Any comments and discussions should be included in the same file.
- If you have any questions, you can contact us. Do NOT hesitate to send an e-mail if you are confused.

For easy reference, a mathematical notation summary table is also attached in the last page of this homework.

Jupyter Notebook Installation

You need to have Python and Pip installed in your computer to install Jupyter Notebook.

Windows

On command prompt (cmd.exe with admin mode):

```
C:\**path**> python -m pip install jupyter
```

After changing your current folder to the folder which you want to work on (see ’cd’ command):

```
C:\**your_working_folder**> jupyter notebook
```

Then it will be launched on your default browser

Ubuntu/Linux/Unix/Mac

On Terminal:

```
$ pip install notebook
```

Then launch with:

```
$ jupyter notebook
```

For more information: https://jupyter.org/install

Homework 2

Problem Description

You will carry out all the tasks below using **Ipython Notebook**. Simply add all your work to the provided template file **HW1_template.ipynb** using jupyter notebook.

You are given 2 different datasets with each having a train and test set separately. You are going to use these datasets in Parts **A and B**. **Note that** the first column in each dataset is indexing, you are not going to use them during the training, remember to remove them before training.

Number of samples	**Dataset 1**	**Dataset 2**
Training set	2000	1600
Test set	500	400
Total	**2500**	**2000**

Part A (15 Points)

Examine the two training datasets 1 and 2:

1. Import your training set (but only the training set for now). You can use pandas' DataFrame for this task (up to you), which is very convenient as it is easy to read .csv files using pandas and can extract data from it as numpy array. You can find a cheatsheet for it here.

2. For each class, calculate and visualize the covariance matrices of datasets 1 and 2 and interpret them (via comment outs) with your own words. Example figure can be seen in Figure 3 **(6 points)**

3. For each dataset, plot two overlaid transparent histograms for each feature (red for class 0 and blue for class 1) and interpret (via comment outs in Jupyter Notebook) them as well. This will allow you to examine the feature distribution within each class and compare them across classes. Example plot can be seen in Figure 1

   - In total you should generate 4 plots (each plot with 2 overlaid histograms): 2 datasets $\times$ 2 features each. **(6 points)**

4. Plot each training dataset (feature 1 on x-axis and feature 2 on y-axis). Example plots can be seen in Figure 2. **(3 points)**

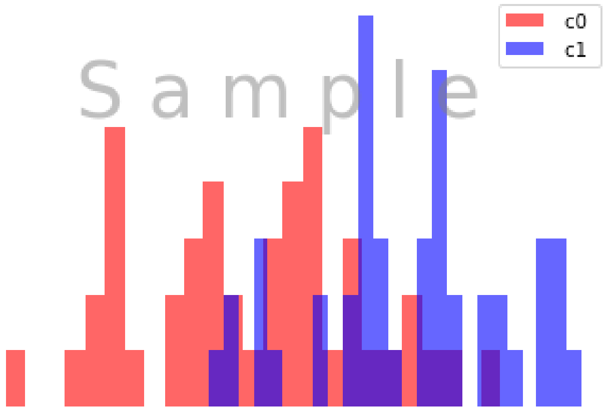

Figure 1: Example plot of distribution of feature 1.

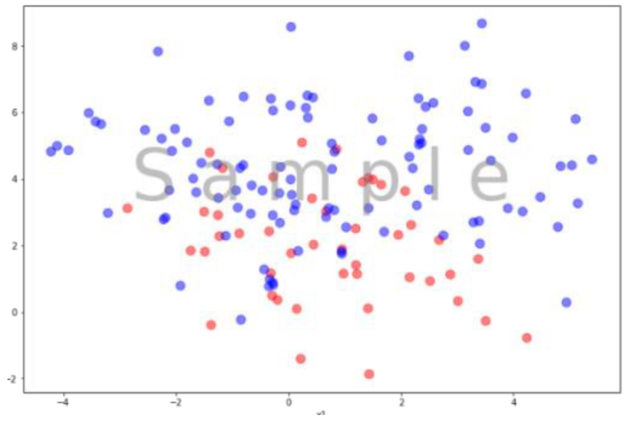

Figure 2: Example scatter plot of classes.

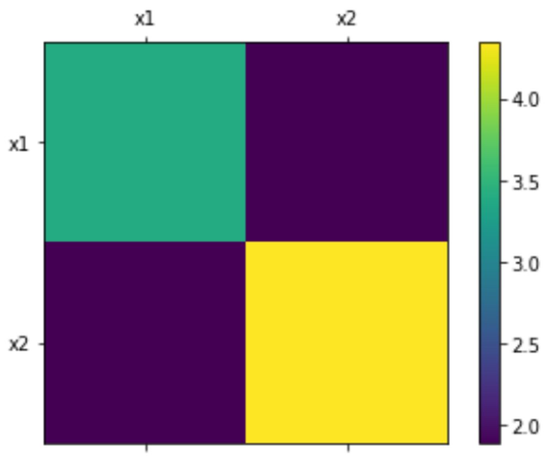

Figure 3: Example covariance matrix.

Part B: Evaluation of Parametric classifier using random single dataset split into train and test sets (60 Points)

You will use **Datasets 1 and 2** in this part. Assume that each class is generated from a multivariate Gaussian distribution.

1. Estimate the mean vectors for each class using the training data and consider the covariance matrices that you have calculated in Part A.: (just **print()** them)

$\mathbf{c}_1$ **(class 1)**	$\mathbf{c}_2$ **(class 2)**
Mean Vector ($\mu_1 = [??]$)	Mean Vector ($\mu_2 = [??]$)
Covariance Matrix $\Sigma_1 = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}$	Covariance Matrix $\Sigma_2 = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}$

2. Compute the linear discriminant function $g_i(x)$ for each class regarding the following cases: **(35 points)**

   - Case 1 **(25 points):** Equal covariances across the classes ($\Sigma_1 = \Sigma_2$).
   - Case 2 **(20 points):** General case where the discriminant function $g_i(x)$ is quadratic.

   You can omit a tolerance of 10^-3 while checking the equality of two values, i.e. $a = b$ if $absolute(a - b) <= 10^-3$.

   You may use the following code block for this part (or feel free to use your own code, it is up to you):

   Algorithm 1: Example code block

```
    def trainBayes(trainingSamples, trainingLabels):
        if "covariance␣matrices␣are␣equal":
            g_iX = ...
        else: #general case
            g_iX = ...
        return g_iX
```

3. **For both datasets 1 and 2,** test your Bayes classifier **(15 points):**

   Apply the model to the test set after you import it and save the results in a vector.

   Note that, you should pretend like you don't know the labels (classes) of the test set while predicting.

Compare the predicted labels by Bayes classifier you have obtained with the actual (ground truth) labels of the test set and calculate the classification error rate as follows:

$$\mathbf{e} = \frac{\#\text{ mis Classification}}{\#\text{ testSamples}} \times 100$$

NOTE: You are going to use two different datasets, therefore classification errors you observe may differ highly. Your aim is to implement the Bayes classifier correctly, not to minimize the error!

Part C: Implementation of PCA (25 Points)

Principal Component Analysis (PCA) is a powerful (and quite popular) tool used to study datasets consisting of observations which have higher feature dimensions. It compresses the data while preserving the maximum amount of information. This can speed up algorithms, help create lower complexity models with better results, and make the data visualization process more intuitive. PCA is easy to compute—it is based on eigenvector analysis. Uncorrelated variables (principal components) are extracted from the data and the variability of the data is captured on the axes constructed from these principal components.

In this part of the assignment, you are asked to **naively** implement PCA.

1. **Implementation of PCA**

   The first step is to implement the PCA algorithm. Write a function that conforms to the following declaration:

   ```
   def PCA(X, k=None, varRetained = 0.95, show = False):
   ```

   - **Inputs:**
     - **X:** Input dataset **(dataframe)**.
     - **k:** number of minumum features that retains the given variance. If k is given, don't compute k **(integer)**.
     - **varRetained:**The minimum percentage (%) of variance that should be retained after PCA is applied **(float)**.
     - **Show:** Whether or not the plot of Variance vs. #Features will be shown **(boolean)**.
   - **Outputs:**
     - **Projection Matrix:** Constructed from the top principal components **(dataframe)**.
     - **Reduced Feature Set:** Transformation of the input dataset into a new nxk dimensional dataset **(dataframe)**.

2. **Exploration with PCA** Now it's time to test your implementation of PCA on data. You will test using the Scikit-learn Iris dataset. This dataset contains 3 classes of the iris plant and each class contains 50 samples. The main property of the Iris classes is that only one class is linearly seperable from the others. The remaining two classes are not linearly seperable. This can be observed in Figure 4.

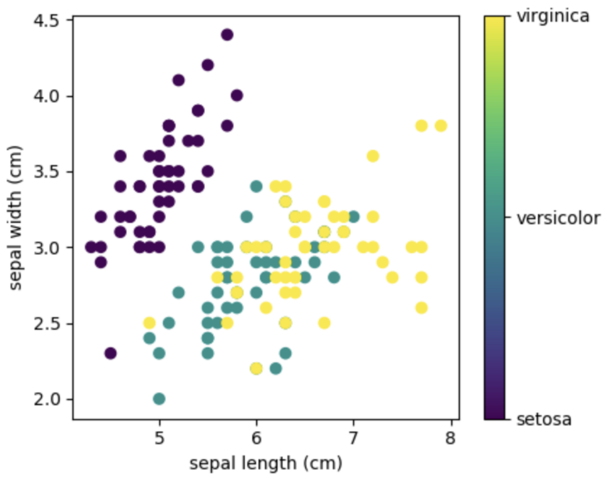

Figure 4: A scatter plot of the Iris dataset before dimensionality reduction with PCA.

After running PCA, the transformed dataset can be plotted again. This time, it is quite apparent that all three classes are now linearly separable (Figure 5).

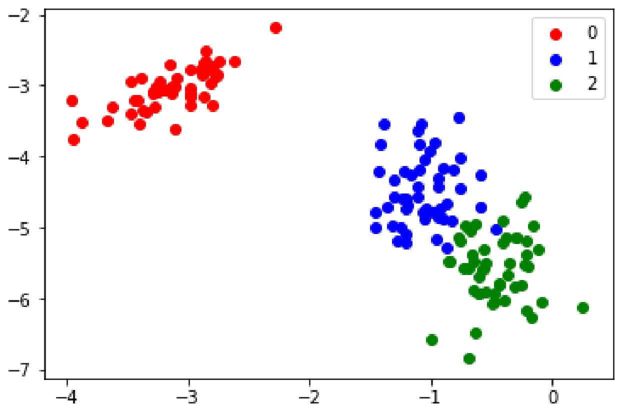

Figure 5: A scatter plot of the Iris dataset after PCA is applied.

Figure 6 illustrates the number of features incorporated into PCA versus the percentage of variance retained from the data. Notice how as more features are incorporated, more of the variance is retained.

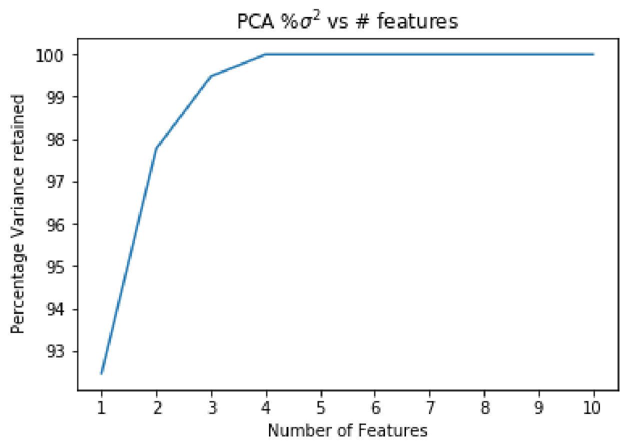

Figure 6: % of Variance Retained vs. # of Features After PCA.

For this part, run the PCA algorithm on **both** the **Iris** and the **Breast Cancer** datasets from scikit-learn. Reproduce the following graphs:

- Scatter plot of the **original data**
- Scatter plot **after** PCA is applied
- PCA **(%** σ **vs # of Features)** Line Chart

3. **Evaluation**

   Answer the following questions:

   - Evaluate and comment on **each** of your resulting plots from the previous part. What did PCA change? How do the number of incorporated features impact the percentage of variance retained? What does this mean for the size of the dataset and performance of learning algorithms?
   - Explain the difference between feature selection and feature extraction. Which category does PCA fall under?
   - Can you think of any scenario where PCA would fail?
   - Why is it important to standardize before applying PCA?

Note: Include any references you used.

Table 2: *Major mathematical notations used in lecture 3.*

Mathematical notation	Definition
$\mathcal{D}$	dataset
n	number of samples in a dataset $\mathcal{D}$
d	number of features
$\mathbf{x} \in \mathbb{R}^{d\times 1}$	feature vector or data point (sample)
$\mathbf{x}^{sample}_{feature}$	–
$\mathbf{x}^{i} \in \mathbb{R}^{d\times 1}$	i^{th} sample in the population
$\mathbf{x}^{i}_{j} \in \mathbb{R}$	j^{th} feature of i^{th} sample in the population
$\mathbf{\Sigma} \in \mathbb{R}^{d\times d}$	covariance matrix of data population $\{\mathbf{x}^i\}_{i=1}^{n}$
$\lvert\mathbf{A}\rvert \in \mathbb{R}$	determinant of matrix A
$p(x) = \frac{1}{\sqrt{2\pi}\sigma} exp(\frac{-1}{2}\frac{\|\|x-\mu\|\|_2^2}{\sigma^2})$	probability density function of a variable $x \in \mathbb{R}$
$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}\lvert\mathbf{\Sigma}\rvert^{1/2}} exp[\frac{-1}{2}(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)]$	probability density function of a multidimensional variable $\mathbf{x} \in \mathbb{R}^{d\times 1}$
$\mu \in \mathbb{R}^{d\times 1}$	sample mean $\mu = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}^i$
$\|\|\mathbf{x}-\mu\|\|_{\mathbf{\Sigma}^{-1}} = (\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu) \in \mathbb{R}$	Mahalanobis distance between $\mathbf{x}$ and μ
$\mathbf{I}_{d\times d} \in \mathbb{R}^{d\times d}$	identify matrix of size d ×d
$\|\|\mathbf{x}-\mu\|\|_{\mathbf{I}_{d\times d}} = (\mathbf{x}-\mu)^T(\mathbf{x}-\mu) \in \mathbb{R}$	Euclidean distance between $\mathbf{x}$ and μ
	also noted as L_2 norm $\|\| \cdot \|\|_2$
$g_i(\mathbf{x}) = \mathbf{w}_i^T\mathbf{x} + \mathbf{w}_{i0}$	discriminant Bayes function for class i when $\mathbf{\Sigma}_i = \sigma_i^2\mathbf{I}$
	general case: when $\sigma_i^2 \neq \sigma_j^2$ for classes i and j (i.e., different means $\mu_i \neq \mu_j$
	but constant variance for all data features in each class)
	special case: when $\sigma_i^2 = \sigma_j^2$ for classes i and j (i.e., different means $\mu_i \neq \mu_j$
	but constant variances across all classes)
	(i.e., lines connecting means of different classes are perpendicular to decision boundaries)
	if $ln(p(c_i)) = ln(p(c_j))$, $g_i(\mathbf{x}) = -\|\|\mathbf{x}-\mu_i\|\|_2^2$
$g_i(\mathbf{x}) = \mathbf{w}_i^T\mathbf{x} + \mathbf{w}_{i0}$	discriminant Bayes function for class i when $\mathbf{\Sigma}_i = \mathbf{\Sigma}_j = \mathbf{\Sigma}$
	(i.e., constant data feature covariance $\mathbf{\Sigma}$ across classes)
	(i.e., lines connecting means of different classes are not perpendicular to decision boundaries)
	if $ln(p(c_i)) = ln(p(c_j))$, $g_i(\mathbf{x}) = -\frac{1}{2}\|\|\mathbf{x}-\mu_i\|\|_{\mathbf{\Sigma}^{-1}}^2$
$g_i(\mathbf{x}) = \mathbf{x}^T\mathbf{W}\mathbf{x} + \mathbf{w}_i^T\mathbf{x} + \mathbf{w}_{i0}$	quadratic discriminant function (decision boundaries are nonlinear)