STATISTICAL LEARNING AND NEURAL NETWORKS, A.A. 2022/2023

COMPUTER LAB 1 - k-NN classifier

Duration: 6 hours

Exercise 1 - Synthetic dataset

In this exercise, you will employ a synthetic dataset (file Lab1_Ex_1_Synthtetic.hdf5), containing labelled training data and test data for two classes. For each example the first two columns represent the features, while the last column represents the label.

Task: your task is to implement a k-NN classifier, which calculates the probability that a given test example belongs to each class, and outputs a class label as the class with the highest probability. You will evaluate the classifier performance computing the average classification accuracy (i.e. the fraction of test examples that have been classified correctly in respect to the full test set).

In particular, you should perform the following:

- Train a k-NN classifier for different values of k.
- Compare accuracy on the training set and the test set. Calculating accuracy of the training set means that you will have to classify each sample in the training set as if it were a test sample; one expects that classification of training samples will perform well, and this may also be used to validate your implementation. Accuracy is defined as the ratio between the number of test samples that are correctly classified, and the total number of test samples. Create a graph using the matplotlib library showing the evolution of the accuracy for different values of k over the test set. Create a second graph to show the evolution of the accuracy for different values of k over the train set and compare the two.
- Identifying overfitting and underfitting in the obtained results.

Note that, for this computer lab, you do not need to employ a validation set.

Other indications:

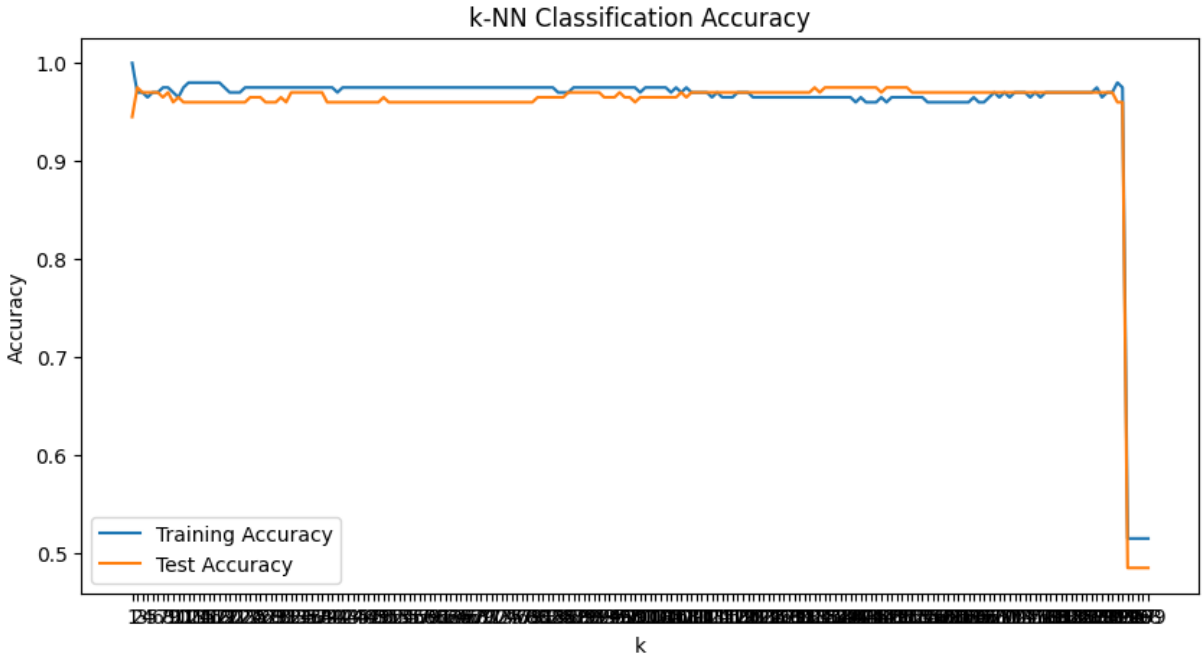
- The student is required to implement the k-NN algorithm from scratch. Only the numpy library is allowed, while other libraries such as scikit_learn are forbidden.

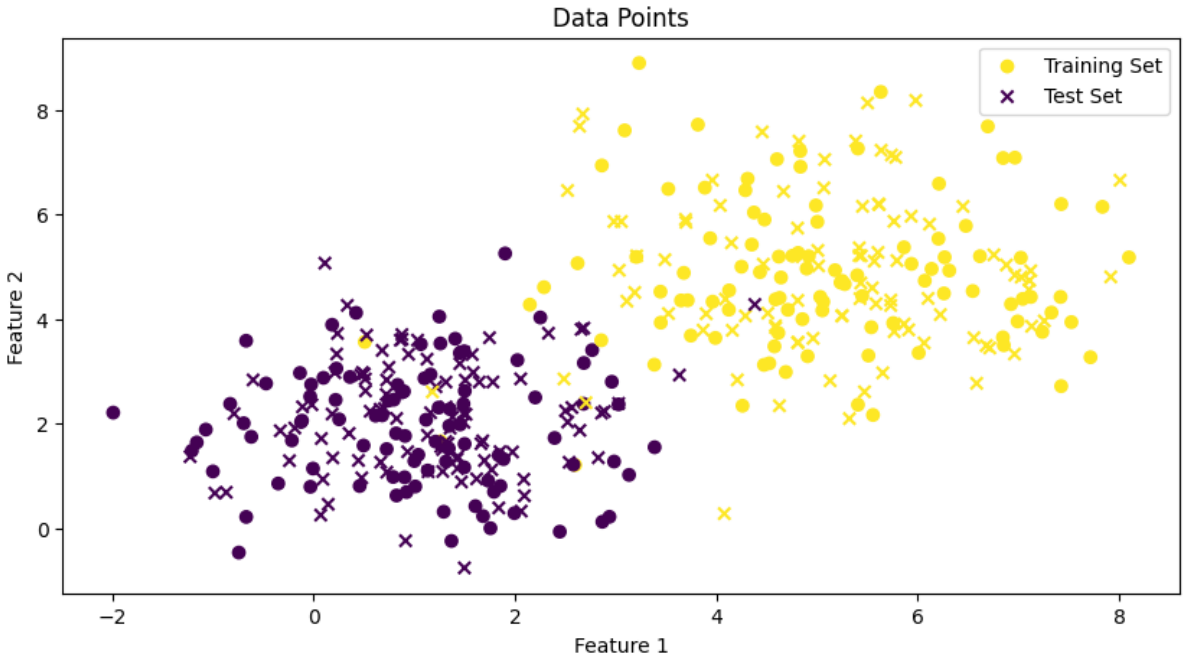
Excercise 1 ----------

KNN

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import h5py
Dataset1 = h5py.File('Lab1_Ex_1_Synthtetic.hdf5', 'r')
Data = np.array(Dataset1.get('Dataset'))

Train_Set = Data[:200, :]
Test_Set = Data[200:, :]
#Implement a function to compute the Euclidean distance between two vectors, and one to implement the k-NN algorithm by:
#   - Taking a sample
#   - Computing all the distances between the sample element and the elements of the training set
#   - sort the the training set based on the distances to the element (the use of functions like np.argsort is allowed)
#   - select the top k elements in terms of distance
#   - evaluate to which class the majority of these k elements belongs to (e.g., it is possible to use the function np.unique


def euclidean_distance(vec1, vec2):
    return np.sqrt(np.sum((vec1 - vec2) ** 2))

def k_nn_classifier(sample, myData, k):
    distances = []
    for data_point in myData:
        distances.append((euclidean_distance(sample[:-1], data_point[:-1]), data_point[-1]))
    distances.sort(key=lambda x: x[0])
    neighbors = distances[:k]
    # print(neighbors)
    classes, counts = np.unique([neighbor[-1] for neighbor in neighbors], return_counts=True)
    # print(classes, counts)
    return classes[np.argmax(counts)]

# Loop over different values of k and compute accuracies
train_accuracies = []
test_accuracies = []
ks = range(1, 200)  # Assuming you want to test k from 1 to 20
for k in ks:
    correct_train = 0
    correct_test = 0

    # Evaluate on training set
    for sample in Train_Set:
        predicted_class = k_nn_classifier(sample, Train_Set, k)
        if predicted_class == sample[-1]:
            correct_train += 1

    # Evaluate on test set
    for sample in Test_Set:
        predicted_class = k_nn_classifier(sample, Train_Set, k)
        if predicted_class == sample[-1]:
```

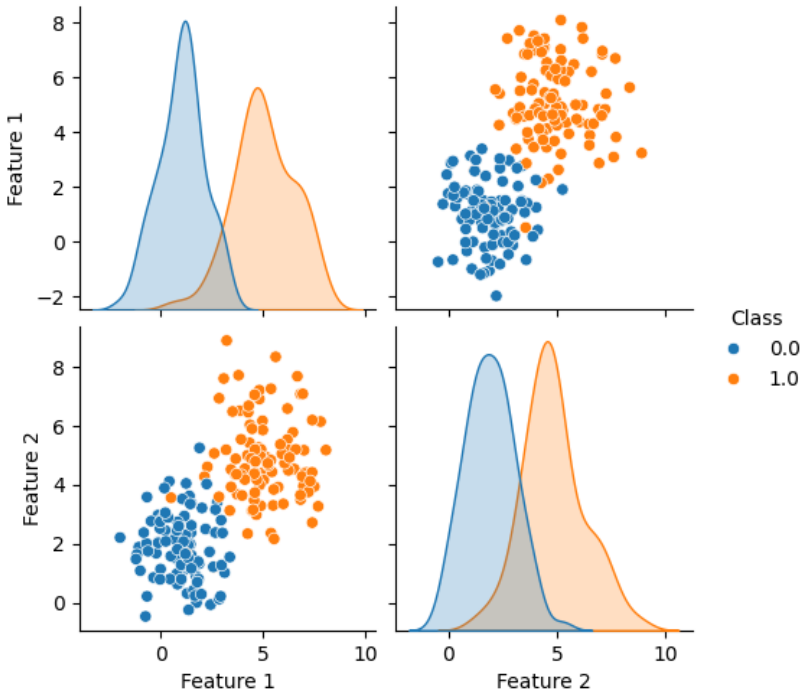
```
            correct_test += 1

    train_accuracies.append(correct_train / len(Train_Set))
    test_accuracies.append(correct_test / len(Test_Set))

# Plot accuracies
plt.figure(figsize=(10, 5))
plt.plot(ks, train_accuracies, label='Training Accuracy')
plt.plot(ks, test_accuracies, label='Test Accuracy')
plt.xticks(ks)
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.legend()
plt.title('k-NN Classification Accuracy')
plt.show()

#plot datapoints and predictions
plt.figure(figsize=(10, 5))
plt.scatter(Train_Set[:, 0], Train_Set[:, 1], c=Train_Set[:, 2], label='Training Set')
plt.scatter(Test_Set[:, 0], Test_Set[:, 1], c=Test_Set[:, 2], label='Test Set', marker='x')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Data Points')
plt.legend()
plt.show()
```





In [ ]:
```
import seaborn as sns
import pandas as pd
sns.pairplot(pd.DataFrame(Train_Set, columns=['Feature 1', 'Feature 2', 'Class']), hue='Class')
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x10f737f10>
```



Comment On Ex1

we implemented an Euclidian Distance calculator which is used to calculated the distance between each single point (sample point with respect to train datapoints), this distance will be used in KNN classifier, distances are sorted and k-first elements are taken and then ArgMax is called over the ksamples which will return the most frequent class among the samples.

this ArgMax result is the prediction of label for the new sample, we assign the label as y_pred. then to calculate the accuracy we compare true y_preds to actual values of labels then we count them and divide by the total predictions(is equal to test size), this will give us the accuray of our prediction method.

for k=1 -> we will observe a bad prediction due to overfit on the training data since we have few training data, as we increase the number of k the accuracy of training set will decrease while the accuracy of test set increases, this is due to variablity in models brahviour, the suitable value of k needs some fine tuning, if k becomes too large then the the model is too simplistic and fails to capture local variations in the data, and too generalized, this might cause underfitting over the training set and poor results for the predictions

Excercise 2 ----------

Wine Testing using Linear Regression

Exercise 2 - Wine dataset

Part 1

In this exercise, a real problem will be examined. The dataset used in this exercise was derived from wine quality dataset from the work "*Modeling wine preferences by data mining from physicochemical properties*" by P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis.

For each element of the dataset 11 features are provided, representing different wine characteristics, such as density, pH and alcholic content, and the final column consists of a quality evaluation on a scale from 1 to 10. More information can be found at https://archive.ics.uci.edu/ml/datasets/wine+quality.

A subset of the dataset containing 400 elements is provided. Create a training set and a test set of 200 samples each. The objective is to:

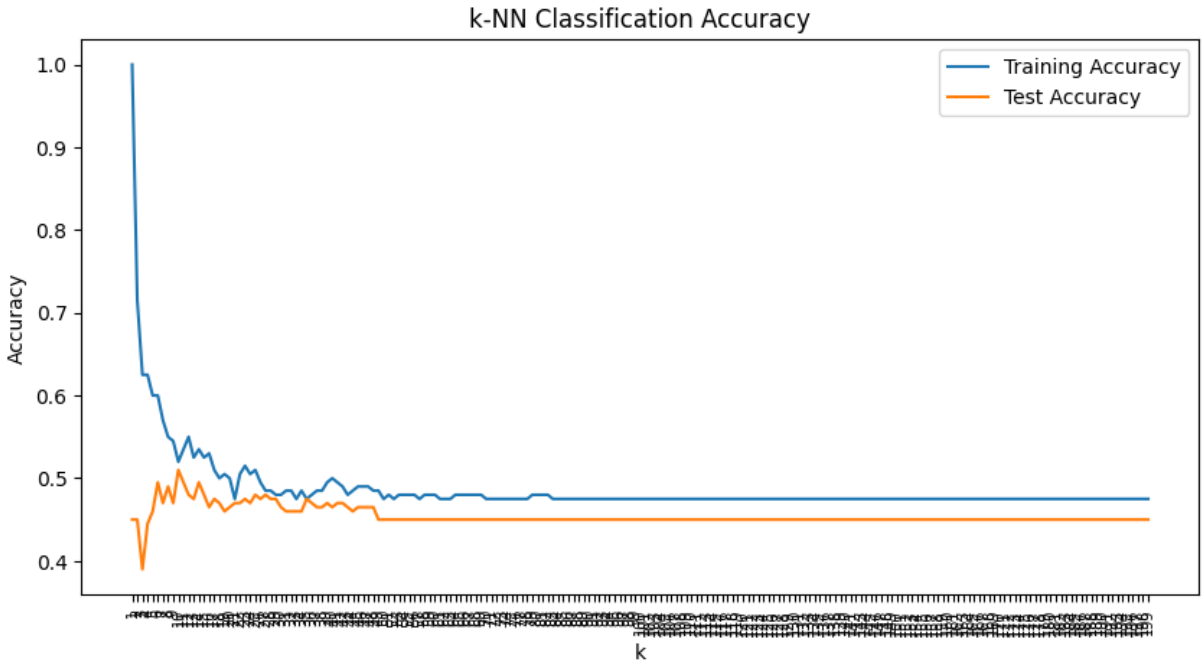
- Predict the wine quality over the test set using the k-NN algorithm and evaluating the prediction accuracy for different values of k. Create a graph using the matplotlib library showing the evolution of the accuracy for different values of k over the test set.
- Identifying overfitting and underfitting in the obtained results.

Part 2

The prediction of the wine quality could also be framed as a regression. Estimate the accuracy and the Mean Square Error achieved using linear resgression. For this task is possible to use the library sklearn and the function linear_model.LinearRegression()

In [ ]:
```
#Part 1

Dataset2 = h5py.File('Lab1_Ex_2_wine.hdf5')

Data = np.array(Dataset2.get('Dataset'))
np.random.shuffle(Data)

Train_Set = Data[:200,:]
Test_Set = Data[200:,:]

#To be completed by the student

#Part 1

# Loop over different values of k and compute accuracies
train_accuracies = []
```

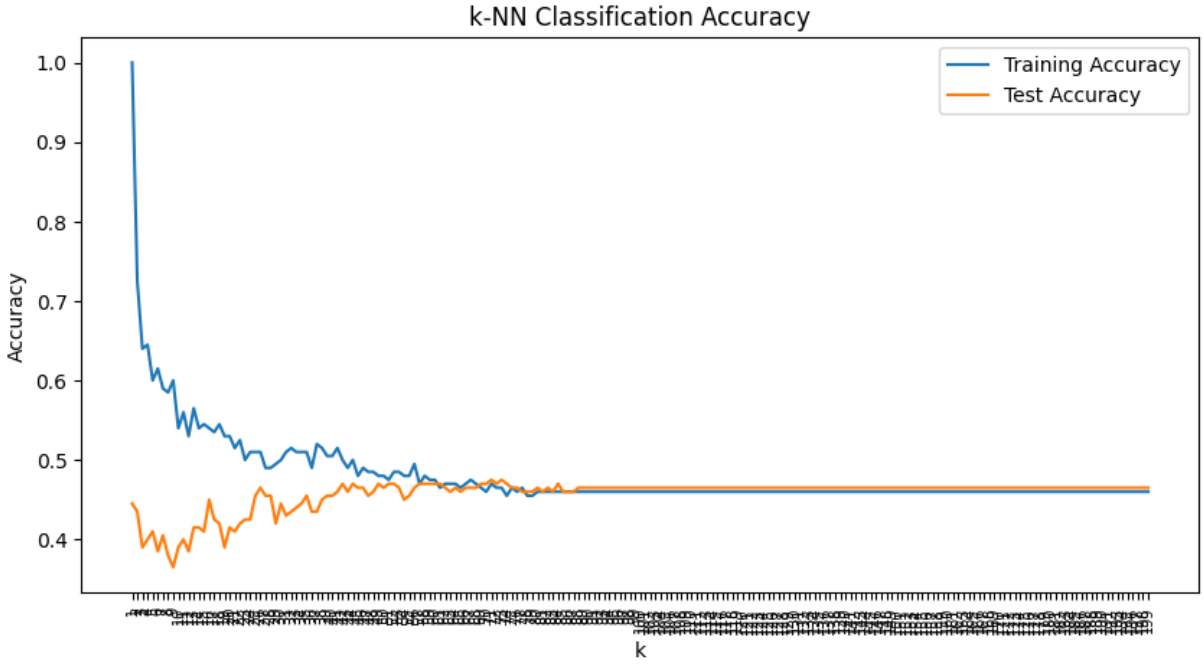
```
test_accuracies = []
ks = range(1, 200)  # Assuming you want to test k from 1 to 30
ypred = []
for k in ks:
    correct_train = 0
    correct_test = 0

    # Evaluate on training set
    for sample in Train_Set:
        predicted_class = k_nn_classifier(sample, Train_Set, k)
        if predicted_class == sample[-1]:
            correct_train += 1

    # Evaluate on test set
    for sample in Test_Set:
        predicted_class = k_nn_classifier(sample, Train_Set, k)
        ypred.append(predicted_class)
        if predicted_class == sample[-1]:
            correct_test += 1

    train_accuracies.append(correct_train / len(Train_Set))
    test_accuracies.append(correct_test / len(Test_Set))

# Plot accuracies
plt.figure(figsize=(10, 5))
plt.plot(ks, train_accuracies, label='Training Accuracy')
plt.plot(ks, test_accuracies, label='Test Accuracy')
plt.xticks(ks, rotation=90, fontsize=7)
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.legend()
plt.title('k-NN Classification Accuracy')
plt.show()
#To be completed by the student
```



In [ ]:

```
#Part 1

Dataset2 = h5py.File('Lab1_Ex_2_wine.hdf5')

Data = np.array(Dataset2.get('Dataset'))
np.random.shuffle(Data)

Train_Set = Data[:200,:]
Test_Set = Data[200:,:]

#normalize the data
Train_Set_Normal = (Train_Set - Train_Set.mean(axis=0)) / Train_Set.std(axis=0)
# Test_Set = (Test_Set - Test_Set.mean(axis=0)) / Test_Set.std(axis=0)

#To be completed by the student

#Part 1

# Loop over different values of k and compute accuracies
train_accuracies = []
test_accuracies = []
ks = range(1, 200)  # Assuming you want to test k from 1 to 30
ypred = []
for k in ks:
    correct_train = 0
    correct_test = 0

    # Evaluate on training set
    for sample in Train_Set:
        predicted_class = k_nn_classifier(sample, Train_Set, k)
```

```
        if predicted_class == sample[-1]:
            correct_train += 1

    # Evaluate on test set
    for sample in Test_Set:
        predicted_class = k_nn_classifier(sample, Train_Set, k)
        ypred.append(predicted_class)
        if predicted_class == sample[-1]:
            correct_test += 1

    train_accuracies.append(correct_train / len(Train_Set))
    test_accuracies.append(correct_test / len(Test_Set))

# Plot accuracies
plt.figure(figsize=(10, 5))
plt.plot(ks, train_accuracies, label='Training Accuracy')
plt.plot(ks, test_accuracies, label='Test Accuracy')
plt.xticks(ks, rotation=90, fontsize=7)
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.legend()
plt.title('k-NN Classification Accuracy')
plt.show()
#To be completed by the student
```



In [ ]:

```
import numpy as np
Dataset2 = h5py.File('Lab1_Ex_2_wine.hdf5')

Data = np.array(Dataset2.get('Dataset'))
np.random.shuffle(Data)

Train_Set = Data[:200,:]
Test_Set = Data[200:,:]
#implementing linear regression from scratch for the given dataset
#The dataset is loaded and divided into training and test sets
#The linear regression model is implemented and tested on the test set
# BEGIN: FILEPATH: /Users/graybook/Documents/Projects/Polito/statistical/Statistical-Labs/Stat Labs/Computer LAB 1/Lab1.ipynb

# Define the linear regression model
class LinearRegression:
    def __init__(self):
        self.coefficients = None

    def fit(self, X, y):
        # Add a column of ones to X for the intercept term
        X = np.concatenate((np.ones((X.shape[0], 1)), X), axis=1)

        # Calculate the coefficients using the normal equation
        self.coefficients = np.linalg.inv(X.T @ X) @ X.T @ y

    def predict(self, X):
        # Add a column of ones to X for the intercept term
        X = np.concatenate((np.ones((X.shape[0], 1)), X), axis=1)

        # Calculate the predicted values
        y_pred = X @ self.coefficients

        return y_pred

# Load the dataset
Dataset2 = h5py.File('Lab1_Ex_2_wine.hdf5')
Data = np.array(Dataset2.get('Dataset'))
np.random.shuffle(Data)

Train_Set = Data[:200, :]
```

```
Test_Set = Data[200:, :]

# Split the features and labels
X_train = Train_Set[:, :-1]
y_train = Train_Set[:, -1]
X_test = Test_Set[:, :-1]
y_test = Test_Set[:, -1]

# Create an instance of the linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Predict the labels for the test data
y_pred = model.predict(X_test)

# Calculate the mean squared error
mse = np.mean((y_pred - y_test) ** 2)

#Accuracy of the model
acc2 = np.sum(np.abs(y_pred - y_test) < 0.5) / len(y_test)

# Print the mean squared error
print("Mean Squared Error:", mse)
print("Accuracy2:", acc2)
```

```
Mean Squared Error: 0.6006491087409916
Accuracy2: 0.485
```

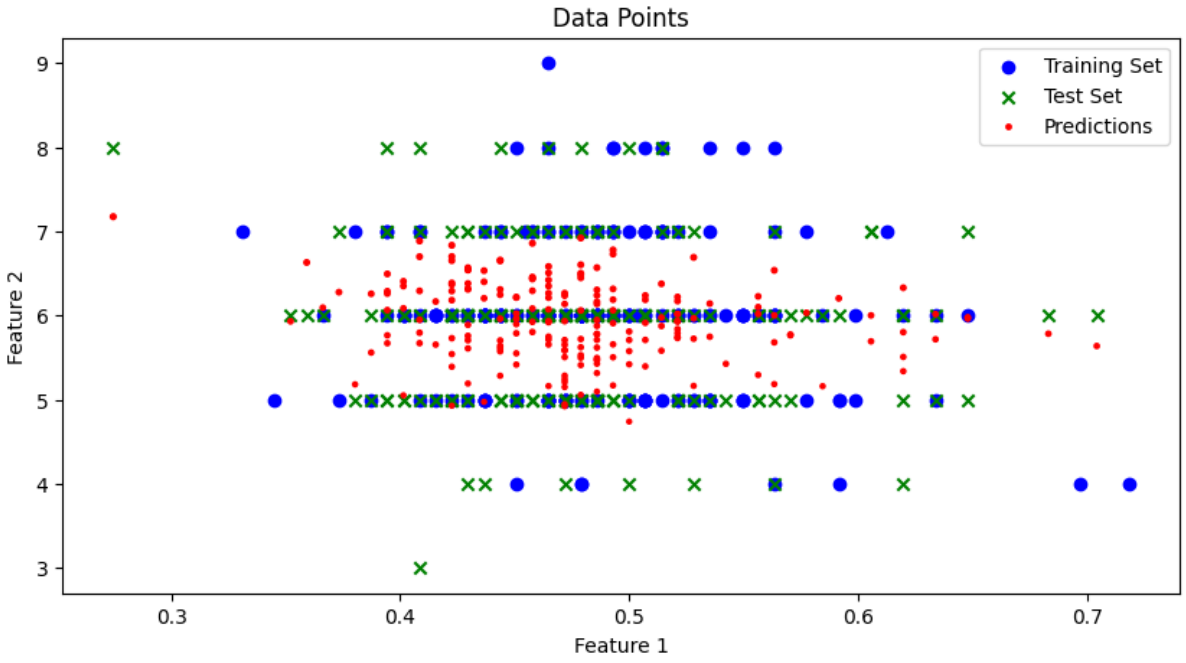
In [ ]:
```
print(Train_Set.shape)
print(np.unique(Train_Set[:,-1], return_counts=True))
```

```
(200, 12)
(array([4., 5., 6., 7., 8., 9.]), array([ 7, 51, 95, 36, 10,  1]))
```

In [ ]:
```
print("Coefficients:", model.coefficients)
```

```
Coefficients: [ 3.16650071e+02  3.13836109e+00 -1.64780640e+00 -5.59508512e-01
  9.71614354e+00 -1.19393342e+00  1.39694150e-01  5.59350998e-01
 -3.32450804e+02  6.66977645e+00  4.47571706e-01 -5.22735955e-01]
```

In [ ]:
```
#plot test and train data points and predictions
plt.figure(figsize=(10, 5))
plt.scatter(X_train[:, 0], y_train, label='Training Set', marker='o', color='blue')
plt.scatter(X_test[:, 0], y_test, label='Test Set', marker='x', color='green')
plt.scatter(X_test[:, 0], y_pred, y_pred, label='Predictions', color='red')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Data Points')
plt.legend()
plt.show()
```



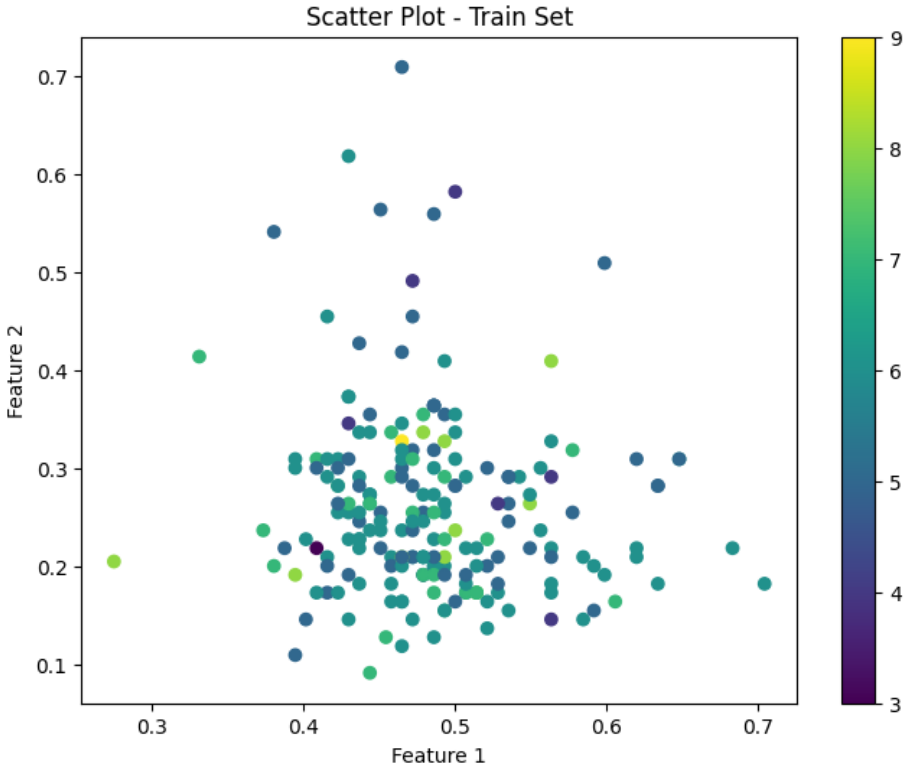
Comment On Ex2

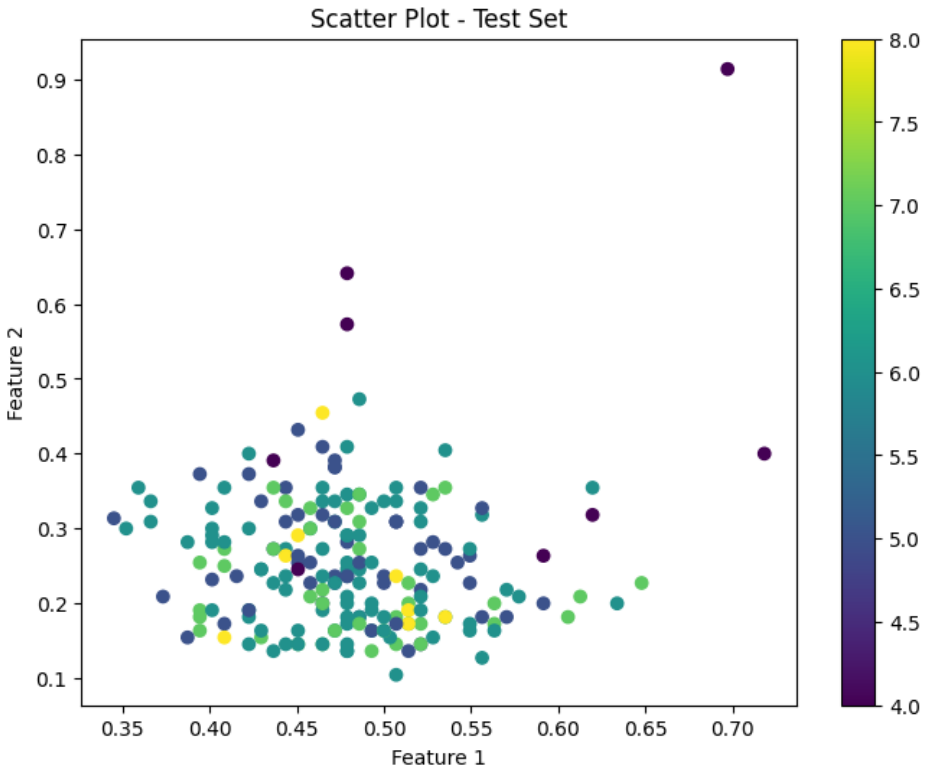
- since we have different value ranges for different features we need to normalize data to avoid giving higher weights to higher ranges and vice-versa

- the value of parameter k indicated the number of neighbours, if we say that test accuracy is lower than 50 while there is a high accuracy for train set we would say we have overfitting which is memorizing the training data and failing to predict the true values for test data, in this case approximately for values under 26 we see the pattern but it is at it's highest at the lower values of k. underfitting is happening for the values over k=50 and system fails to understand the actual patterns and classes in the data due to over generalization.

- Regression is used to predict a value in continuous series of data, we can use regression for the classification problems if we ignore the floating part of the result and treat the result as a class label (as discrete values). inn our case the accuracy was poor as so maybe it is not the best choise to use regression for classification.

In [ ]:
```
import matplotlib.pyplot as plt

# Scatter plot for train set
plt.figure(figsize=(8, 6))
plt.scatter(Train_Set[:, 0], Train_Set[:, 1], c=Train_Set[:, -1])
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Scatter Plot - Train Set')
plt.colorbar()
plt.show()

# Scatter plot for test set
plt.figure(figsize=(8, 6))
plt.scatter(Test_Set[:, 0], Test_Set[:, 1], c=Test_Set[:, -1])
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Scatter Plot - Test Set')
plt.colorbar()
plt.show()
```





In [ ]:
```
#Part 2
from sklearn import linear_model
clf = linear_model.LinearRegression()

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

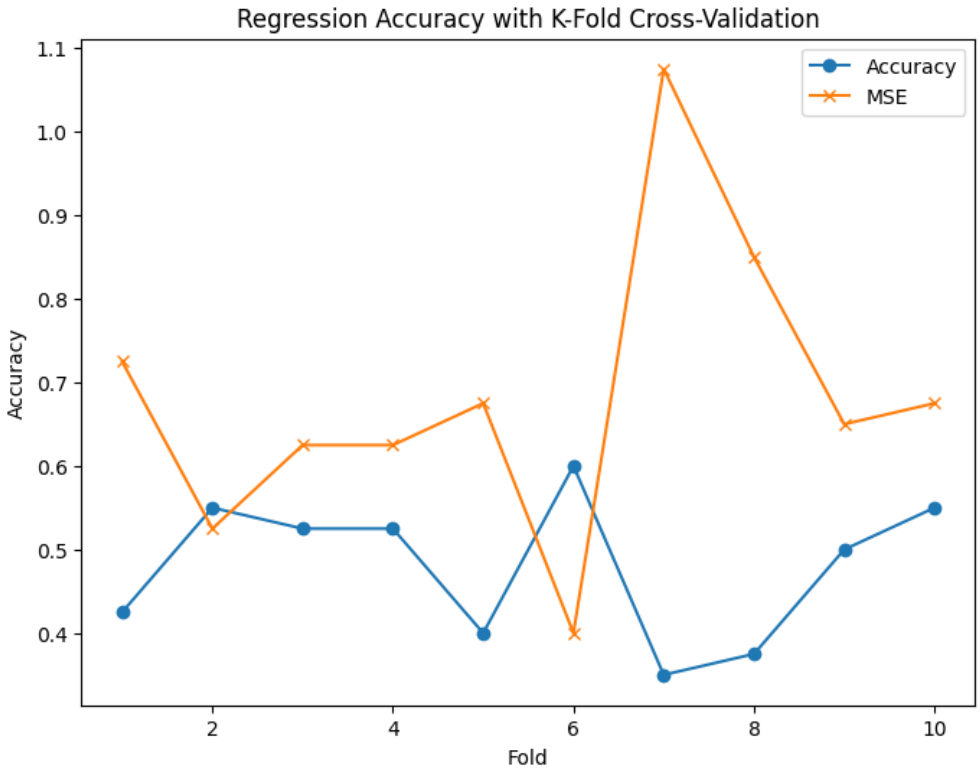
```
from sklearn.metrics import mean_squared_error

Dataset2 = h5py.File('Lab1_Ex_2_wine.hdf5')

Data = np.array(Dataset2.get('Dataset'))
np.random.shuffle(Data)

# Split dataset

X_train = Data[:300, :-1]
y_train = Data[:300, -1]
X_train = (X_train - X_train.mean(axis=0)) / X_train.std(axis=0)


X_test = Data[100:, :-1]
y_test = Data[100:, -1]
X_test = (X_test - X_test.mean(axis=0)) / X_test.std(axis=0)

print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
# Train linear regression model
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Make predictions
y_pred = regressor.predict(X_test)
y_pred = np.round(y_pred)

# Evaluate model on training set
x_pred = regressor.predict(X_train)
x_pred = np.round(x_pred)

# # Evaluate model
def myAccuracy(y_true, y_pred):
    accuracy = 0
    for i in range(len(y_pred)):
        if y_pred[i] == y_true[i]:
            accuracy += 1
    accuracy /= len(y_true)
    return accuracy

test_acc = myAccuracy(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
print(f'Test Accuracy: {test_acc * 100:.2f}%')
print(f'Test MSE: {mse}')
# Evaluate model on training set
train_acc = myAccuracy(y_train, x_pred)
mse = mean_squared_error(y_train, x_pred)
print(f'Train Accuracy: {train_acc * 100:.2f}%')
print(f'Train MSE: {mse}')
```

```
(300, 11) (300,) (300, 11) (300,)
Test Accuracy: 49.67%
Test MSE: 0.65
Train Accuracy: 49.00%
Train MSE: 0.6566666666666666
```

In [ ]:

```
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

Dataset2 = h5py.File('Lab1_Ex_2_wine.hdf5')

Data = np.array(Dataset2.get('Dataset'))
np.random.shuffle(Data)
# Split dataset
X = Data[:, :-1]
y = Data[:, -1]
X = (X - X.mean(axis=0)) / X.std(axis=0)

# Perform k-fold cross-validation
k = 10
kf = KFold(n_splits=k, shuffle=True)

accuracies = []
mse_values = []
train_accuracies = []
for train_index, test_index in kf.split(X):
  X_train, X_test = X[train_index], X[test_index]
  y_train, y_test = y[train_index], y[test_index]

  # Train linear regression model
  regressor = LinearRegression()
  regressor.fit(X_train, y_train)

  # Make predictions on training set
  y_pred_train = regressor.predict(X_train)
  y_pred_train = np.round(y_pred_train)

  # Make predictions
  y_pred = regressor.predict(X_test)
  y_pred = np.round(y_pred)

  # Evaluate model
  accuracy = np.sum(y_pred == y_test) / len(y_test)
  accuracies.append(accuracy)
```

```
  train_acc = np.sum(y_pred_train == y_train) / len(y_train)
  train_accuracies.append(train_acc)

  mse = mean_squared_error(y_test, y_pred)
  mse_values.append(mse)

# Plot accuracies
plt.figure(figsize=(8, 6))
plt.plot(range(1, k+1), accuracies, marker='o')
plt.plot(range(1, k+1), mse_values, marker='x')
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.title('Regression Accuracy with K-Fold Cross-Validation')
plt.legend(['Accuracy', 'MSE'])
plt.show()

for i in range(10):
    print(f'Fold {i+1}: ACC:{accuracies[i] * 100:.2f}% - MSE:{mse_values[i]}, | Train ACC:{train_accuracies[i] * 100:.2f}%')
```




```
Fold 1: ACC:42.50% - MSE:0.725, | Train ACC:50.56%
Fold 2: ACC:55.00% - MSE:0.525, | Train ACC:49.44%
Fold 3: ACC:52.50% - MSE:0.625, | Train ACC:48.89%
Fold 4: ACC:52.50% - MSE:0.625, | Train ACC:50.28%
Fold 5: ACC:40.00% - MSE:0.675, | Train ACC:51.11%
Fold 6: ACC:60.00% - MSE:0.4, | Train ACC:48.06%
Fold 7: ACC:35.00% - MSE:1.075, | Train ACC:51.67%
Fold 8: ACC:37.50% - MSE:0.85, | Train ACC:50.83%
Fold 9: ACC:50.00% - MSE:0.65, | Train ACC:50.00%
Fold 10: ACC:55.00% - MSE:0.675, | Train ACC:48.33%
```

Student's comments to exercise 2

Add comments to the results of Exercise 2 here (may use LateX for formulas if needed).

Excercise 3 ----------

Speech Signals

Exercise 3: Phoneme Dataset

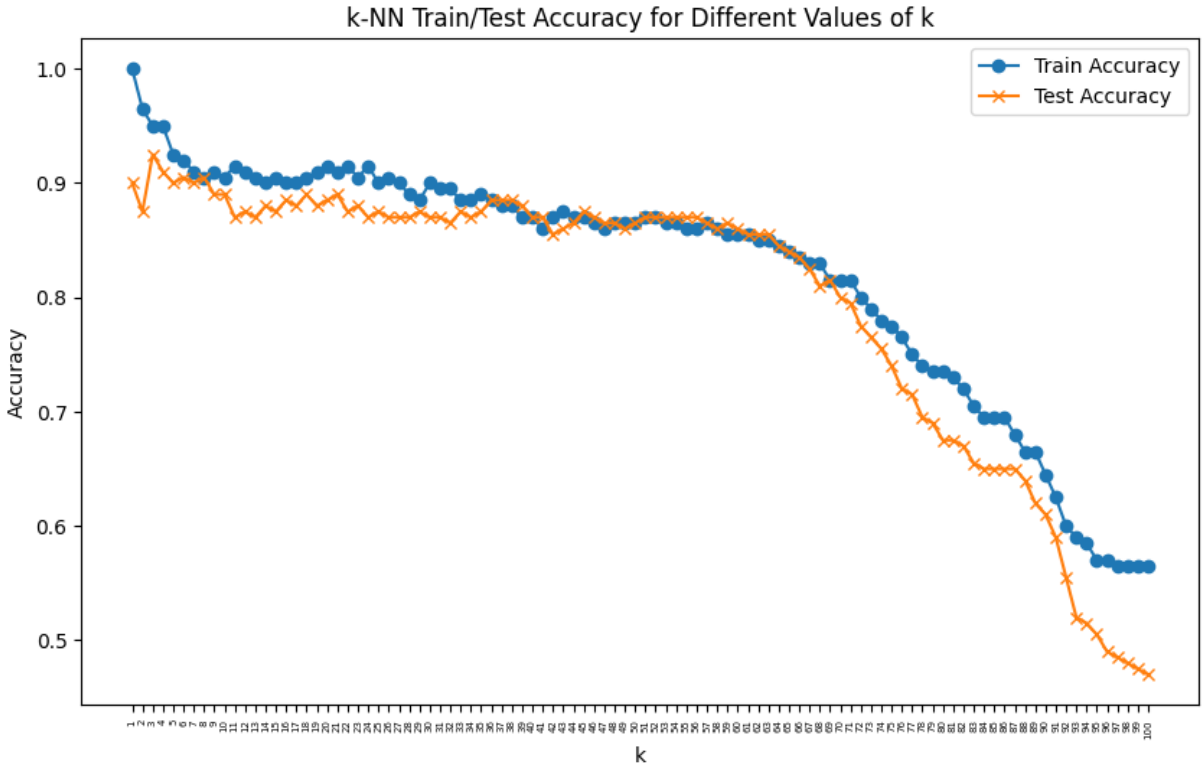
In this exercise the Phoneme dataset is examined https://catalog.ldc.upenn.edu/LDC93s1. Each line represents 256 samples gathered at a 16 kHz of different speech signals. The objective is to classify wether the sound emitted is a "sh", "iy", "dcl", "aa", "ao" phoneme.

Again, a subset of the dataset containing 400 elements is provided. Create a training set and a test set of 200 samples each.

- Classify the samples which compose the test set using the k-NN algorithm and evaluate the prediction accuracy for different values of k. Create a graph using the matplotlib library showing the evolution of the accuracy for different values of k over the test set.
- Identifying overfitting and underfitting in the obtained results.

In [ ]:
```
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt


Dataset3 = h5py.File('Lab1_Ex_3.hdf5')
```

```
Data = np.array(Dataset3.get('Dataset'))
np.random.shuffle(Data)

# Assume phoneme_dataset is already loaded with proper format
# phoneme_dataset = ... # You should load your dataset here

# Split dataset into features and labels
Train_Set = Data[:200, :]
Test_Set = Data[200:, :]

# Test different values of k
train_accuracies = []
test_accuracies = []
ks = range(1, 101)  # Assuming you want to test k from 1 to 30
for k in ks:
    correct_train = 0
    correct_test = 0

    # Evaluate on training set
    for sample in Train_Set:
        predicted_class = k_nn_classifier(sample, Train_Set, k)
        if predicted_class == sample[-1]:
            correct_train += 1

    # Evaluate on test set
    for sample in Test_Set:
        predicted_class = k_nn_classifier(sample, Train_Set, k)
        if predicted_class == sample[-1]:
            correct_test += 1

    train_accuracies.append(correct_train / len(Train_Set))
    test_accuracies.append(correct_test / len(Test_Set))


# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(ks, train_accuracies, label='Train Accuracy', marker='o')
plt.plot(ks, test_accuracies, label='Test Accuracy', marker='x')
plt.title('k-NN Train/Test Accuracy for Different Values of k')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.legend()
plt.xticks(ks, rotation=90, fontsize=5)
plt.show()
```

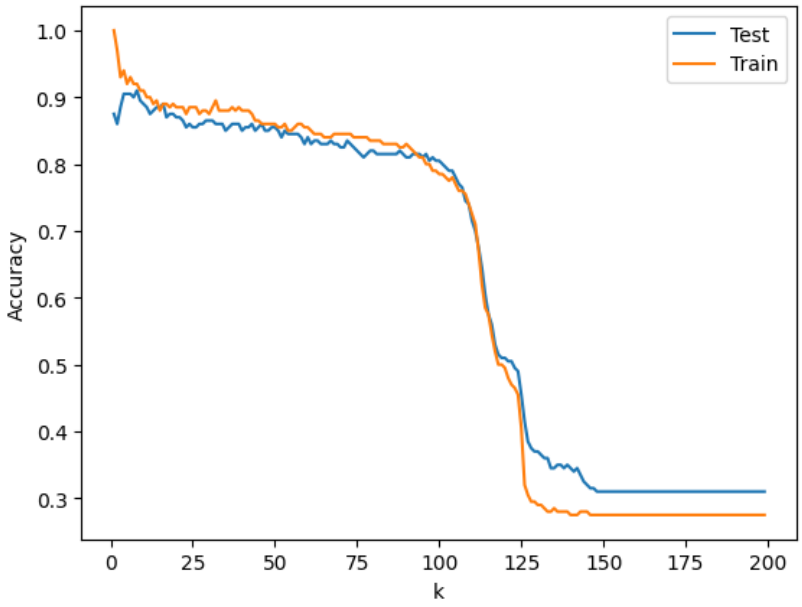


In [ ]:

```
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt


Dataset3 = h5py.File('Lab1_Ex_3.hdf5')

Data = np.array(Dataset3.get('Dataset'))
np.random.shuffle(Data)

# Assume phoneme_dataset is already loaded with proper format
# phoneme_dataset = ... # You should load your dataset here

# Split dataset into features and labels
Train_Set = Data[:200, :]
Test_Set = Data[200:, :]
```

```
XTrain = Train_Set[:, :-1]
yTrain = Train_Set[:, -1]
XTest = Test_Set[:, :-1]
yTest = Test_Set[:, -1]

#normalization
XTrain = (XTrain - XTrain.mean(axis=0)) / XTrain.std(axis=0)
XTest = (XTest - XTest.mean(axis=0)) / XTest.std(axis=0)

#implement knn from scratch
def knn(XTrain, yTrain, XTest, k):
    yPred = np.zeros(XTest.shape[0])
    for i in range(XTest.shape[0]):
        distances = np.sqrt(np.sum((XTrain - XTest[i, :])**2, axis=1))
        nearest = np.argsort(distances)[:k]
        yPred[i] = np.bincount(yTrain[nearest].astype(int)).argmax()
        # print(f"bin count: {np.bincount(yTrain[nearest].astype(int))}")
        # print("values of yTrain[nearest]: ", yTrain[nearest])
        # print(f"distances nearest: {nearest} -> yPred[{i}] = {yPred[i]}")
    return yPred

#now we can use the knn function to make predictions
yPred = knn(XTrain, yTrain, XTest, 3)
test_accuracy = np.mean(yPred == yTest)
print('Test accuracy:', test_accuracy)

#now we test knn for different k values
k_values = np.arange(1, 200)
accuracy = np.zeros(len(k_values))
trainAccuracy = np.zeros(len(k_values))
for i in range(len(k_values)):
    yPred = knn(XTrain, yTrain, XTest, k_values[i])
    accuracy[i] = np.mean(yPred == yTest)

for i in range(len(k_values)):
    ytrainPred = knn(XTrain, yTrain, XTrain, k_values[i])
    trainAccuracy[i] = np.mean(ytrainPred == yTrain)

# for i in range(len(k_values)):
#     ytrainPred = knn(XTrain, yTrain, XTrain, k_values[i])
#     trainAccuracy[i] = np.mean(ytrainPred == yTrain)


plt.plot(k_values, accuracy)
plt.plot(k_values, trainAccuracy)
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.legend(['Test', 'Train'])
plt.show()
```

```
Test accuracy: 0.885
```



Comment On Ex3

- in this exercise the data had 256 features, caluculating the distance for each point is a complicated and computationally heavy process which took longer time to be done

- as it can be seen as we increase the value of k the train accuracy decreases and test accuracy increases and we need to find an optimal value for k in this range.

- the lower the value of k the more overfitting over train data and the larger the value of k the lower the accuracy for both train and test sets, since there is a higher number of features and higher number of neighbours resulting in random assignements

- low k -> too granular - capturing false patterns

- large k -> too generalized - ignoring actual existing patterns

STATISTICAL LEARNING AND NEURAL NETWORKS, A.A. 2022/2023

COMPUTER LAB 2 - Model fitting and classification

Duration: 6 hours

Exercise 1 – Model fitting for continuous distributions: Multivariate Gaussian

In this exercise, you will employ a dataset based on the classic dataset *Iris Plants Database* https://archive.ics.uci.edu/ml/datasets/iris. You will be provided a subset of this dataset comprising only two classes (*Iris Setosa* and *Iris Versicolour*), and only two features per class (*petal length* in cm and *petal width* in cm). The objective is to the determine the kind of iris based on the content of the features.

Task: you have to fit class-conditional Gaussian multivariate distributions to the data, and visualize the probability density functions. In particular, you should perform the following:

- Divide the dataset in two parts (*Iris Setosa* which corresponds to class zero and *Iris Versicolour* which correspond to class 1). Then work only on one class at a time.
- Plot the data of each class (use the *plt.scatter( )* function)
- Visualize the histogram of petal length and petal width (use e.g. the *plt.hist( )* function)
- Calculate the maximum likelihood estimate of the mean and covariance matrix under a multivariate Gaussian model, independently for each class (these are the parameters of the class-conditional distributions). Note: is the Gaussian model good for these data?
- Visualize the 2-D joint pdf of petal length and width for the two classes.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import h5py

Dataset1 = h5py.File("Lab2_Ex_1_Iris.hdf5")
Data = np.array(Dataset1.get('Dataset'))
```

In [ ]:
```
Data.shape
```

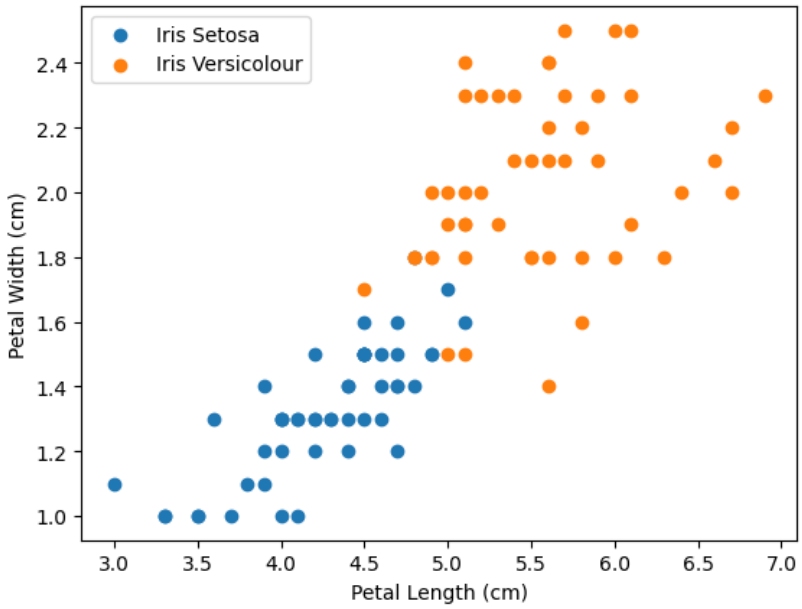
Out[ ]:
```
(100, 3)
```

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import h5py

#Dataset1 = h5py.File("/content/Lab2_Ex_1_Iris.hdf5")
#Data = np.array(Dataset1.get('Dataset'))


import numpy as np
import matplotlib.pyplot as plt
import h5py
from scipy.stats import multivariate_normal
from matplotlib import cm
```

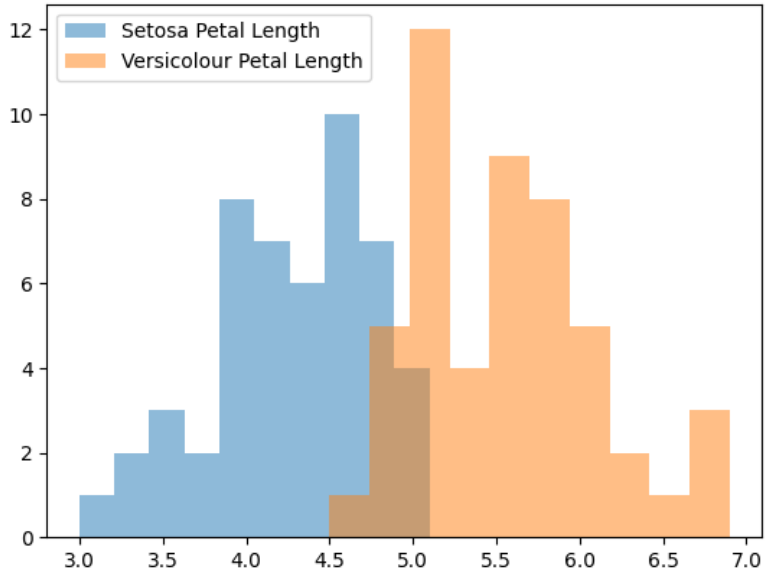
In [ ]:
```
# how many distinct labels are there in the dataset
labels = np.unique(Data[:,-1])
print(labels)
#How many samples are there in the dataset for each label
for label in labels:
  print("Label ", label, " has ", np.sum(Data[:,-1] == label), " samples")
```

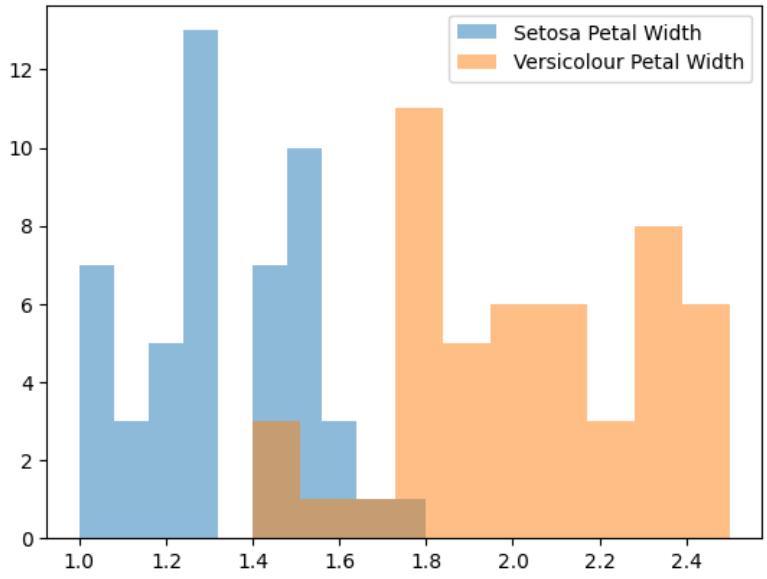
```
[0. 1.]
Label  0.0  has  50  samples
Label  1.0  has  50  samples
```

In [ ]:
```
#Separate the dataset in the two classes, you can use the numpy function argsort and unique to do this.
X = Data[:, :-1]  # feature values
y = Data[:, -1]   # class labels
# Divide the dataset by class
class0 = X[y == 0]
class1 = X[y == 1]

# class0 = np.where(Data[:,-1] == 0)
# class1 = np.where(Data[:,-1] == 1)
# class0 = np.where(Data[:,-1] == 0)
# class1 = np.where(Data[:,-1] == 1)
# class0X = Data[class0[:, :-1]]
# class1X = Data[class1[:, :-1]]
# class0Y = Data[class0X, -1]
# class1Y = Data[class1X, -1]

#Draw the scatter plot of the two classes on the same image
# Scatter plot
plt.scatter(class0[:, 0], class0[:, 1], label='Iris Setosa')
plt.scatter(class1[:, 0], class1[:, 1], label='Iris Versicolour')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal Width (cm)')
plt.legend()
plt.show()
```



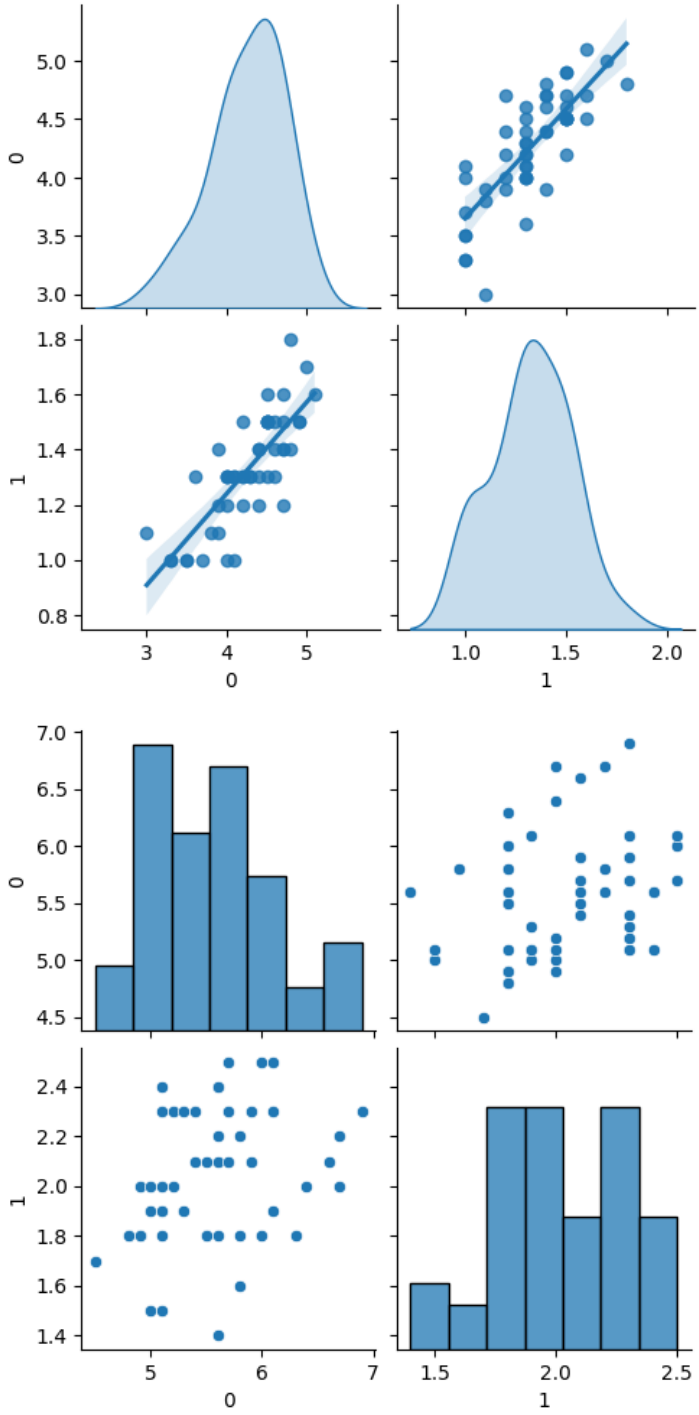
In [ ]:
```
#Visualize the histogram of petal length and petal width (use e.g. the plt.hist() function)
# Histograms
plt.hist(class0[:, 0], alpha=0.5, label='Setosa Petal Length')
plt.hist(class1[:, 0], alpha=0.5, label='Versicolour Petal Length')
plt.legend()
plt.show()

plt.hist(class0[:, 1], alpha=0.5, label='Setosa Petal Width')
plt.hist(class1[:, 1], alpha=0.5, label='Versicolour Petal Width')
plt.legend()
plt.show()
```





In [ ]:
```
import seaborn as sns
import pandas as pd
sns.pairplot(pd.DataFrame(class0), kind='reg', diag_kind='kde')
sns.pairplot(pd.DataFrame(class1))
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x14992d350>



In [ ]:
```
class0.shape
class1.shape
class0.mean(axis=0)
class1.mean(axis=0)
```

Out[ ]: array([5.552, 2.026])

In [ ]:
```
# print(class0)
```

In [ ]:
```
#Calculate mean and covariance matrix under a multivariate Gaussian model. Scalar products can be computed with the function
#The transpose can be obtained with the function np.transpose
#To make a scalar product between arrays in the form [Mx1]x[1xM] starting from 1D array A it may be necessary to add a new ax

covClass0 =  np.matmul(np.transpose(class0 - class0.mean(axis=0)), class0 - class0.mean(axis=0)) / class0.shape[0]
covClass1 =  np.matmul(np.transpose(class1 - class1.mean(axis=0)), class1 - class1.mean(axis=0)) / class1.shape[0]

setosa_mean = np.mean(class0, axis=0)
setosa_cov = np.cov(class0.T)
versicolour_mean = np.mean(class1, axis=0)
versicolour_cov = np.cov(class1.T)
print('Setosa Mean:', setosa_mean)
print('Versicolour Mean:', versicolour_mean)

print('Setosa Covariance:', setosa_cov)
print('Versicolour Covariance:', versicolour_cov)

print('Covariance matrix for class 0:', covClass0)
print('Covariance matrix for class 1:', covClass1)
```

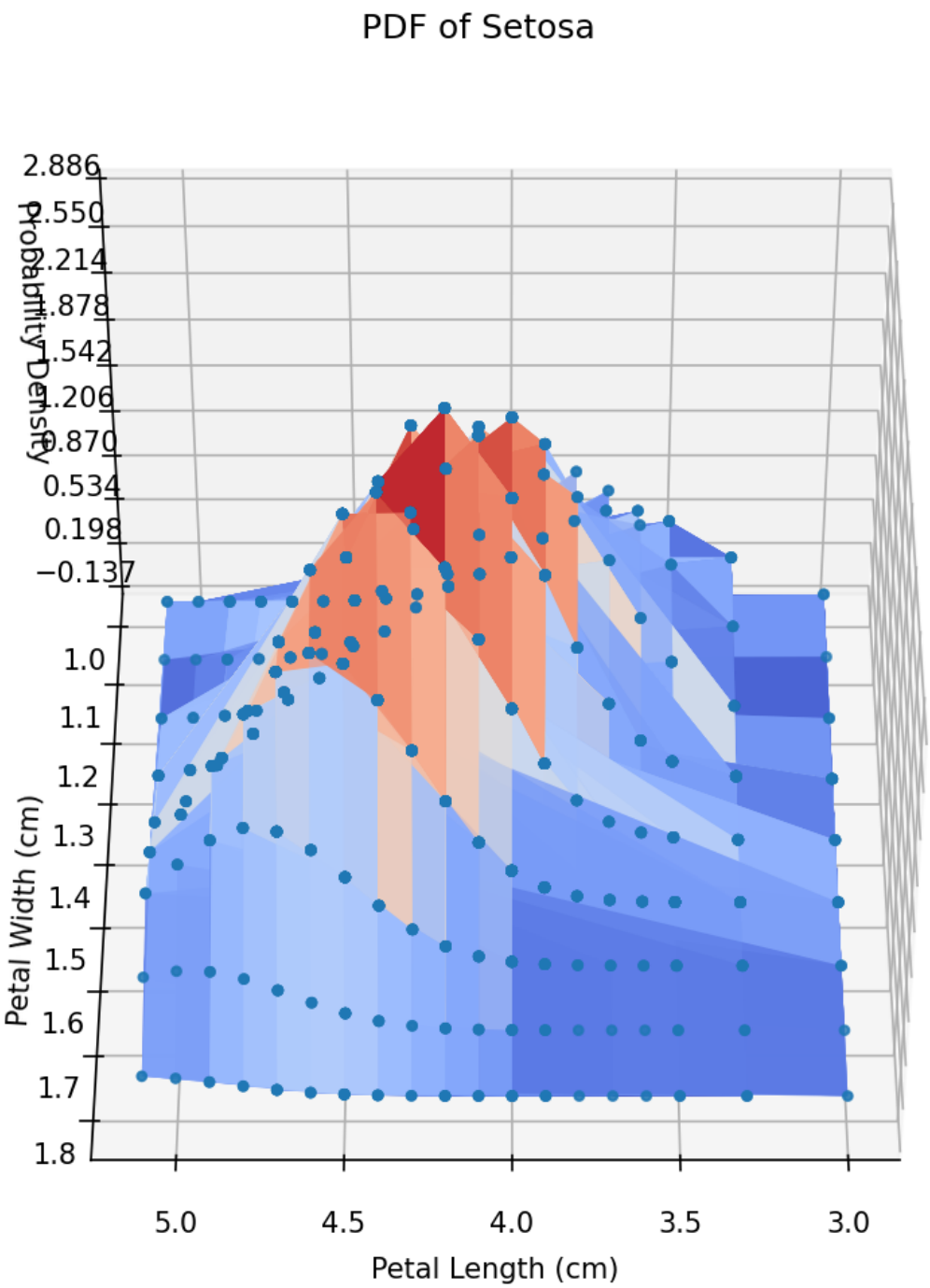
```
Setosa Mean: [4.26  1.326]
Versicolour Mean: [5.552 2.026]
Setosa Covariance: [[0.22081633 0.07310204]
 [0.07310204 0.03910612]]
Versicolour Covariance: [[0.30458776 0.04882449]
 [0.04882449 0.07543265]]
Covariance matrix for class 0: [[0.2164   0.07164 ]
 [0.07164  0.038324]]
Covariance matrix for class 1: [[0.298496 0.047848]
 [0.047848 0.073924]]
```

In [ ]:

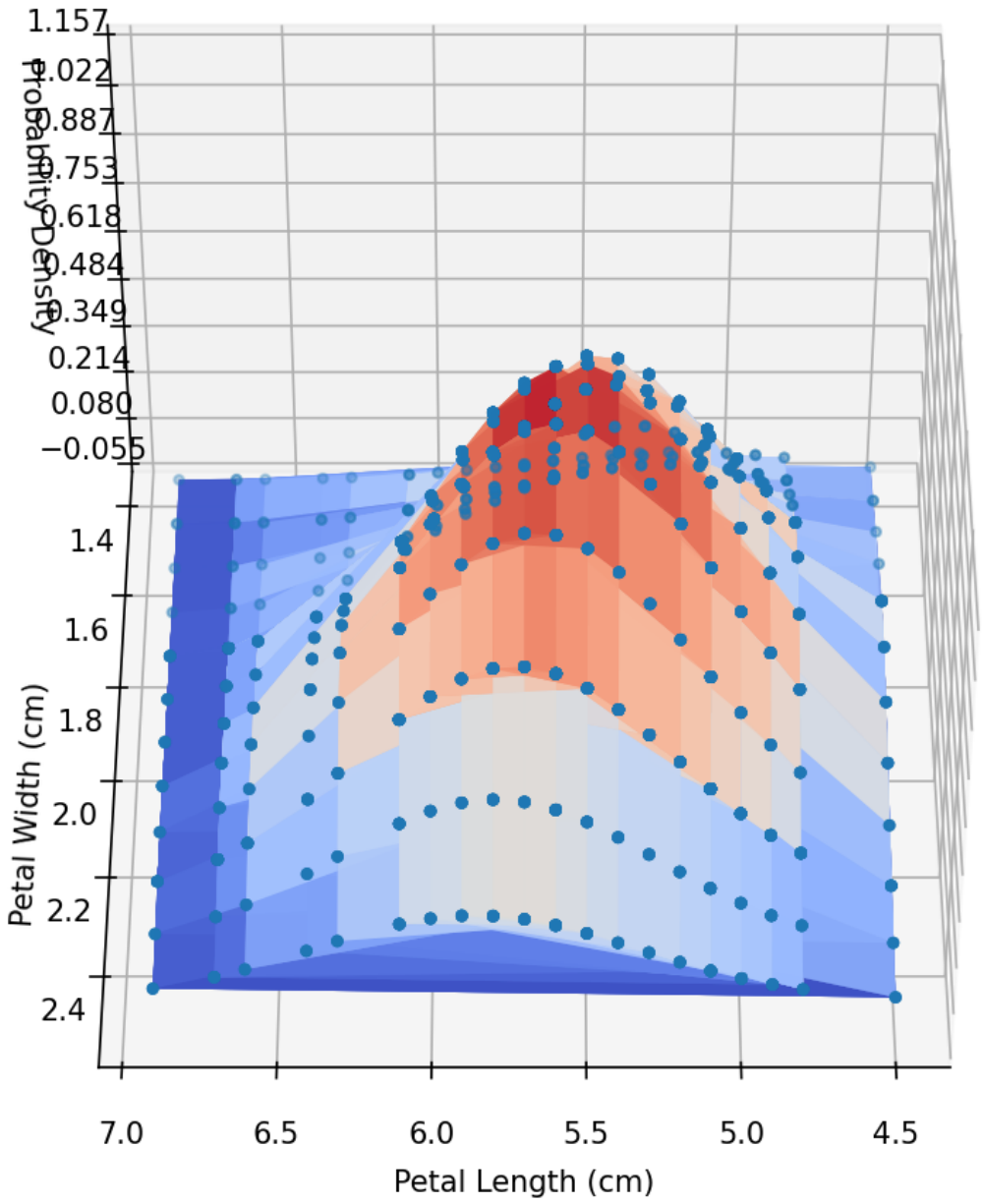
```
#Visualize the 2-D joint pdf of petal length and width, a pdf function can be initialized by providing mean and covariance ma
from scipy.stats import multivariate_normal
from matplotlib import cm
from matplotlib.ticker import LinearLocator

pdf_class0 = multivariate_normal(mean=setosa_mean, cov=setosa_cov)
pdf_class1 = multivariate_normal(mean=versicolour_mean, cov=versicolour_cov)

#now manually calculating the multi-variate normal distribution
# def multivariate_normal_pdf(x, mean, cov):
#     n = mean.shape[0]
#     det = np.linalg.det(cov)
#     inv = np.linalg.inv(cov)
#     norm_const = 1.0 / (np.power((2 * np.pi), n / 2) * np.power(det, 0.5))
#     x_mu = x - mean
#     result = np.exp(-0.5 * (x_mu @ inv @ x_mu.T))
#     return norm_const * result

# pdf_class0 = multivariate_normal_pdf(class0, setosa_mean, setosa_cov)
# pdf_class1 = multivariate_normal_pdf(class1, versicolour_mean, versicolour_cov)

# #now we calculate pdf of the class0 and class1 from scratch
# def plotter(pdfff, data, title):
#     X = data[:, 0]
#     Y = data[:, 1]

#     X, Y = np.meshgrid(X, Y)
#     X_flat = X.flatten()
#     Y_flat = Y.flatten()
#     XY_list = np.concatenate((X_flat[:,np.newaxis],Y_flat[:,np.newaxis]),axis=1)
#     PDF_values = np.reshape(pdfff, np.shape(X))

#     fig, ax = plt.subplots(subplot_kw={"projection": "3d"},figsize=(8, 8), dpi=150)
#     ax.plot_surface(X, Y, PDF_values, cmap=cm.coolwarm, alpha=0.7, linewidth=0)
#     plt.title('PDF of ' + title)
#     plt.xlabel('Petal Length (cm)')
#     plt.ylabel('Petal Width (cm)')
#     ax.set_zlabel('Probability Density')
#     ax.zaxis.set_major_locator(LinearLocator(10))
#     ax.view_init( 45, 90)
#     ax.scatter3D(X_flat, Y_flat, PDF_values, s=10)
#     plt.show()

# plotter(pdf_class0, class0, 'Setosa')
# plotter(pdf_class1, class1, 'Versicolour')

#Create a grid of x and y values on which to sample the pdf, this is done by providing a list of x-y of coordinates to the fu
#A 3D view of the pdf can be obtained using the function ax.plot_surface

#Code Example:

def plotter(pdfff, data, title):
    print(data.shape)
    X = data[:, 0]
    Y = data[:, 1]

    X, Y = np.meshgrid(X, Y)
    X_flat = X.flatten()
    Y_flat = Y.flatten()
    XY_list = np.concatenate((X_flat[:,np.newaxis],Y_flat[:,np.newaxis]),axis=1)
    PDF_values = np.reshape(pdfff.pdf(XY_list), np.shape(X))

    fig, ax = plt.subplots(subplot_kw={"projection": "3d"},figsize=(8, 8), dpi=150)
    ax.plot_surface(X, Y, PDF_values, cmap=cm.coolwarm, alpha=0.7, linewidth=0)
    plt.title('PDF of ' + title)
    plt.xlabel('Petal Length (cm)')
    plt.ylabel('Petal Width (cm)')
    ax.set_zlabel('Probability Density')
    ax.zaxis.set_major_locator(LinearLocator(10))
    ax.view_init( 45, 90)
    ax.scatter3D(X_flat, Y_flat, PDF_values, s=10)
    plt.show()
    print(XY_list.shape)

plotter(pdf_class0, class0, 'Setosa')
plotter(pdf_class1, class1, 'Versicolour')


#To change the orientation of the 3D plot the function ax.view_init(), for a view from above select ax.view_init( 90, 0)
#After visualizing the pdf you can plot the points of the dataset on the estimated pdf using ax.scatter3D()
#For a better visualization of the points we suggest to make the pdf plot semi-transparent using the alpha parameter
#Code Example:
# PDF_points_class0 = pdf_class0.pdf(Features_class0)
#ax.scatter3D(Points_Class0_Feature0, Points_Class0_Feature1, PDF_points_class0, s=10)
```

```
#Note: the sample code was written only for class 0 but two plots have to be done, one for class 0 and one for class 1
#Note 2: the content of variables like Features_class0, Points_Class0_Feature0, Points_Class0_Feature1 should be substituted
```

```
(50, 2)
```




```
(2500, 2)
(50, 2)
```

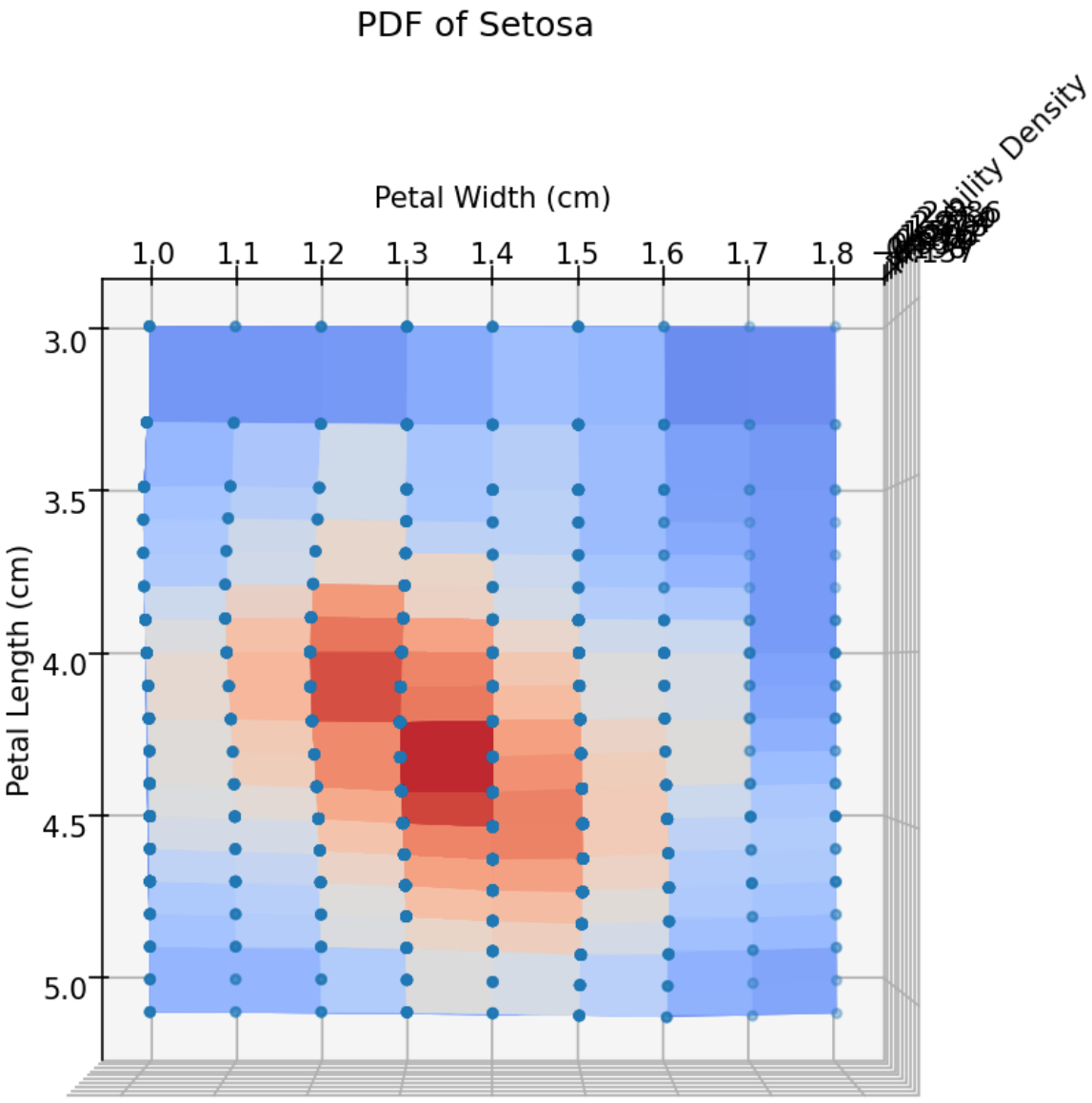


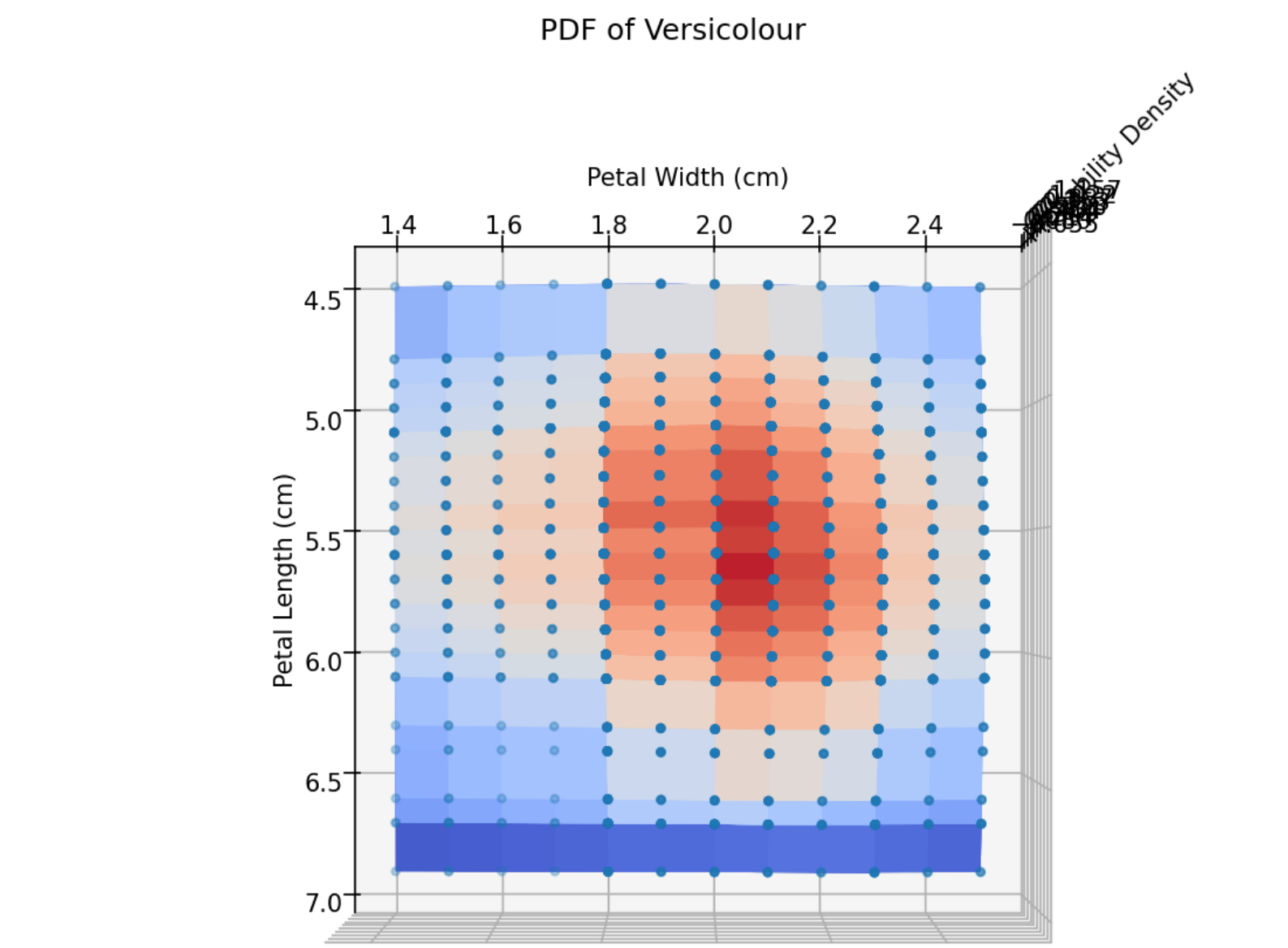
(2500, 2)

In [ ]:

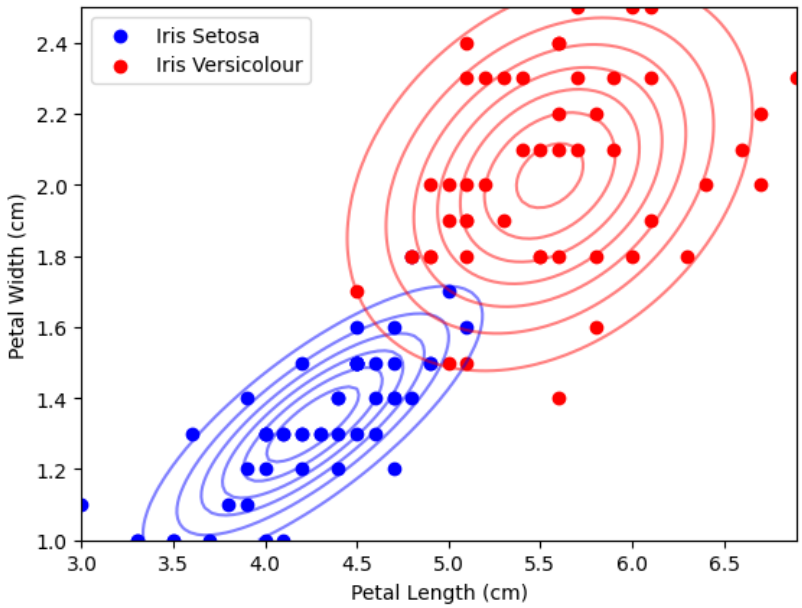
```
#Visualize the 2-D joint pdf of petal length and width, a pdf function can be initialized by providing mean and covariance ma
from scipy.stats import multivariate_normal
from matplotlib import cm
from matplotlib.ticker import LinearLocator

pdf_class0 = multivariate_normal(mean=setosa_mean, cov=setosa_cov)
pdf_class1 = multivariate_normal(mean=versicolour_mean, cov=versicolour_cov)

#Create a grid of x and y values on which to sample the pdf, this is done by providing a list of x-y of coordinates to the fu
#A 3D view of the pdf can be obtained using the function ax.plot_surface

#Code Example:

def plotter(pdfff, data, title):
    X = data[:, 0]
    Y = data[:, 1]

    X, Y = np.meshgrid(X, Y)
    X_flat = X.flatten()
    Y_flat = Y.flatten()
    XY_list = np.concatenate((X_flat[:,np.newaxis],Y_flat[:,np.newaxis]),axis=1)
    PDF_values = np.reshape(pdfff.pdf(XY_list), np.shape(X))

    fig, ax = plt.subplots(subplot_kw={"projection": "3d"},figsize=(8, 8), dpi=150)
    ax.plot_surface(X, Y, PDF_values, cmap=cm.coolwarm, alpha=0.7, linewidth=0)
    plt.title('PDF of ' + title)
    plt.xlabel('Petal Length (cm)')
    plt.ylabel('Petal Width (cm)')
    ax.set_zlabel('Probability Density')
    ax.zaxis.set_major_locator(LinearLocator(10))
    ax.view_init( 90, 0)
    ax.scatter3D(X_flat, Y_flat, PDF_values, s=10)
    plt.show()
```

```
plotter(pdf_class0, class0, 'Setosa')
plotter(pdf_class1, class1, 'Versicolour')


#To change the orientation of the 3D plot the function ax.view_init(), for a view from above select ax.view_init( 90, 0)
#After visualizing the pdf you can plot the points of the dataset on the estimated pdf using ax.scatter3D()
#For a better visualization of the points we suggest to make the pdf plot semi-transparent using the alpha parameter
#Code Example:
# PDF_points_class0 = pdf_class0.pdf(Features_class0)
#ax.scatter3D(Points_Class0_Feature0, Points_Class0_Feature1, PDF_points_class0, s=10)
#Note: the sample code was written only for class 0 but two plots have to be done, one for class 0 and one for class 1
#Note 2: the content of variables like Features_class0, Points_Class0_Feature0, Points_Class0_Feature1 should be substituted
```





In [ ]:
```
x, y = np.meshgrid(np.linspace(min(class0[:, 0].min(), class1[:, 0].min()),
                               max(class0[:, 0].max(), class1[:, 0].max()), 100),
                   np.linspace(min(class0[:, 1].min(), class1[:, 1].min()),
                               max(class0[:, 1].max(), class1[:, 1].max()), 100))

pos = np.dstack((x, y))
pdf_setosa = multivariate_normal.pdf(pos, setosa_mean, setosa_cov)
pdf_versicolour = multivariate_normal.pdf(pos, versicolour_mean, versicolour_cov)

plt.contour(x, y, pdf_setosa, colors='blue', alpha=0.5)
plt.contour(x, y, pdf_versicolour, colors='red', alpha=0.5)
plt.scatter(class0[:, 0], class0[:, 1], color='blue', label='Iris Setosa')
plt.scatter(class1[:, 0], class1[:, 1], color='red', label='Iris Versicolour')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal Width (cm)')
plt.legend()
plt.show()
```



Student's comments to Exercise 1

Add comments to the results of Exercise 1 here (may use LateX for formulas if needed).

Exercise 2 - Model fitting for discrete distributions: Bag of Words

In this exercise, you will employ a real dataset (file *SMSSpamCollection*). The SMS Spam Collection v.1 (https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset) is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged acording being ham (legitimate) or spam. Task: you have to fit the parameters employed by a Naïve Bayes Classifier, using a Bernoulli model. Under this model, the parameters are:

- π_c, the prior probabilities of each class.
- θ_{jc}, the probability that feature j is equal to 1 in class c.

Model fitting can be done using the pseudocode at the end of the Lecture 3 slides.

Display the class-conditional densities θ_{jc1} and θ_{jc2}. Try to identify "uninformative" features (i.e., features j such that $\theta_{jc1} \simeq \theta_{jc2}$).

In [ ]:
```
# this is a bad day
# pham = 0.6
# pspam= 0.4

# p(this|ham) = 0.2
# p(is|ham) = 0.1
# p(a|ham) = 0.3
# p(bad|ham) = 0.1
# p(day|ham) = 0.3
# p(this|spam) = 0.9
# p(is|spam) = 0.1
# p(a|spam) = 0.1
# p(bad|spam) = 0.0


# p(ham|this is a bad day) = p(this|ham) * p(is|ham) * p(a|ham) * p(bad|ham) * p(day|ham) * pham
# p(spam|this is a bad day) = p(this|spam) * p(is|spam) * p(a|spam) * p(bad|spam) * p(day|spam) * pspam
#probability of presence of a word in a test document given the class, the formula is the following:
# p(word|class) = (number of documents in the class containing the word + 1) / (number of documents in the class + 2)
# theta*p(word|class) + (1-theta)*p(word|class)
#theta is the probality of the word being present in the class and 1-theta is the probability of the word being present in t

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer

# reading the data
data = pd.read_csv("SMSSpamCollection", encoding="ISO-8859-1", sep="\t", header=None)
data.rename(columns={0: "labels", 1: "text"}, inplace=True)
display(data)

# Transform data to bag of word representation
#Featurenames look like this: ['aa', 'aah', 'aaniye', 'aaoooright', 'aathilove', 'ab', 'abbey', 'abdomen', 'abeg', 'abel']
#The bag of word representation is a matrix where each row is a document and each column is a word and it lloks like the fol
# 0 1 0 0 0 0 0 0 0 0
# 0 1 1 1 0 0 0 0 0 0
bagger = CountVectorizer(
    max_features=2500,
    binary=True,  # Bernulli Model, this helps to remove the frequency of the word and only consider if the word is present
    token_pattern=r"(?u)\b[a-zA-Z][a-zA-Z]+\b",
)
bag = bagger.fit_transform(data["text"]).toarray()
feature_names = bagger.get_feature_names_out()
```

```
data = pd.concat([data, pd.DataFrame(bag, columns=feature_names)], axis=1)
display(data)
X_train = data.iloc[:3000,2:].to_numpy()
X_test = data.iloc[3000:,2:].to_numpy()
y_train = data.iloc[:3000,0].to_numpy() == 'ham'
y_test = data.iloc[3000:,0].to_numpy() == 'ham'
```

	labels	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will Ã¼ b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

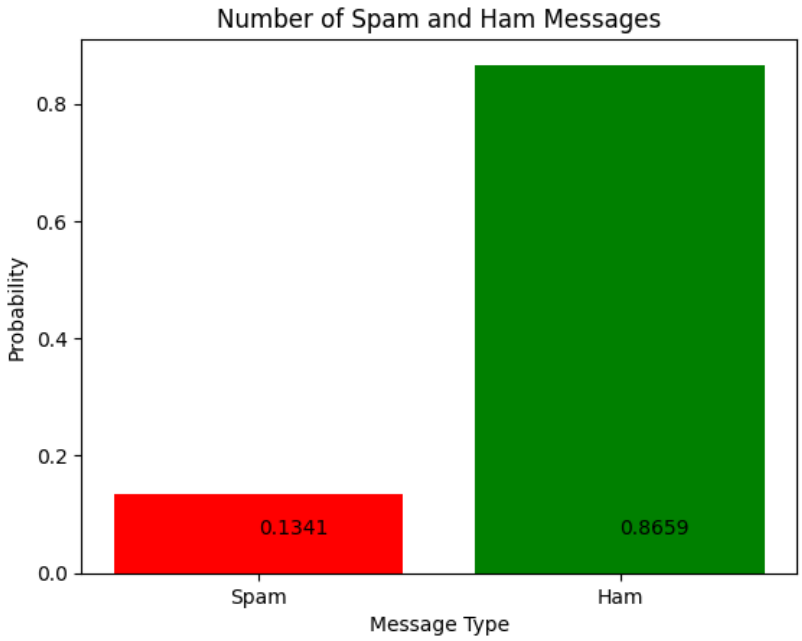
	labels	text	aah	aathi	abi	abiola	able	about	abt	abta	...	you	your	yours	yourself	yr	yrs	yummy	yun	yup	zed
0	ham	Go until jurong point, crazy.. Available only ...	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	ham	Ok lar... Joking wif u oni...	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	ham	U dun say so early hor... U c already then say...	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	ham	Nah I don't think he goes to usf, he lives aro...	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5568	ham	Will Ã¼ b going to esplanade fr home?	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5569	ham	Pity, * was in mood for that. So...any other s...	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5570	ham	The guy did some bitching but I acted like i'd...	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5571	ham	Rofl. Its true to its name	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5572 rows × 2502 columns

In [ ]:
```
# from google.colab import drive
# drive.mount('/content/drive')
```

In [ ]:
```
#Evaluate the probabilities of the two classes, and the class conditional densities.
n_spam = np.sum(data['labels'] == 'spam')
n_ham = np.sum(data['labels'] == 'ham')
n_total = len(data)
prior_spam = n_spam / n_total
prior_ham = n_ham / n_total
print('P(Spam):', prior_spam)
print('P(Ham):', prior_ham)

#plot the bar plot of the two classes with two colors
#write the value inside the bars
plt.bar(['Spam', 'Ham'], [prior_spam, prior_ham], color=['red', 'green'])
plt.text(0, prior_spam/2, str(prior_spam.round(4)))
plt.text(1, prior_spam/2, str(prior_ham.round(4)))
plt.xlabel('Message Type')
plt.ylabel('Probability')
plt.title('Number of Spam and Ham Messages')
plt.show()
```

```
P(Spam): 0.13406317300789664
P(Ham): 0.8659368269921034
```



In [ ]:
```
#What is the probability of the word "free" given the class spam and ham?
# p(word|class) = (number of documents in the class containing the word + 1) / (number of documents in the class + 2)
# theta*p(word|class) + (1-theta)*p(word|class)
#theta is the probality of the word being present in the class and 1-theta is the probability of the word being Absent in the
#we count the number of occurences of the word in the class and divide by the number of documents in the class
#we add 1 to the numerator and 2 to the denominator to avoid division by zero and add-one smoothing
Njc_class0 = [np.sum(X_train[y_train == 0][:, i]) for i in range(X_train.shape[1])]
Njc_class1 = [np.sum(X_train[y_train == 1][:, i]) for i in range(X_train.shape[1])]
```

In [ ]:
```
theta_class0 = np.array(Njc_class0) / np.sum(Njc_class0)
theta_class1 = np.array(Njc_class1) / np.sum(Njc_class1)
```

In [ ]:
```
#uninformative features are those that have zero probability in both classes and informative features are those that have no
uninformative_features = np.where((theta_class0 == 0) & (theta_class1 == 0))[0] #where both classes have zero probability
print('Uninformative Features:', uninformative_features.shape)
informative_features = np.where((theta_class0 != 0) | (theta_class1 != 0))[0] #where at least one class has non-zero probabi
print('Informative Features:', informative_features.shape)
```

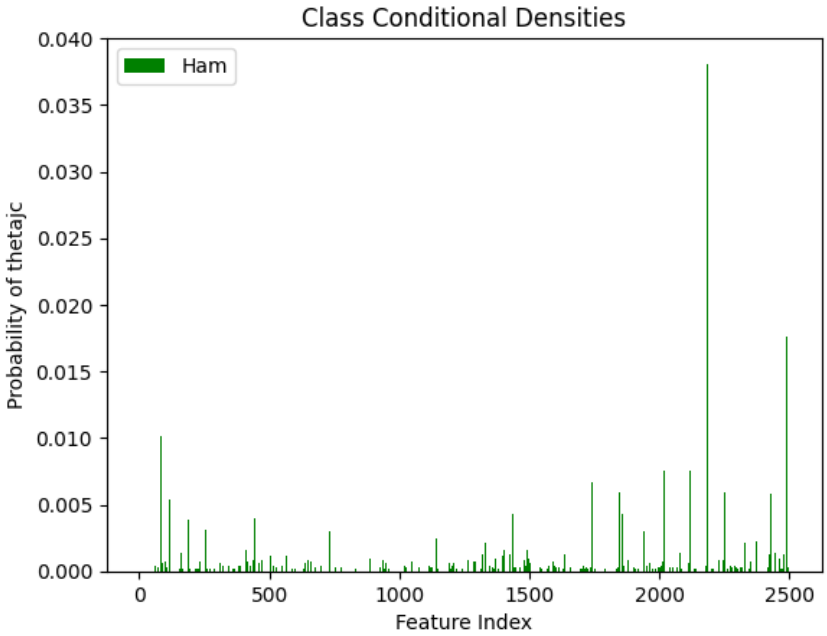
```
Uninformative Features: (60,)
Informative Features: (2440,)
```

In [ ]:
```
print('Number of Uninformative Features:', len(uninformative_features))
print('Uninformative Features:', uninformative_features)

print('Number of Informative Features:', len(informative_features))
print('Informative Features:', informative_features)

print('Njc Class 0:', Njc_class0)
print('Theta Class 0:', theta_class0)

print('Njc Class 1:', Njc_class1)
print('Theta Class 1:', theta_class1)
```

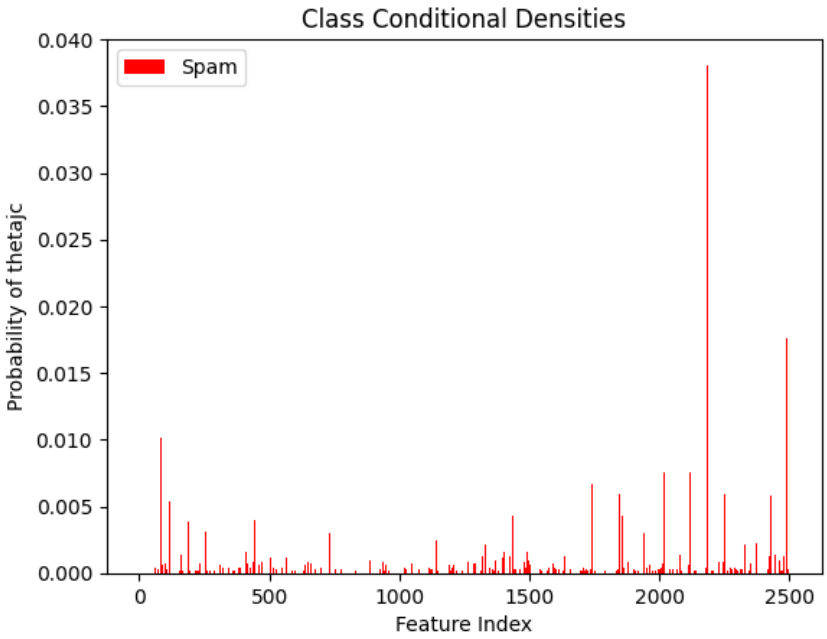
```
Number of Uninformative Features: 60
Uninformative Features: [  12   33  139  150  153  177  231  242  247  300  302  318  353  423
  433  437  528  567  606  620  729  766  793  894  942  987 1032 1132
 1183 1189 1227 1254 1307 1382 1402 1510 1526 1579 1611 1626 1633 1647
 1651 1684 1726 1853 1940 2014 2090 2120 2163 2166 2190 2264 2299 2340
 2367 2411 2449 2459]
Number of Informative Features: 2440
Informative Features: [   0    1    2 ... 2497 2498 2499]
Njc Class 0: [0, 0, 0, 0, 0, 7, 0, 2, 6, 0, 0, 2, 0, 0, 0, 9, 0, 0, 3, 2, 0, 0, 0, 0, 0, 0, 4, 7, 0, 2, 0, 0, 1, 0, 0, 1, 0,
3, 0, 0, 2, 0, 6, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 1, 0, 17, 0, 0, 3, 1, 1, 0, 0, 2, 0, 0, 6, 1,
0, 1, 0, 0, 0, 9, 71, 0, 3, 0, 4, 4, 5, 0, 0, 0, 7, 0, 0, 1, 0, 0, 5, 0, 0, 0, 0, 0, 0, 2, 0, 14, 0, 0, 0, 0, 0, 0, 1, 0, 38,
5, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 3, 0, 0, 18, 6, 0, 0, 1, 0, 0, 0, 0, 0, 0, 14, 0, 0, 0, 13, 0, 5, 0, 1, 0, 0, 2, 0, 1, 0, 1
0, 3, 0, 10, 19, 2, 0, 3, 1, 1, 13, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 27, 0, 2, 1, 0, 0, 1, 21, 0,
0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 6, 0, 1, 1, 1, 1, 0, 1, 0, 1, 4, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 5, 0, 0, 0, 0, 15,
0, 3, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 1, 1, 0, 22, 0, 2, 0, 2, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 6, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 16, 0, 0, 0, 0, 1, 0, 0, 167, 0, 0, 9, 0, 0, 0, 2, 5, 1, 4, 0, 16, 0, 16, 0, 0, 0, 0, 0,
1, 0, 3, 0, 4, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 39, 0, 0, 0, 0, 0, 0, 0, 4, 4, 3, 3, 0, 0, 0, 0, 0, 0, 1, 0, 0, 18, 0, 1, 0, 5, 5, 0,
1, 0, 23, 1, 2, 1, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 3, 5, 1, 1, 0, 0, 3, 0, 1, 61, 0, 0, 0, 0, 3, 0, 2, 0, 0,
7, 0, 31, 0, 14, 0, 0, 0, 0, 11, 2, 0, 13, 0, 5, 29, 2, 0, 2, 0, 0, 0, 2, 6, 2, 3, 0, 1, 7, 1, 3, 0, 2, 0, 1, 0, 6, 9, 1, 0,
0, 0, 28, 2, 10, 1, 0, 0, 0, 1, 0, 1, 4, 0, 13, 3, 1, 3, 0, 4, 0, 0, 0, 0, 0, 0, 1, 0, 1, 6, 1, 7, 1, 5, 0, 32, 0, 0, 3,
4, 0, 6, 3, 8, 24, 0, 1, 0, 0, 0, 0, 0, 3, 0, 0, 1, 0, 2, 0, 0, 6, 2, 8, 1, 9, 5, 0, 0, 0, 9, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0,
0, 3, 0, 0, 2, 0, 8, 0, 0, 0, 0, 2, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 6, 0, 1, 1, 9, 0, 0, 0, 0, 0,
2, 0, 8, 0, 0, 0, 2, 0, 0, 0, 5, 0, 0, 0, 0, 4, 2, 9, 1, 0, 1, 7, 1, 2, 1, 23, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 2, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 6, 6, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 4, 9, 0, 0, 0, 0, 0, 6, 1, 0, 1, 1, 0, 0, 6, 0, 4, 3, 0, 2, 1, 3, 0, 5,
2, 2, 6, 6, 3, 15, 0, 2, 0, 0, 0, 0, 0, 0, 0, 5, 2, 4, 1, 2, 23, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 7, 0, 0,
3, 0, 0, 0, 1, 0, 1, 0, 0, 3, 8, 3, 3, 3, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 11, 1, 21, 0, 0, 0, 0, 0, 0, 4, 0, 1, 2, 0, 1, 1, 0, 0, 0, 0, 5, 4, 0, 1, 2, 2, 2, 1, 2, 0, 0, 1, 89, 0, 0, 0, 0, 1,
0, 0, 4, 0, 1, 0, 0, 0, 96, 2, 9, 5, 0, 0, 0, 1, 0, 5, 4, 0, 0, 1, 0, 0, 62, 0, 0, 0, 0, 0, 0, 0, 3, 0, 2, 1, 0, 0, 0, 2, 8,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 48, 0, 1, 4, 7, 10, 0, 3, 0, 2, 0, 5, 0, 0, 0, 0, 0, 0, 17,
0, 0, 0, 3, 0, 0, 0, 6, 0, 0, 0, 0, 0, 1, 3, 4, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 28, 0, 0, 4, 0, 0, 0, 4, 0, 0, 7, 0,
0, 0, 0, 0, 0, 6, 2, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 15, 0, 0, 0, 73, 1, 0, 1, 0, 0, 0, 0, 0, 2, 1, 2, 0, 0,
0, 0, 0, 4, 13, 2, 6, 5, 9, 10, 2, 0, 1, 0, 0, 0, 1, 0, 4, 0, 0, 3, 0, 0, 2, 0, 17, 0, 2, 1, 2, 0, 0, 0, 0, 0, 4, 0, 0, 3,
0, 0, 0, 9, 1, 0, 2, 0, 5, 1, 0, 0, 0, 0, 8, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 2, 1, 0, 1, 0, 0, 0, 7, 1, 0, 16, 0, 0,
0, 1, 2, 0, 0, 3, 5, 0, 42, 3, 0, 0, 1, 0, 2, 0, 0, 0, 0, 8, 0, 3, 2, 0, 0, 0, 1, 0, 0, 2, 0, 0, 1, 0, 1, 0, 5, 0, 0, 3, 1,
0, 0, 7, 0, 0, 82, 0, 0, 1, 0, 0, 0, 0, 16, 0, 2, 0, 0, 0, 0, 0, 0, 1, 2, 0, 1, 1, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 2,
0, 0, 0, 3, 0, 0, 1, 0, 0, 1, 0, 0, 51, 0, 0, 0, 0, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 2, 0, 0, 0, 1, 0, 2, 0, 0, 0, 0, 13, 0, 1,
0, 0, 0, 0, 0, 2, 0, 3, 9, 17, 0, 0, 3, 7, 1, 0, 18, 0, 0, 0, 0, 0, 0, 7, 7, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 7, 0, 16, 4, 1, 3, 0, 2, 1, 1, 14, 2, 0, 1, 3, 0, 0, 2, 4, 4, 3, 3, 4, 0, 1, 0,
0, 0, 1, 0, 1, 6, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 7, 0, 0, 2, 0, 0, 0, 1, 5, 0, 8, 2, 3, 0, 4, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 1, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 2, 0, 0, 0, 3, 5, 0, 7, 0, 0, 5, 6, 0, 0, 14, 0,
0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 5, 0, 0, 0, 0, 5, 1, 1, 2, 1, 0, 15, 9, 0, 2, 0, 1, 0, 0, 18, 0, 0, 0, 15, 3, 0, 0, 0, 6, 0,
0, 0, 0, 0, 0, 0, 1, 14, 53, 7, 3, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 5, 1, 1, 0, 1, 0, 18, 0, 1, 0, 7, 0, 0, 0, 1, 0, 0, 32,
0, 2, 4, 1, 0, 0, 0, 0, 0, 0, 0, 0, 12, 4, 0, 7, 0, 0, 0, 1, 8, 2, 0, 0, 0, 11, 0, 0, 0, 1, 0, 0, 1, 4, 0, 0, 0, 9, 1, 11,
0, 2, 0, 0, 32, 2, 7, 9, 0, 1, 0, 1, 1, 0, 0, 37, 0, 0, 30, 1, 2, 0, 0, 0, 2, 1, 0, 15, 0, 2, 2, 108, 1, 8, 1, 0, 11, 4, 0,
0, 0, 0, 0, 2, 0, 55, 2, 18, 8, 1, 1, 0, 0, 0, 4, 0, 0, 0, 2, 0, 75, 0, 6, 0, 0, 0, 44, 3, 0, 1, 8, 0, 0, 11, 3, 93, 14,
0, 7, 1, 0, 1, 0, 0, 4, 0, 0, 44, 38, 0, 0, 0, 11, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 2, 1, 0, 0, 7, 4, 0, 3, 0, 0, 3, 0, 1,
0, 0, 3, 0, 2, 0, 2, 24, 1, 0, 0, 1, 0, 0, 1, 0, 2, 0, 0, 26, 6, 2, 0, 8, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 3, 0,
0, 0, 0, 0, 0, 11, 2, 9, 0, 0, 0, 29, 1, 0, 0, 6, 5, 0, 3, 20, 7, 3, 2, 0, 7, 0, 7, 2, 2, 3, 0, 0, 0, 0, 1, 0, 0, 2, 2, 0, 0,
3, 9, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 9, 0, 8, 45, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 3, 0, 1, 0, 1, 0,
2, 0, 1, 0, 0, 1, 0, 4, 0, 4, 0, 0, 0, 0, 0, 12, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 3, 0, 0, 7, 0, 7, 5, 0, 0, 1, 0,
1, 10, 1, 0, 1, 0, 0, 0, 3, 3, 18, 1, 0, 0, 0, 1, 1, 5, 2, 1, 2, 0, 0, 1, 3, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 5, 2,
1, 47, 2, 0, 6, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 3, 0, 0, 0, 7, 1, 2, 2, 0, 0, 1, 2, 14, 3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
1, 0, 0, 11, 3, 0, 0, 2, 0, 0, 1, 0, 0, 0, 0, 0, 11, 0, 0, 0, 2, 2, 0, 0, 0, 3, 3, 7, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 7, 0, 8, 0, 1, 0, 0, 0, 0, 14, 2, 2, 1, 0, 0, 0, 42, 1, 0, 0, 7, 0, 0, 2, 0, 0, 0, 0, 30, 9, 1, 1,
0, 3, 11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 2, 0, 0, 0, 2, 0, 4, 0, 0, 2, 0, 0, 15, 0, 0, 0, 0, 0, 0, 0, 0, 2,
0, 2, 0, 2, 1, 1, 0, 4, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 21, 0, 0,
0, 17, 0, 0, 0, 0, 3, 0, 3, 2, 7, 0, 0, 0, 0, 0, 0, 1, 0, 5, 4, 2, 0, 0, 2, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 6, 1, 0, 6, 14, 5, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 2, 0, 0, 0, 2, 0, 1, 4, 0, 2, 3, 7, 0, 2, 0, 5, 0, 4, 0, 2, 0, 53, 0, 0, 4, 0, 0, 1,
1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 7, 2, 3, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 7, 0, 0, 1, 0, 3, 0, 0, 3, 0, 0,
0, 0, 2, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 10, 0, 2, 0, 1, 1, 0, 0, 2, 0, 0, 0, 1, 5, 2, 0, 0, 2, 0, 0, 0, 0, 0, 9, 0, 1, 0, 0,
0, 1, 2, 6, 2, 4, 0, 0, 0, 0, 53, 2, 0, 0, 2, 9, 0, 1, 0, 1, 7, 0, 0, 13, 0, 94, 0, 1, 0, 4, 6, 0, 8, 1, 9, 1, 0, 1, 0, 0,
7, 39, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2, 6, 0, 0, 0, 0, 0, 3, 3, 3, 2, 0, 0, 5, 268, 0, 2, 1
3, 9, 0, 0, 0, 0, 1, 0, 0, 5, 22, 14, 2, 0, 1, 0, 12, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 2, 0, 0, 6, 3, 0, 1, 0, 0,
0, 6, 8, 7, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 99, 3, 3, 6, 6, 0, 0, 4, 0, 42, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2, 2, 0,
6, 11, 4, 14, 7, 3, 1, 0, 1, 74, 2, 0, 35, 0, 0, 0, 0, 6, 0, 5, 1, 3, 0, 0, 0, 0, 0, 2, 13, 0, 0, 3, 1, 5, 0, 5, 0, 3, 0, 1,
0, 1, 11, 2, 2, 0, 0, 0, 0, 2, 4, 0, 2, 3, 0, 0, 2, 0, 1, 10, 15, 0, 0, 0, 0, 10, 0, 0, 1, 0, 0, 0, 0, 1, 0, 7, 10, 1, 1, 1,
6, 1, 5, 0, 0, 0, 0, 0, 0, 0, 1, 25, 0, 0, 0, 0, 1, 1, 0, 0, 0, 31, 1, 3, 16, 4, 0, 0, 2, 3, 0, 0, 0, 2, 1, 12, 0, 1, 7, 0,
2, 0, 0, 4, 1, 0, 20, 0, 0, 0, 5, 2, 0, 0, 0, 22, 0, 38, 0, 10, 0, 1, 0, 0, 2, 0, 0, 0, 51, 2, 0, 1, 17, 0, 9, 0, 0, 0, 1,
1, 41, 0, 0, 0, 0, 0, 17, 0, 3, 0, 0, 0, 1, 0, 0, 0, 0, 0, 10, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 56, 0, 0, 3, 7, 0, 3, 8, 0, 0,
1, 2, 0, 0, 0, 1, 6, 6, 0, 0, 1, 9, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 124, 115, 9, 1, 8, 2, 0, 0, 0, 3]
Theta Class 0: [0.         0.         0.         ... 0.         0.         0.00042644]
Njc Class 1: [1, 4, 1, 7, 15, 77, 13, 0, 0, 4, 2, 2, 0, 2, 2, 12, 2, 6, 0, 1, 1, 15, 2, 3, 2, 1, 9, 0, 1, 1, 5, 1, 0, 0, 4,
2, 14, 50, 19, 5, 35, 7, 1, 2, 8, 23, 2, 7, 3, 17, 3, 3, 5, 3, 3, 5, 1, 3, 7, 2, 3, 0, 6, 0, 2, 133, 2, 10, 5, 0, 50, 17, 4,
39, 34, 1, 123, 0, 2, 0, 1, 38, 3, 50, 390, 6, 0, 18, 2, 0, 7, 1, 1, 3, 61, 3, 7, 10, 1, 34, 2, 12, 4, 3, 6, 2, 1, 3, 2, 4,
4, 2, 3, 3, 2, 0, 12, 204, 2, 2, 2, 2, 4, 1, 2, 2, 29, 2, 3, 0, 1, 2, 74, 0, 42, 10, 8, 3, 7, 0, 2, 8, 3, 199, 3, 2, 1, 0,
2, 0, 0, 0, 2, 0, 7, 3, 3, 2, 0, 0, 2, 1, 0, 8, 10, 36, 0, 16, 61, 14, 3, 1, 4, 3, 6, 2, 4, 0, 1, 2, 7, 7, 2, 5, 2, 5, 8, 4,
2, 172, 10, 20, 3, 3, 17, 4, 48, 2, 1, 27, 2, 3, 1, 2, 21, 9, 1, 4, 3, 22, 3, 24, 13, 3, 17, 1, 4, 3, 2, 3, 13, 22, 3, 0, 6,
2, 2, 1, 2, 6, 2, 0, 6, 0, 2, 2, 5, 3, 0, 5, 8, 6, 0, 2, 1, 9, 2, 0, 7, 2, 9, 5, 1, 1, 9, 3, 1, 9, 0, 8, 1, 9, 2, 3, 12, 2,
2, 1, 2, 1, 1, 8, 1, 5, 2, 12, 4, 3, 2, 2, 3, 5, 2, 2, 14, 1, 13, 225, 2, 33, 4, 3, 3, 67, 3, 5, 3, 4, 3, 1, 0, 120, 0, 16,
0, 4, 4, 1, 9, 6, 2, 0, 14, 0, 4, 214, 1, 1, 0, 3, 27, 25, 4, 1, 27, 1, 2, 3, 1, 2, 13, 1, 1, 3, 10, 7, 2, 5, 2, 2, 9, 3, 0,
1, 0, 0, 1, 2, 2, 3, 2, 0, 2, 0, 4, 12, 4, 4, 3, 0, 2, 0, 2, 4, 1, 2, 1, 2, 25, 2, 4, 1, 2, 8, 2, 11, 1, 5, 3, 1, 2, 2, 3,
1, 5, 0, 5, 3, 5, 1, 1, 1, 0, 29, 8, 2, 3, 0, 2, 10, 5, 3, 0, 2, 0, 2, 1, 6, 3, 5, 3, 3, 1, 1, 0, 10, 3, 11, 114, 1, 9, 6, 2
2, 0, 1, 1, 8, 1, 3, 4, 0, 4, 0, 0, 0, 4, 7, 0, 4, 1, 2, 2, 1, 1, 7, 0, 1, 1, 4, 3, 2, 18, 6, 0, 4, 34, 7, 0, 2, 27, 4, 2,
7, 6, 4, 12, 1, 1, 5, 6, 1, 2, 4, 2, 0, 1, 0, 2, 0, 3, 1, 5, 3, 2, 1, 0, 0, 5, 7, 1, 5, 65, 10, 6, 2, 1, 5, 1, 0, 6, 1, 6, 2
2, 6, 0, 0, 1, 108, 18, 11, 4, 5, 52, 2, 2, 2, 0, 6, 9, 14, 2, 5, 2, 4, 1, 3, 1, 0, 0, 0, 2, 15, 3, 3, 8, 4, 1, 1, 4, 61, 2
2, 14, 6, 3, 1, 1, 5, 5, 3, 0, 3, 3, 19, 2, 1, 3, 2, 14, 0, 1, 3, 2, 1, 4, 0, 10, 178, 0, 2, 4, 18, 12, 6, 3, 0, 3, 6, 45,
4, 72, 30, 73, 3, 2, 1, 5, 32, 7, 0, 3, 5, 4, 1, 2, 9, 1, 2, 6, 2, 12, 7, 2, 5, 6, 1, 4, 0, 15, 1, 28, 17, 4, 1, 2, 9, 22,
2, 2, 10, 23, 0, 4, 4, 0, 0, 2, 1, 6, 2, 9, 11, 3, 4, 6, 3, 1, 19, 3, 0, 4, 2, 3, 0, 0, 2, 17, 2, 11, 3, 5, 0, 0, 1, 2, 1,
1, 2, 4, 2, 2, 0, 8, 31, 15, 14, 16, 2, 7, 17, 4, 3, 2, 3, 6, 1, 5, 3, 1, 1, 2, 2, 3, 1, 5, 0, 2, 1, 4, 3, 8, 4, 4, 2, 3, 2,
10, 0, 3, 0, 2, 0, 9, 9, 3, 3, 4, 4, 2, 6, 1, 3, 35, 12, 5, 12, 1, 4, 3, 28, 3, 2, 3, 3, 6, 2, 4, 3, 2, 3, 0, 2, 8, 27, 26,
5, 21, 8, 3, 3, 33, 3, 1, 2, 2, 3, 0, 1, 2, 3, 2, 0, 1, 3, 2, 0, 1, 0, 0, 4, 12, 1, 0, 244, 3, 2, 6, 0, 9, 20, 3, 4, 7, 1,
9, 3, 1, 25, 0, 0, 0, 1, 3, 4, 5, 10, 20, 27, 11, 6, 1, 6, 3, 81, 0, 15, 1, 8, 1, 13, 3, 10, 5, 0, 2, 2, 3, 3, 6, 0, 2, 3,
2, 2, 2, 1, 5, 2, 5, 1, 0, 9, 5, 5, 3, 2, 1, 3, 3, 0, 0, 143, 7, 6, 31, 0, 4, 2, 15, 1, 4, 1, 47, 3, 4, 4, 5, 2, 3, 124, 24,
16, 10, 96, 2, 6, 25, 115, 8, 5, 1, 4, 3, 1, 121, 1, 9, 3, 1, 2, 3, 52, 4, 2, 1, 6, 1, 3, 124, 0, 29, 2, 21, 3, 1, 12, 18,
6, 5, 43, 11, 29, 1, 3, 12, 0, 15, 0, 2, 8, 1, 1, 4, 0, 2, 12, 2, 9, 6, 1, 6, 54, 6, 0, 44, 3, 1, 14, 229, 9, 6, 1, 26, 1, 9
8, 11, 2, 16, 4, 19, 5, 8, 3, 2, 2, 25, 19, 51, 64, 51, 0, 53, 4, 0, 51, 1, 33, 1, 6, 4, 0, 7, 10, 0, 3, 8, 0, 2, 6, 3, 1, 8
```

```
6, 0, 2, 2, 1, 3, 2, 54, 1, 5, 0, 3, 4, 3, 5, 1, 16, 9, 25, 152, 5, 3, 3, 2, 1, 0, 0, 0, 6, 2, 7, 3, 2, 12, 2, 1, 3, 2, 3,
0, 0, 1, 1, 12, 6, 2, 0, 1, 3, 177, 1, 1, 7, 20, 30, 3, 4, 4, 5, 1, 403, 0, 2, 2, 1, 4, 0, 5, 4, 1, 1, 0, 0, 0, 2, 2, 1, 3,
1, 4, 3, 0, 1, 3, 3, 4, 2, 3, 16, 3, 3, 6, 0, 1, 1, 0, 1, 1, 331, 1, 5, 1, 7, 1, 1, 1, 299, 3, 107, 9, 1, 5, 1, 3, 3, 1, 0,
2, 1, 3, 1, 0, 11, 3, 2, 3, 2, 3, 3, 30, 4, 6, 5, 4, 2, 2, 0, 2, 4, 1, 2, 1, 1, 2, 18, 153, 8, 3, 6, 2, 0, 4, 5, 31, 3, 2,
4, 4, 2, 2, 2, 5, 2, 9, 7, 3, 2, 14, 1, 2, 5, 129, 0, 7, 6, 3, 4, 2, 2, 1, 4, 0, 0, 1, 10, 21, 0, 26, 35, 66, 2, 2, 5, 1, 3,
1, 7, 0, 0, 3, 4, 6, 27, 6, 10, 6, 3, 14, 2, 1, 11, 10, 2, 3, 6, 10, 7, 39, 8, 3, 2, 19, 6, 0, 3, 3, 29, 4, 9, 0, 117, 3, 5,
0, 1, 1, 4, 5, 2, 14, 10, 0, 2, 2, 146, 5, 8, 4, 0, 0, 5, 2, 0, 39, 1, 2, 22, 3, 14, 1, 2, 12, 5, 78, 6, 2, 2, 0, 9, 20, 10,
2, 1, 2, 9, 93, 7, 10, 6, 3, 3, 5, 1, 0, 124, 0, 2, 5, 22, 15, 0, 2, 4, 2, 0, 2, 2, 17, 1, 3, 8, 5, 18, 1, 1, 47, 12, 17, 1,
0, 1, 30, 3, 4, 2, 25, 3, 9, 1, 3, 1, 5, 3, 3, 3, 1, 3, 0, 4, 0, 4, 3, 0, 25, 4, 15, 342, 2, 9, 2, 14, 10, 3, 1, 0, 2, 1, 0,
37, 2, 21, 5, 3, 1, 0, 1, 4, 4, 7, 33, 5, 7, 2, 1, 4, 23, 2, 14, 25, 7, 3, 5, 6, 15, 2, 6, 40, 12, 9, 4, 4, 7, 2, 5, 4, 0,
5, 0, 0, 0, 7, 4, 2, 8, 4, 3, 3, 6, 29, 0, 17, 2, 6, 2, 2, 3, 39, 37, 16, 3, 0, 8, 12, 6, 7, 6, 4, 30, 1, 0, 6, 3, 0, 11, 5
7, 10, 2, 4, 5, 5, 2, 14, 5, 330, 8, 5, 8, 4, 16, 0, 1, 2, 0, 0, 2, 4, 4, 2, 2, 6, 2, 82, 2, 6, 2, 5, 0, 3, 1, 0, 7, 17, 34,
0, 6, 22, 36, 1, 5, 51, 2, 2, 8, 161, 5, 12, 0, 0, 0, 3, 7, 9, 1, 6, 2, 210, 1, 20, 3, 160, 13, 0, 0, 3, 30, 1, 3, 1, 3, 4,
2, 5, 2, 247, 24, 5, 1, 19, 1, 2, 56, 2, 154, 16, 1, 8, 8, 3, 5, 213, 22, 79, 4, 2, 9, 64, 0, 3, 7, 0, 5, 3, 0, 0, 124, 2,
4, 4, 1, 8, 1, 3, 15, 27, 4, 6, 33, 116, 10, 1, 0, 32, 4, 1, 17, 2, 4, 2, 1, 17, 2, 6, 6, 0, 4, 3, 0, 8, 0, 4, 4, 2, 3, 2,
5, 20, 6, 2, 0, 3, 0, 1, 21, 8, 3, 1, 19, 2, 3, 5, 5, 1, 0, 2, 1, 38, 1, 2, 2, 4, 34, 3, 9, 5, 1, 4, 1, 3, 2, 3, 1, 1, 2, 1,
5, 31, 2, 15, 3, 0, 4, 5, 11, 1, 2, 2, 3, 36, 1, 4, 1, 51, 9, 11, 4, 2, 0, 0, 0, 10, 2, 2, 0, 0, 0, 0, 4, 3, 1, 2, 0, 0, 5,
9, 0, 3, 2, 0, 1, 5, 2, 4, 0, 2, 2, 6, 0, 2, 2, 2, 2, 9, 5, 0, 0, 2, 3, 20, 1, 0, 10, 17, 16, 3, 4, 1, 7, 0, 5, 2, 2, 0, 0,
2, 7, 3, 1, 3, 0, 1, 18, 5, 4, 2, 3, 7, 2, 4, 2, 2, 3, 19, 0, 2, 0, 2, 3, 8, 3, 1, 1, 5, 3, 1, 0, 1, 2, 1, 0, 5, 3, 0, 2, 5
1, 17, 5, 2, 8, 9, 20, 16, 1, 1, 47, 4, 7, 2, 0, 2, 5, 1, 1, 3, 1, 0, 2, 0, 0, 3, 0, 2, 3, 1, 3, 3, 2, 0, 1, 2, 2, 1, 1, 20,
3, 2, 1, 4, 5, 0, 0, 3, 25, 2, 2, 0, 4, 1, 4, 4, 2, 2, 2, 0, 7, 3, 0, 1, 2, 3, 0, 0, 0, 0, 2, 2, 41, 6, 0, 0, 9, 5, 3, 3, 6,
1, 2, 3, 2, 23, 3, 2, 1, 3, 4, 2, 5, 4, 1, 1, 1, 15, 1, 3, 2, 2, 12, 0, 5, 49, 3, 2, 1, 24, 17, 2, 5, 0, 0, 4, 2, 12, 51, 1
0, 13, 2, 2, 8, 2, 13, 1, 1, 2, 5, 4, 3, 8, 9, 3, 2, 5, 10, 3, 3, 2, 73, 9, 1, 2, 3, 5, 3, 0, 3, 5, 2, 4, 4, 71, 2, 7, 4, 2
9, 2, 0, 0, 3, 3, 7, 1, 1, 9, 1, 4, 1, 4, 2, 3, 20, 2, 2, 5, 60, 1, 2, 1, 3, 2, 19, 1, 2, 8, 10, 3, 2, 3, 0, 2, 42, 4, 2, 1
2, 6, 7, 5, 4, 1, 2, 2, 7, 8, 4, 2, 2, 6, 1, 7, 2, 13, 1, 2, 0, 0, 20, 11, 8, 3, 3, 3, 6, 1, 0, 0, 1, 0, 2, 2, 6, 29, 14, 3,
2, 4, 6, 5, 0, 16, 1, 5, 11, 2, 0, 9, 5, 4, 7, 197, 3, 3, 3, 1, 0, 2, 63, 5, 28, 3, 40, 1, 6, 5, 5, 2, 1, 0, 28, 2, 2, 83,
6, 2, 1, 2, 2, 4, 9, 1, 2, 4, 0, 2, 1, 16, 20, 0, 1, 2, 3, 6, 1, 6, 2, 3, 4, 2, 0, 0, 0, 1, 5, 1, 2, 4, 0, 1, 1, 16, 7, 2,
5, 0, 3, 11, 4, 0, 0, 79, 2, 0, 2, 18, 2, 1, 2, 1, 12, 3, 2, 3, 3, 1, 1, 2, 1, 4, 5, 2, 19, 4, 6, 3, 1, 0, 0, 0, 0, 0, 1, 2,
3, 1, 2, 1, 1, 10, 5, 2, 1, 2, 3, 3, 3, 0, 3, 7, 0, 4, 41, 3, 4, 3, 1, 17, 1, 1, 6, 1, 3, 4, 4, 2, 62, 4, 6, 1, 10, 21, 7,
5, 0, 3, 0, 2, 1, 2, 0, 4, 3, 1, 1, 4, 2, 1, 6, 61, 4, 2, 1, 1, 3, 9, 0, 0, 2, 0, 3, 1, 10, 3, 38, 0, 0, 3, 0, 5, 6, 21, 2,
17, 32, 2, 22, 253, 21, 467, 1, 7, 42, 0, 123, 2, 119, 8, 56, 32, 25, 77, 2, 10, 4, 124, 32, 1, 5, 9, 12, 12, 22, 1, 4, 2,
6, 3, 5, 0, 4, 4, 0, 3, 3, 4, 14, 11, 100, 18, 3, 10, 1, 1, 0, 0, 0, 0, 1, 20, 0, 660, 2, 0, 71, 2, 0, 7, 27, 2, 1, 2, 11, 4
1, 0, 0, 28, 6, 61, 13, 4, 3, 6, 1, 3, 9, 2, 5, 1, 6, 11, 4, 5, 3, 4, 1, 9, 3, 4, 5, 4, 11, 2, 6, 5, 16, 16, 0, 2, 5, 2, 3,
2, 4, 16, 2, 4, 20, 7, 0, 0, 1, 0, 2, 7, 0, 9, 0, 4, 2, 1, 2, 5, 2, 3, 5, 1, 1, 3, 1, 0, 4, 3, 0, 4, 0, 0, 15, 143, 2, 0, 0,
2, 1, 100, 0, 3, 5, 2, 4, 3, 2, 28, 1, 18, 7, 1, 5, 3, 6, 5, 3, 5, 0, 2, 0, 1, 1, 0, 2, 49, 2, 0, 2, 36, 3, 5, 1, 0, 0, 2,
3, 3, 1, 0, 4, 2, 0, 0, 1, 5, 0, 2, 1, 0, 0, 3, 3, 38, 2, 19, 10, 4, 6, 2, 4, 0, 1, 23, 7, 19, 93, 15, 2, 10, 0, 4, 88, 4,
4, 47, 17, 19, 6, 4, 58, 162, 3, 3, 2, 3, 3, 4, 0, 3, 6, 33, 10, 2, 0, 2, 6, 3, 1, 64, 2, 14, 33, 27, 3, 132, 7, 8, 145, 5,
65, 1, 1, 23, 14, 3, 38, 12, 2, 2, 45, 11, 13, 10, 6, 163, 4, 4, 5, 0, 1, 2, 0, 2, 19, 4, 5, 6, 140, 4, 10, 4, 5, 1, 0, 1,
5, 4, 2, 2, 12, 7, 6, 5, 3, 22, 5, 11, 50, 3, 14, 1, 23, 8, 5, 15, 2, 4, 11, 35, 0, 6, 4, 7, 2, 1, 2, 2, 5, 1, 0, 3, 0, 5,
1, 6, 13, 3, 4, 27, 3, 3, 9, 3, 37, 21, 9, 1, 6, 0, 48, 3, 13, 31, 3, 4, 3, 19, 3, 5, 1, 741, 195, 5, 14, 2, 2, 2, 4, 21, 0]
Theta Class 1: [3.55467084e-05 1.42186833e-04 3.55467084e-05 ... 1.42186833e-04
 7.46480876e-04 0.00000000e+00]
```

In [ ]:
```
#Display the class-conditional densities θjc1 and θjc2.
#Use the function plt.bar() to plot the bar plot of the class conditional densities for the first 10 features
#Write the value inside the bars
# plt.bar(range(len(theta_class0)), theta_class0, color='red', label='Spam', width=1)
plt.bar(range(len(theta_class1)), theta_class0, color='green', label='Ham', width=1)#put Njc_class0 to see the number of occu
plt.xlabel('Feature Index')
plt.ylabel('Probability of thetajc')
plt.title('Class Conditional Densities')
plt.legend()
plt.show()
```



In [ ]:
```
plt.bar(range(len(theta_class0)), theta_class0, color='red', label='Spam', width=1)
# plt.bar(range(len(theta_class1)), theta_class1, color='green', label='Ham', width=1)
plt.xlabel('Feature Index')
plt.ylabel('Probability of thetajc')
plt.title('Class Conditional Densities')
plt.legend()
plt.show()
```



Student's comments to Exercise 2

Add comments to the results of Exercise 2 here (may use LateX for formulas if needed).

Exercise 3 - Classification – discrete data

In this exercise, you will design a Naïve Bayes Classifier (NBC) for the Bag of Words (BoW) features for document classification that have been prepared in *Exercise 2*. In particular, in *exercise 2*, you have already estimated the following parameters:

- The prior probabilities of each class, $\pi_c = p(y = c)$.
- The class-conditional probabilities of each feature, $\theta_{jc} = p(x_j = 1|y = c)$.

These parameters have been estimated from the training data. In this exercise, you will use the test data, and classify each test vector using an NBC whose model has been fitted in *Exercise 2*. In particular, you will do the following:

- For each test vector, calculate the MAP estimate of the class the test vector belongs to. Remember: the MAP classifier chooses the class that maximizes $\max_c \log p(y = c|x) \propto \log p(x|y = c) + \log p(c)$. In the NBC, the features (i.e. each entry of x) are assumed to be statistically independent, so $p(x|y = c) = \prod_{j=1}^{D} p(x_j|y = c)$. This formula allows you to calculate $p(x|y = c)$ for a given test vector x using the parameters θ_{jc} already calculated in *Exercise 2*. Note that, after the logarithm, the product becomes a summation. It is much better to use the logarithm in order to avoid underflow.
- See how the accuracy changes when the prior is not taken into account (e.g. by comparing the MLE and MAP estimate).
- After classifying a test vector using the NBC, the obtained class can be compared with the truth (vector *ytest*).
- The accuracy of the classifier can be computed as the percentage of times that the NBC provides the correct class for a test vector.
- Repeat the same operations using the training data as test data, and compare the accuracy of the classifier on the training and test data.
- Note: It is expected that students implement the Naive Bayes classifier from scratch without using pre-made functions such as sklearn.naive_bayes

Optional:

If you plot the class-conditional densities as done at the end of Exercise 2, you will see that many features are uninformative; e.g., words that appear very often (or very rarely) in documents belonging to either class are not very helpful to classify a document. The NBC can perform a lot better if these uninformative features are disregarded during the classification, i.e. only a subset of the features, chosen among the most informative ones, are retained. To rank the features by "significance", one can employ the mutual information between feature x_j and class y (see Sec. 3.5.4 of the textbook):

$$I(X, Y) = \sum_{xj} \sum_{y} p(x_j, y) \log \frac{p(x_j, y)}{p(x_j)p(y)}$$

For binary features, the mutual information of feature j can be written as:

$$I_j = \sum_{c} \left[\theta_{jc}\pi_c \log \frac{\theta_{jc}}{\theta_j} + (1 - \theta_{jc})\pi_c \log \frac{1 - \theta_{jc}}{1 - \theta_j} \right]$$

with $\theta_j = p(x_j = 1) = \sum_c \pi_c \theta_{jc}$. For this part, you should:

- Calculate I_j for all features. Note: try to avoid divisions by zero adding the machine precision constant *eps* to the denominators.
- Rank the features by decreasing values of I_j, and keep only the K most important ones.
- Run the classifier employing only the K most important features, and calculate the accuracy.
- Plot the accuracy as a function of K.

In [ ]:
```
#Evaluate the MAP on the test set
# ham is 0 and spam is 1
MAP_Sentence0 = []
MAP_Sentence1 = []
MAP_Sentence_i_0 = 0
MAP_Sentence_i_1 = 0
for i in range(len(X_test)):
    Pxj0 = 1
    Pxj1 = 1
    for j in range(len(X_test[i])):
        if X_test[i][j] == 1:
            if theta_class0[j] != 0: # this is because we have not done the add-one smoothing
                Pxj0 *= theta_class0[j]
            if theta_class1[j] != 0:
                Pxj1 *= theta_class1[j]
    MAP_Sentence_i_0 = np.log(prior_ham) + np.log(Pxj0) # this is MAP estimation since we are using the log(prior) + log(like
    MAP_Sentence_i_1 = np.log(prior_spam) + np.log(Pxj1)
    MAP_Sentence0.append(MAP_Sentence_i_0)
    MAP_Sentence1.append(MAP_Sentence_i_1)

y_pred = []
for i in range(len(MAP_Sentence0)):
    if abs(MAP_Sentence0[i]) > abs(MAP_Sentence1[i]):
        y_pred.append(0)
    else:
        y_pred.append(1)

y_pred = np.array(y_pred)
y_test = y_test.astype(int)
y_pred = y_pred.astype(int)
accuracy = np.sum(y_pred == y_test) / len(y_test)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8615863141524106
```

In [ ]:
```
#Evaluate the MAP on the test set without prior probabilities
MAP_Sentence0 = []
MAP_Sentence1 = []
MAP_Sentence_i_0 = 0
MAP_Sentence_i_1 = 0
for i in range(len(X_test)):
    Pxj0 = 1
    Pxj1 = 1
    for j in range(len(X_test[i])):
        if X_test[i][j] == 1:
            if theta_class0[j] != 0:
                Pxj0 *= theta_class0[j]
            if theta_class1[j] != 0:
                Pxj1 *= theta_class1[j]
    MAP_Sentence_i_0 =  np.log(Pxj0)
    MAP_Sentence_i_1 =  np.log(Pxj1)
    MAP_Sentence0.append(MAP_Sentence_i_0)
    MAP_Sentence1.append(MAP_Sentence_i_1)

y_pred = []
for i in range(len(MAP_Sentence0)):
    if abs(MAP_Sentence0[i]) > abs(MAP_Sentence1[i]):
        y_pred.append(0)
    else:
        y_pred.append(1)

y_pred = np.array(y_pred)
y_test = y_test.astype(int)
y_pred = y_pred.astype(int)
accuracy = np.sum(y_pred == y_test) / len(y_test)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8409797822706065
```

In [ ]:
```
#Evaluate the MAP on Training Set
MAP_Sentence0 = []
MAP_Sentence1 = []
MAP_Sentence_i_0 = 0
MAP_Sentence_i_1 = 0
for i in range(len(X_train)):
    Pxj0 = 1
    Pxj1 = 1
    for j in range(len(X_train[i])):
        if X_train[i][j] == 1:
            if theta_class0[j] != 0:
                Pxj0 *= theta_class0[j]
            if theta_class1[j] != 0:
                Pxj1 *= theta_class1[j]
    MAP_Sentence_i_0 = np.log(prior_ham) + np.log(Pxj0)
    MAP_Sentence_i_1 = np.log(prior_spam) + np.log(Pxj1)
    MAP_Sentence0.append(MAP_Sentence_i_0)
    MAP_Sentence1.append(MAP_Sentence_i_1)

y_pred = []
for i in range(len(MAP_Sentence0)):
    if abs(MAP_Sentence0[i]) > abs(MAP_Sentence1[i]):
        y_pred.append(0)
    else:
        y_pred.append(1)
```

```
y_pred = np.array(y_pred)
y_train = y_train.astype(int)
y_pred = y_pred.astype(int)
accuracy = np.sum(y_pred == y_train) / len(y_train)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8826666666666667
```

In [ ]:
```
#Evaluate the MLE on the test set
#MAP withou prior probabilities is the same as MLE
MLE_Sentence0 = []
MLE_Sentence1 = []
MLE_Sentence_i_0 = 0
MLE_Sentence_i_1 = 0

for i in range(len(X_test)):
    Pxj0 = 1
    Pxj1 = 1
    for j in range(len(X_test[i])):
        if X_test[i][j] == 1:
            if theta_class0[j] != 0:
                Pxj0 *= theta_class0[j]
            if theta_class1[j] != 0:
                Pxj1 *= theta_class1[j]
    MLE_Sentence_i_0 = np.log(Pxj0)
    MLE_Sentence_i_1 = np.log(Pxj1)
    MLE_Sentence0.append(MLE_Sentence_i_0)
    MLE_Sentence1.append(MLE_Sentence_i_1)

y_pred = []
for i in range(len(MLE_Sentence0)):
    if abs(MLE_Sentence0[i]) <= abs(MLE_Sentence1[i]):
        y_pred.append(1)
    else:
        y_pred.append(0)

y_pred = np.array(y_pred)
y_test = y_test.astype(int)
y_pred = y_pred.astype(int)
accuracy = np.sum(y_pred == y_test) / len(y_test)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8409797822706065
```

In [ ]:
```
#Evaluate the MLE on the trianing set
#MAP withou prior probabilities is the same as MLE
MLE_Sentence0 = []
MLE_Sentence1 = []
MLE_Sentence_i_0 = 0
MLE_Sentence_i_1 = 0

for i in range(len(X_train)):
    Pxj0 = 1
    Pxj1 = 1
    for j in range(len(X_train[i])):
        if X_train[i][j] == 1:
            if theta_class0[j] != 0:
                Pxj0 *= theta_class0[j]
            if theta_class1[j] != 0:
                Pxj1 *= theta_class1[j]
    MLE_Sentence_i_0 = np.log(Pxj0)
    MLE_Sentence_i_1 = np.log(Pxj1)
    MLE_Sentence0.append(MLE_Sentence_i_0)
    MLE_Sentence1.append(MLE_Sentence_i_1)

y_pred = []
for i in range(len(MLE_Sentence0)):
    if abs(MLE_Sentence0[i]) <= abs(MLE_Sentence1[i]):
        y_pred.append(1)
    else:
        y_pred.append(0)

y_pred = np.array(y_pred)
y_train = y_train.astype(int)
y_pred = y_pred.astype(int)
accuracy = np.sum(y_pred == y_train) / len(y_train)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8686666666666667
```

Student's comments to Exercise 3

Add comments to the results of Exercise 3 here (may use LateX for formulas if needed).

Exercise 4 – Plotting the ROC curve

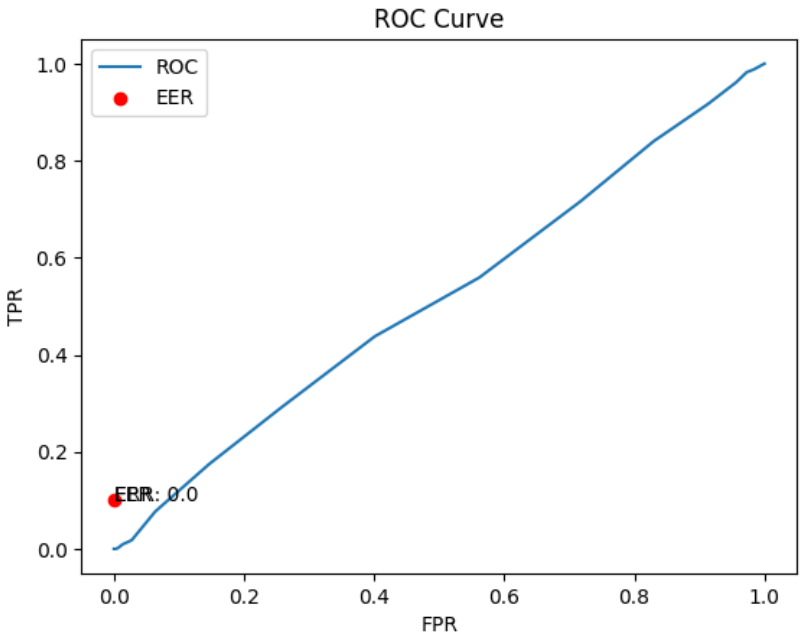
For the discrete data classification problem of Exercise 3, analyse the performance of the classifier plotting the complete ROC curve, instead of simply measuring the accuracy. This requires to do the following.

- Instead of classifying the documents choosing class 1 if $p(y=1|x) > p(y=2|x)$, now you can generalize this to choosing class 1 if $\frac{p(y=1|x)}{p(y=2|x)} > \tau \in [0, \infty)$ for some threshold τ that determines the compromise between true positive rate (TPR) and false positive rate (FPR).
- Choose a **reasonable** range of values for τ. For each value of τ, compute the TPR and FPR on the dataset (Hint: determine suitable minimum and maximum values for τ, and sample densely enough in that range).

- Plot a curve of TPR as a function of FPR – this is the ROC curve for this classifier.
- Determine an estimate of the Equal Error Rate (EER), i.e. the point of the ROC curve such that TPR+FPR=1.

In [ ]:

```
# #compute the ROC curve
def ROCEstimator(mapClass0, mapClass1, taw):
    y_pred_test_roc = []
    for i in range(len(mapClass0)):
        if np.abs(mapClass0[i]) / np.abs(mapClass1[i]) > taw:
            y_pred_test_roc.append(0)
        elif np.abs(mapClass0[i]) / np.abs(mapClass1[i]) < taw:
            y_pred_test_roc.append(1)
    return y_pred_test_roc

#compute accuracy, TPR, FPR and EER

def Accuracy(y, y_pred):
    return np.sum(y == y_pred) / len(y)

def TPR(y, y_pred):
    nTrue = 0
    nPos = 0
    for i in range(len(y)):
        if y[i] == 0 :
            nTrue += 1
            if y_pred[i] == 0:
                nPos += 1
    TPR = nPos / nTrue
    return TPR

def FPR(y, y_pred):
    nFalse = 0
    nNeg = 0
    for i in range(len(y)):
        if y[i] == 1 :
            nFalse += 1
            if y_pred[i] == 0:
                nNeg += 1
    FPR = nNeg / nFalse
    return FPR

def FNR(y, y_pred):
    nTrue = 0
    nNeg = 0
    for i in range(len(y)):
        if y[i] == 1 :
            nTrue += 1
            if y_pred[i] == 1:
                nNeg += 1
    FNR = nNeg / nTrue
    return FNR

taws = np.arange(0,15,0.1)
TPRs = []
FPRs = []
FNRs = []
ERRs = []
ERR_Points = []

for i in taws:
    y_pred_test_roc = ROCEstimator(MAP_Sentence0, MAP_Sentence1, i)
    TPRr = TPR(y_test, y_pred_test_roc)
    FPRr = FPR(y_test, y_pred_test_roc)
    FNRr = FNR(y_test, y_pred_test_roc)
    TPRs.append(TPRr)
    FPRs.append(FPRr)
    # if abs(FPRr[i] - (1 - TPRr[i])) < 1e-6:
    #     eer = (FPRr[i] + (1 - TPRr[i])) / 2
    #     ERRs.append(eer)
    #     ERR_Points.append(i)
    #find ERR Points where tpr_fp = 1
    if FNRr - FPRr < 0.05:
        ERRs.append(i)
        ERR_Points.append(i)


#plot the ROC curve
#show the point corresponding to the EER on the ROC curve
#show the value of ERR on the plot
plt.plot(FPRs, TPRs, label='ROC')
plt.scatter(ERR_Points[0], ERR_Points[1], color='red', label='EER')
plt.text(ERR_Points[0], ERR_Points[1], 'EER')
plt.text(ERR_Points[0], ERR_Points[1], 'ERR: ' + str(ERRs[0]))
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC Curve')
plt.legend()
plt.show()
```



Student's comments to Exercise 4

Add comments to the results of Exercise 4 here (may use LateX for formulas if needed).

Exercise 5 – Classification – continuous data

This exercise employs the Iris dataset already employed in Exercise 1, and performs model fitting and classification using several versions of **Gaussian discriminative analysis**. However, for this exercise the available data have to be divided into two sets, namely *training* and *test* data.

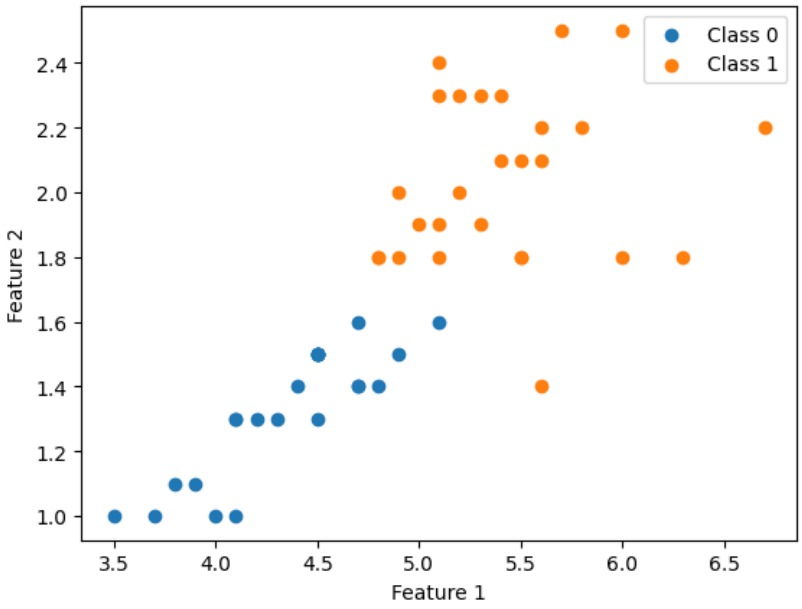
You will have to 1) re-fit the training data to the specific model (see below), 2) classify each of the test samples, and 3) calculate the accuracy of each classifier.

Classifiers to be employed:

- Two-class quadratic discriminant analysis (fitting: both mean values and covariance matrices are class-specific – same as in exercise 1).
- Two-class linear discriminant analysis (fitting: class-specific mean values as in the previous case. Shared covariance matrix is calculated putting together the elements of both classes; the mean values should also be recalculated accordingly).

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import h5py

Dataset1 = h5py.File("Lab2_Ex_1_Iris.hdf5")
Data = np.array(Dataset1.get('Dataset'))

Train = Data[:50,:]
Test = Data[50:,:]
```

In [ ]:
```
#create training and test set
X_train = Train[:,:-1]
y_train = Train[:,-1]
X_test = Test[:,:-1]
y_test = Test[:,-1]

#Separate the dataset in the two classes, you can use the numpy function argsort and unique to do this.
class0 = X_train[y_train == 0]
class1 = X_train[y_train == 1]

#Draw the scatter plot of the two classes on the same image
plt.scatter(class0[:,0], class0[:,1], label='Class 0')
plt.scatter(class1[:,0], class1[:,1], label='Class 1')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

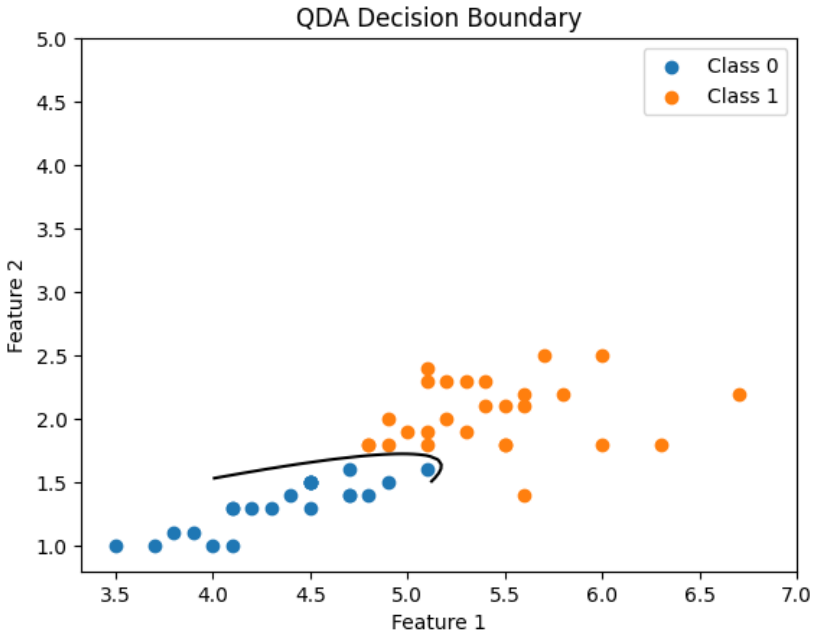


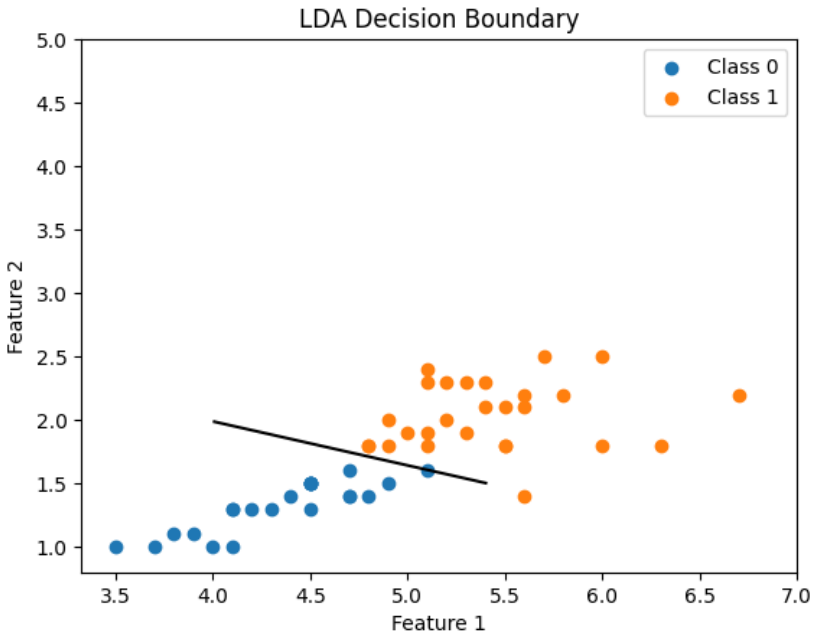
In [ ]:

```
#calculate the mean and covariance matrix for each class
mean_class0 = np.mean(class0, axis=0)
mean_class1 = np.mean(class1, axis=0)
cov_class0 =  np.matmul(np.transpose(class0 - class0.mean(axis=0)), class0 - class0.mean(axis=0)) / class0.shape[0]
cov_class1 =  np.matmul(np.transpose(class1 - class1.mean(axis=0)), class1 - class1.mean(axis=0)) / class1.shape[0]

# cov_class0 = np.cov(class0.T)
# cov_class1 = np.cov(class1.T)

#quadradic discriminant Analysis (QDA) Function
# this is the formula for QDA: with log: -0.5 * log(det(cov)) - 0.5 * (x - mean) @ inv(cov) @ (x - mean) - x.shape[0] / 2 *
def QDA(x, mean, cov):
    return -0.5 * np.log(np.linalg.det(cov)) - 0.5 * (x - mean) @ np.linalg.inv(cov) @ (x - mean) - x.shape[0] / 2 * np.log(

#predict the class of the test set
y_pred = []
for i in range(len(X_test)):
    if QDA(X_test[i], mean_class0, cov_class0) > QDA(X_test[i], mean_class1, cov_class1):
        y_pred.append(0)
    else:
        y_pred.append(1)

#compute the accuracy
accuracy = np.sum(y_pred == y_test) / len(y_test)
print('QDA Accuracy:', accuracy)

#using Linear Discriminant Analysis (LDA) Function
def LDA(x, mean, cov):
    return x @ np.linalg.inv(cov) @ mean - 0.5 * mean @ np.linalg.inv(cov) @ mean + np.log(0.5)

#predict the class of the test set
ldaMean = np.mean(X_train, axis=0)
ldaCov = np.cov(X_train.T)

y_pred = []
for i in range(len(X_test)):
    if LDA(X_test[i], mean_class0, ldaCov) > LDA(X_test[i], mean_class1, ldaCov):
        y_pred.append(0)
    else:
        y_pred.append(1)

#compute the accuracy
accuracy = np.sum(y_pred == y_test) / len(y_test)
print('LDA Accuracy:', accuracy)

import matplotlib.pyplot as plt
import numpy as np

# Assuming class0 and class1 are lists
class0 = np.array(class0)
class1 = np.array(class1)


# Plot the decision boundary for QDA
# Scatter plot of the two classes
plt.scatter(class0[:, 0], class0[:, 1], label='Class 0')
plt.scatter(class1[:, 0], class1[:, 1], label='Class 1')

# Assuming mean_class0, cov_class0, mean_class1, cov_class1 are also lists
mean_class0 = np.array(mean_class0)
cov_class0 = np.array(cov_class0)
mean_class1 = np.array(mean_class1)
cov_class1 = np.array(cov_class1)

x = np.linspace(4, 7, 100)
y = np.linspace(1.5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = np.zeros(X.shape)
```

```
for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        point = np.array([X[i, j], Y[i, j]])
        Z[i, j] = QDA(point, mean_class0, cov_class0) - QDA(point, mean_class1, cov_class1)

plt.contour(X, Y, Z, levels=0, colors='black')

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.title('QDA Decision Boundary')
plt.show()
plt.clf()

#plot the scatter plot of the two classes
plt.scatter(class0[:,0], class0[:,1], label='Class 0')
plt.scatter(class1[:,0], class1[:,1], label='Class 1')



#plot the decision boundary for LDA
x = np.linspace(4, 7, 100)
y = np.linspace(1.5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = np.zeros(X.shape)
for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        Z[i,j] = LDA([X[i,j], Y[i,j]], mean_class0, ldaCov) - LDA([X[i,j], Y[i,j]], mean_class1, ldaCov)
plt.contour(X, Y, Z, levels=0, colors='black')

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.title('LDA Decision Boundary')
plt.show()
```

```
QDA Accuracy: 0.94
LDA Accuracy: 0.9
```





In [ ]:

Student's comments to Exercise 5

Add comments to the results of Exercise 5 here (may use LateX for formulas if needed).

Exercise 6 – Classification – continuous data

Classify the data in the phoneme dataset from Lab. 1 using quadratic discriminant analysis, linear discriminat analysis and a Naive Bayes classifier.

Compute the accuracy of each classifier and compare its performance with that of the k-nn classifier developed in Lab. 1.

Note: in Lab1 we had to employ a subset of the oiginal dataset due to the fact that k-nn has a quadratic complexity making it unfit for use on large datasets. The algorithms illustrated in this Lab have smaller complexity and thus it is possible to train on more data.

For this exercise you can use the sklearn library.

In [ ]:
```
import numpy as np
import h5py
from scipy.stats import multivariate_normal

Dataset2 = h5py.File("Lab2_Ex_6_phoneme.hdf5")
Data = np.array(Dataset2.get('Dataset'))

Train = Data[:4000,:]
Test = Data[4000:,:]
len_dat = np.shape(Test)[0]

from sklearn import metrics
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB

#Part 1
clf = LinearDiscriminantAnalysis()
clf.fit(Train[:,:-1], Train[:,-1])
y_pred = clf.predict(Test[:,:-1])
accuracy = metrics.accuracy_score(Test[:,-1], y_pred)
print('LDA Accuracy:', accuracy)
#Complete here

#Part 2
clf2 = QuadraticDiscriminantAnalysis()
clf2.fit(Train[:,:-1], Train[:,-1])
y_pred = clf2.predict(Test[:,:-1])
accuracy = metrics.accuracy_score(Test[:,-1], y_pred)
print('QDA Accuracy:', accuracy)
#Complete here

#Part 3
clf3 = GaussianNB()
clf3.fit(Train[:,:-1], Train[:,-1])
y_pred = clf3.predict(Test[:,:-1])
accuracy = metrics.accuracy_score(Test[:,-1], y_pred)
print('Naive Bayes Accuracy:', accuracy)
#Complete here
```

```
LDA Accuracy: 0.9410609037328095
QDA Accuracy: 0.888015717092338
Naive Bayes Accuracy: 0.9017681728880157
```

Student's comments to Exercise 6

Add comments to the results of Exercise 6 here (may use LateX for formulas if needed).

STATISTICAL LEARNING AND NEURAL NETWORKS, A.A. 2022/2023

COMPUTER LAB 3 - Principal component analysis

Duration: 3 hours

Introduction:

Hyperspectral images are scientific images of the Earth, acquired by satellites or aicrafts; rather than having three R/G/B color channels, these images have a lot more "color" components obtained through a fine sampling of the wavelength (hence the name "hyper"-spectral). The resulting 3-dimensional dataset has one image (spectral band or "color") for every sampled wavelength, which represents the measured radiance from each pixel at that specific wavelength. Hyperspectral images are very useful for image analysis. For every pixel at a given spatial position, it is possible to extract a so-called spectral vector, i.e. the 1-dimensional vector of values assumed by that pixel at all wavelengths. Assuming that each pixel is composed of just one substance, the spectral vector represents the radiance of that substance at all the wavelengths that have been sampled. Spectral vectors, therefore, can be used to infer which substance is contained in a given pixel – a typical classification problem that has a lot of practical applications in agriculture, analysis of land use / land cover, and other applications related to the study of the environment.

In this lab you will use a real hyperspectral image that has been acquired by the AVIRIS instrument, an airborne hyperspectral imager operated by the NASA. The image represents a scene of Indian Pines (Indiana, USA). It has a size of 145x145 pixels and 220 spectral bands. Along with the image, a ground truth is available, in terms of labels specifying which class (out of 16) each pixel belongs to. The classes are reported below; for more information, please see http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes#Indian_Pines

#	Class	Samples
1	Alfalfa	46
2	Corn-notill	1428
3	Corn-mintill	830
4	Corn	237
5	Grass-pasture	483
6	Grass-trees	730
7	Grass-pasture-mowed	28
8	Hay-windrowed	478
9	Oats	20
10	Soybean-notill	972
11	Soybean-mintill	2455
12	Soybean-clean	593
13	Wheat	205
14	Woods	1265
15	Buildings-Grass-Trees-Drives	386
16	Stone-Steel-Towers	93

The purpose of this computer lab is twofold: To apply PCA to the spectral vectors in order to reduce their dimensionality.

- To apply PCA to the spectral vectors in order to reduce their dimensionality.
- To perform classification on the reduced data (optional)

Exercise 1 – PCA

In this exercise, you will employ the Indian Pines dataset. You will not do this for the entire dataset, but only for the spectral vectors belonging to **two classes** (as in the optional exercise you will perform 2-class classification on the PCA coefficients).

Reminder: the input to the PCA must always have zero mean: besides the sample covariance, you will have to compute the **mean value μ** over the training set and subtract it from each test vector before applying PCA.

Task: You have to reduce the dimensionality of the spectral vectors of the two classes you have chosen using PCA. In particular, you should perform the following:

- Extract spectral vectors of two classes, as described above (see sample code below).
- Estimate the sample covariance matrix of the dataset as a whole (i.e., considering together spectral vectors of the two classes)
- Perform the eigenvector decomposition of the sample covariance matrix. You can use the numpy linalg.eig function, which outputs the matrix containing the eigenvectors as columns, and a diagonal matric containing the eigenvalues on the main diagonal.
  - Note: in the output matrix, eigenvectors/eigenvalues are not necessarily ordered by eigenvalue magnitude. You should sort them by yourself.
- Choose a number of dimensions K<=220.
- Construct the eigenvector matrix W for K components (i.e., select the last K columns)
- Using W, compute the PCA coefficients for each spectral vector in the data set
- Then from the PCA coefficients obtain an approximation of the corresponding vector and compute the error (mean square error - MSE)
- Plot the average MSE over the test set as a function of K.
- Plot the eigenvectors corresponding to the 3 largest eigenvalues – this will give you an idea of the basis functions employed by PCA

In [ ]:
```
import numpy as np
import h5py
import scipy.io

mat = scipy.io.loadmat('Indian_pines.mat')
indian_pines = np.array(mat['indian_pines']) # For
# print(indian_pines.shape) 145x145x220
# print(len(indian_pines[0])) 145
# print(indian_pines[0][0]) # 220

mat = scipy.io.loadmat('Indian_pines_gt.mat')
indian_pines_gt = np.array(mat['indian_pines_gt'])
# print(indian_pines_gt.shape) 145x145
# print(len(indian_pines_gt[0])) 145
```

In [ ]:
```
indian_pines_gt.shape

# print(indian_pines_gt[0][0])
# print(indian_pines)
# print(indian_pines_gt[1])
# print(indian_pines.shape)
```

Out[ ]:
```
(145, 145)
```

Extract spectral vectors of two classes, as described above (see sample code below).

In [ ]:
```
class1_value = 4 # 4 is the value of the class 1 in the ground truth which is the corn field

class1=np.zeros((1500,220))
n=0
for i in range(145):
 for j in range(145):
   if indian_pines_gt[i,j]== class1_value:
     class1[n,:] = indian_pines[i,j,:]/1.
     n=n+1
class1=class1[:n,:]

#choosing the second class which is the woods and value is 14
class2_value = 14
class2=np.zeros((1500,220))
n=0
for i in range(145):
  for j in range(145):
    if indian_pines_gt[i,j]== class2_value:
      class2[n,:] = indian_pines[i,j,:]/1.
      n=n+1
class2=class2[:n,:]

#print (class1.shape) # 227x220
#print (class2.shape) # 1265x220
```

Estimate the sample covariance matrix of the dataset as a whole (i.e., considering together spectral vectors of the two classes)

In [ ]:
```
# #1. Compute the mean of the data
# #2. Compute the covariance matrix of the data
# #3. Compute the inverse of the covariance matrix
# #4. Compute the determinant of the covariance matrix
# #5. Compute the Mahalanobis distance between the mean of the data and the data points
# #6. Compute the threshold value for the Mahalanobis distance
# #7. Classify the data points based on the threshold value
# #8. Compute the confusion matrix
# #9. Compute the overall accuracy

# #1. Compute the mean of the data
# mean1 = np.mean(class1, axis=0)
# mean2 = np.mean(class2, axis=0)

# #2. Compute the covariance matrix of the data
# cov1 = np.cov(class1, rowvar=False)
# cov2 = np.cov(class2, rowvar=False)

# #3. Compute the inverse of the covariance matrix
# cov1_inv = np.linalg.inv(cov1)
# cov2_inv = np.linalg.inv(cov2)

# #4. Compute the determinant of the covariance matrix
# cov1_det = np.linalg.det(cov1)
# cov2_det = np.linalg.det(cov2)

# #5. Compute the Mahalanobis distance between the mean of the data and the data points
# def mahalanobis_distance(mean, cov_inv, cov_det, data):
#     data = data - mean
#     data_transpose = np.transpose(data)
#     data = np.dot(data, cov_inv)
#     data = np.dot(data, data_transpose)
#     data = np.sqrt(data)
#     data = data / np.sqrt(cov_det)
#     return data

# mahalanobis_distance1 = mahalanobis_distance(mean1, cov1_inv, cov1_det, class1)
# mahalanobis_distance2 = mahalanobis_distance(mean2, cov2_inv, cov2_det, class2)

# #6. Compute the threshold value for the Mahalanobis distance
```

```
# threshold = 0.5 * (np.mean(mahalanobis_distance1) + np.mean(mahalanobis_distance2))

# #7. Classify the data points based on the threshold value
# def classify_data_points(mahalanobis_distance, threshold):
#     return mahalanobis_distance < threshold

# class1_classification = classify_data_points(mahalanobis_distance1, threshold)
# class2_classification = classify_data_points(mahalanobis_distance2, threshold)

# #8. Compute the confusion matrix
# def confusion_matrix(class1_classification, class2_classification):
#     TP = np.sum(class1_classification)
#     FP = np.sum(class2_classification)
#     FN = len(class1_classification) - TP
#     TN = len(class2_classification) - FP
#     return TP, FP, FN, TN

# TP, FP, FN, TN = confusion_matrix(class1_classification, class2_classification)

# #9. Compute the overall accuracy
# overall_accuracy = (TP + TN) / (TP + FP + FN + TN)

# print("Overall Accuracy: ", overall_accuracy)
```

In [ ]:
```
import pandas as pd

#Insert code here
both_classes = np.concatenate((class1, class2), axis=0)
bothDf = pd.DataFrame(both_classes)
#shuffle the data
bothDf = bothDf.sample(frac=1)
#split the data into training and testing
bothClass_trainDF = bothDf.iloc[:1000, :]
bothClass_testDF = bothDf.iloc[1000:, :]

bothClassTest = bothClass_testDF.to_numpy()
bothClassTrain = bothClass_trainDF.to_numpy()

#normalizing the data
meanBothClassDF = bothClass_trainDF.mean()
meanBothClass = meanBothClassDF.to_numpy()
bothClassTrain -= meanBothClass
bothClassTest -= meanBothClass
bothClassTrain = bothClassTrain.T
bothClassTest = bothClassTest.T

covBothClassTrain = np.cov(bothClassTrain)
```

In [ ]:
```
print(covBothClassTrain.shape)
```

```
(220, 220)
```

Perform the eigenvector decomposition of the sample covariance matrix. You can use the numpy linalg.eig function, which outputs the matrix containing the eigenvectors as columns, and a diagonal matric containing the eigenvalues on the main diagonal.

In [ ]:
```
from numpy import linalg as LA

w, v = LA.eig(covBothClassTrain)
w = np.array([w])
#Note: in the output matrix, eigenvectors/eigenvalues are not necessarily ordered by eigenvalue magnitude. You should sort th
#sorting the eigenvalues and eigenvectors
v_sorting = np.concatenate((v, w), axis=0)
v_sorting_df = pd.DataFrame(v_sorting)
v_sorting_df_sorted = v_sorting_df.sort_values(by=220, axis=1, ascending=False)
v_sorting_sorted = v_sorting_df_sorted.to_numpy()
v_sorted = v_sorting_sorted[:220, :]
```

In [ ]:
```
# print(v_sorted.shape)
# print(v_sorting_sorted[0])
```

Choose a number of dimensions K<=220.Construct the eigenvector matrix W for K components (i.e., select the last K columns)

In [ ]:
```
#Insert code here
#select the first 10 eigenvectors
vNew = v_sorted[:, :50]
vNewTranspose = vNew.T

print(vNew.shape) # 220x50
print(vNewTranspose.shape) # 50x220
```

```
(220, 50)
(50, 220)
```

Using W, compute the PCA coefficients for each spectral vector in the data set. Then from the PCA coefficients obtain an approximation of the corresponding vector and compute the error (mean square error - MSE)

In [ ]:
```
#Note: remember to remove the mean from the vectors of the dataset
z = vNewTranspose @ bothClassTest
hat_bothClass = vNew @ z

# calculate MSE
```

```
SE = 0
for i in range(hat_bothClass.shape[1]):
  a = np.subtract(hat_bothClass[:,i], bothClassTest[:,i])
  SE = a @ a.T
  SE+=SE
MSE = SE/hat_bothClass.shape[1]
print(MSE)
```

```
182.92100868910816
```

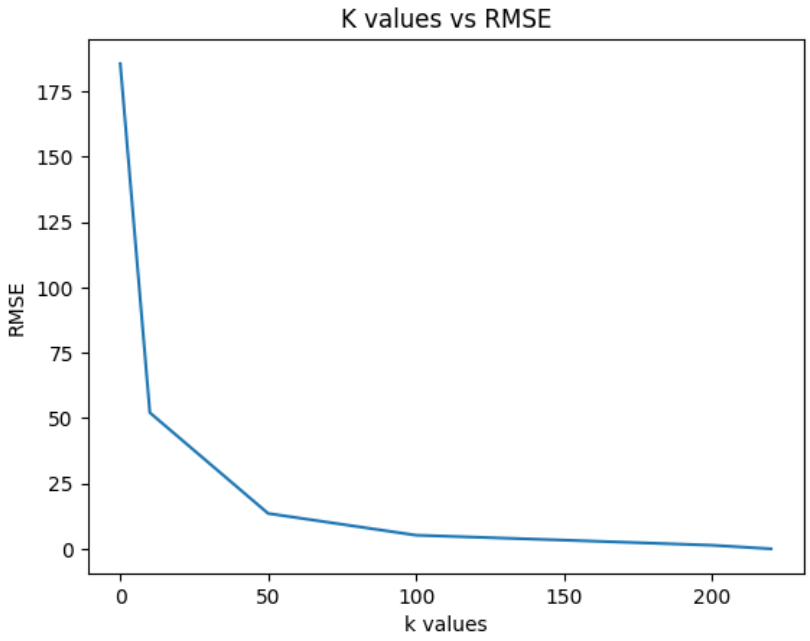
In [ ]:
```
bothClassTest.shape
```

Out[ ]:
```
(220, 502)
```

Plot the average MSE over the test set as a function of K.

In [ ]:
```
import matplotlib.pyplot as plt
#Insert code here
myZ = []
def MSE(a):
  k = a
  v_new = v_sorted[:, :k]
  vNew_Transpose = v_new.T
  z = vNew_Transpose @ bothClassTest
  myZ.append(z)
  hat_bothClass = v_new @ z
  SE = 0
  for i in range(hat_bothClass.shape[1]):
    a = np.subtract(hat_bothClass[:,i], bothClassTest[:,i])
    SE = a @ a.T
    SE+=SE
  MSE = SE/hat_bothClass.shape[1]
  RMSE = np.sqrt(MSE)
  return RMSE

kValues = [0,10,50,100,150,200,220]
RMSEs = []
for i in kValues:
  RMSEs.append(MSE(i))
print(RMSEs)


plt.plot(kValues, RMSEs)
plt.xlabel('k values')
plt.ylabel('RMSE')
plt.title('K values vs RMSE')
plt.show()
```

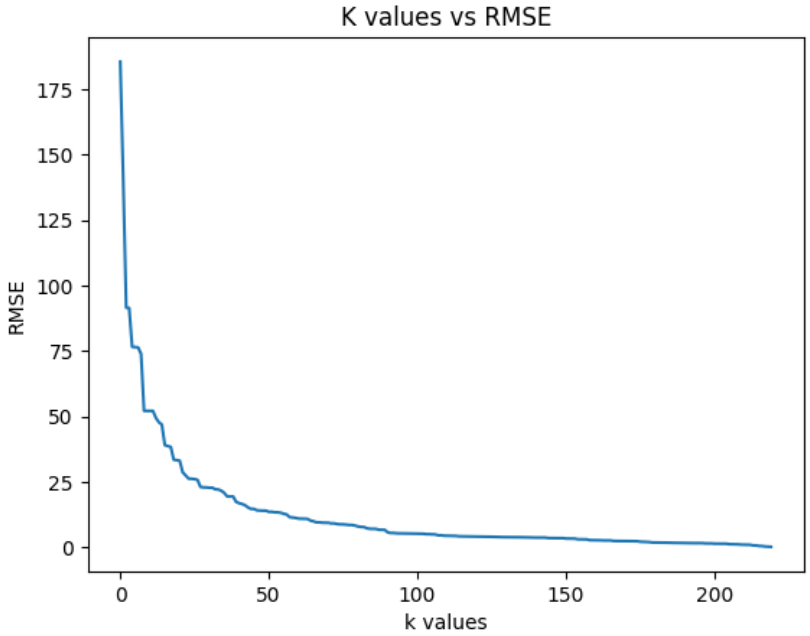
```
[185.63350795373972, 52.092308986800326, 13.52482934047998, 5.20616600201366, 3.3190329959496383, 1.3607582488095031, 2.8933
776355776e-10]
```



Plot the eigenvectors corresponding to the 3 largest eigenvalues – this will give you an idea of the basis functions employed by PCA

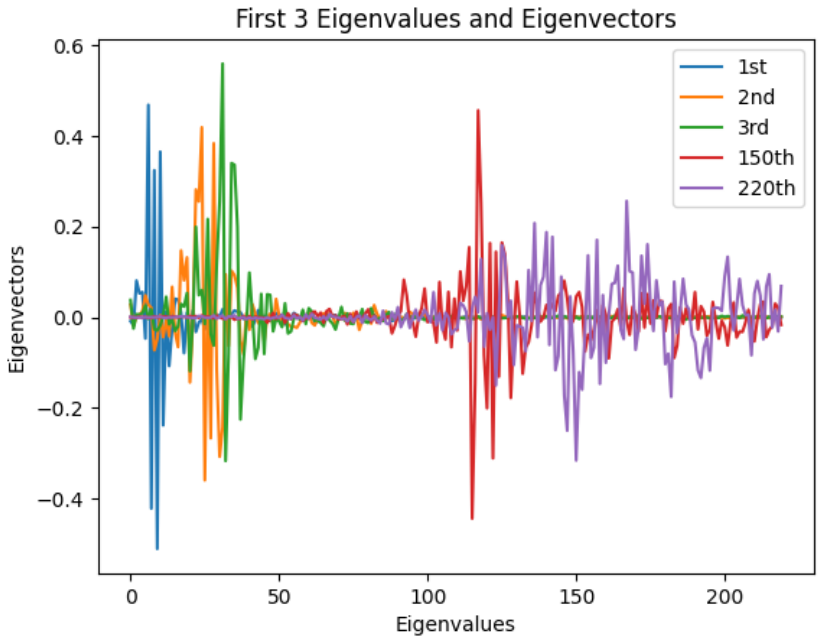
In [ ]:
```
#Insert code here
RMSEK = []
for i in range(len(hat_bothClass)):
  RMSEK.append(MSE(i))
print(RMSEK)

kValues = [i for i in range(220)]
plt.plot(kValues, RMSEK)
plt.xlabel('k values')
plt.ylabel('RMSE')
plt.title('K values vs RMSE')
plt.show()
```

```
[185.63350795373972, 140.59127130194943, 91.55560290556663, 91.46221496374893, 76.5975185325978, 76.49497672496888, 76.39384
245694988, 73.90649373717905, 52.12697861941187, 52.126920965440625, 52.092308986800326, 52.092036533054504, 49.425858764086
34, 47.763787561838086, 46.861791367706104, 39.030412551969874, 38.67955799655896, 38.328279362538424, 33.45464308564907, 3
3.289183292867364, 33.08434974033851, 28.68005713253948, 27.419108200208765, 26.220798258067795, 26.15979591260632, 26.02655
399440744, 25.748159474146817, 23.041406245665975, 22.88472533733759, 22.865661542577417, 22.747387946197602, 22.74604446016
3063, 22.128359550742726, 22.06387480450158, 21.605828328221456, 20.744271600640026, 19.388891799591168, 19.3691086170469, 1
9.369102281963357, 17.331541414336765, 16.83259667943201, 16.53106822899361, 16.035649905805204, 15.202249189283625, 14.6417
32147303363, 14.633108347173804, 14.09646120334366, 14.009122370063906, 13.977640469324763, 13.97700244146275, 13.5248293404
7998, 13.522235811892743, 13.338235332439895, 13.324858092076775, 13.160615490910994, 12.80135810777848, 12.506900856913989,
11.507542525411127, 11.360686913324946, 11.208027546491145, 10.942502747447337, 10.890253592205074, 10.890228533541407, 10.8
27738324055431, 10.19350264289451, 9.926408825858188, 9.505925824648852, 9.505726126188144, 9.378218865322744, 9.35025280471
5382, 9.339041748931576, 9.11456710428558, 9.114469818004006, 8.86418226816186, 8.755240118842638, 8.754192655468955, 8.6347
56290735002, 8.509605840467819, 8.48379379685436, 8.273396749507016, 7.948278856989093, 7.755077731834123, 7.72315903429777
3, 7.24026877282789, 7.042302076368043, 7.019499657326633, 7.002117946333455, 6.690565130465119, 6.681120950834765, 6.635239
542333492, 5.686028154873698, 5.44814662326173, 5.445812458496918, 5.318783188321104, 5.297609204574494, 5.281842695615186,
5.263010593343056, 5.2619781997152035, 5.260805934336257, 5.2245795377260285, 5.20616600201366, 5.2037519173065885, 5.142252
381877384, 4.991395079672352, 4.983811514194792, 4.903089071483303, 4.8836319634213545, 4.551337894370747, 4.54136687785643
3, 4.406446337146338, 4.356799406688445, 4.356061549112548, 4.326891733023763, 4.308050281386169, 4.180461270655399, 4.16691
1567478475, 4.115636212739788, 4.111605209010875, 4.107902782162861, 4.107348199235545, 4.106998549975373, 4.10216227334065
4, 4.075933154708472, 4.055312919297914, 4.020649654909389, 3.980874258725647, 3.9411885676818326, 3.909974274475139, 3.8668
156900981003, 3.8590848068248884, 3.8579495713833585, 3.8573024518862002, 3.855666238391819, 3.8428487951194477, 3.832244183
81089, 3.8046100608141225, 3.767953103332346, 3.7509609155621186, 3.7509606696303037, 3.6839119211636135, 3.676865618884454,
3.6754688039532404, 3.6750232878362055, 3.6730058824193943, 3.501094225799247, 3.471812947395098, 3.463503174183527, 3.46344
2346989066, 3.46107693872914, 3.459972600475868, 3.3190329959496383, 3.2850399782962394, 3.27983298292124, 3.273753416627594
3, 3.046657249106597, 3.038375075489037, 3.0315682778361, 2.99643400207361, 2.71493558444622, 2.70820307673432, 2.7052683942
59873, 2.6680427460023637, 2.60081998313357, 2.588265401537808, 2.5881239430389638, 2.5880929962634864, 2.3897752660791647,
2.3637843608910245, 2.360122824786564, 2.347316074638622, 2.342460137042945, 2.341771775692417, 2.3175662283094787, 2.317524
24172176, 2.2828128083205406, 2.1091603296776196, 2.0751849318171214, 2.013371618938261, 1.972517375157501, 1.81067548060672
11, 1.8017657007332417, 1.8006034409741305, 1.7858273992563574, 1.7596080054571157, 1.7417897821985755, 1.6847154447635984,
1.6810289351507934, 1.6564739605934864, 1.6457129080689654, 1.6422993002423474, 1.6065457261518385, 1.6063359155282897, 1.56
94307130922909, 1.5679831235405914, 1.565163408723794, 1.5578205044904603, 1.557669665461373, 1.4525481853014583, 1.45254698
78301318, 1.451066738649818, 1.3607582488095031, 1.3568888387865161, 1.3546547351872258, 1.3327412425605278, 1.3320431736105
731, 1.1669069501139135, 1.149126899094925, 1.1289732993473072, 1.089340516031547, 0.9940062437466045, 0.9714848844390498,
0.9558374323417164, 0.9208868753904774, 0.6757586271682987, 0.6712035058876081, 0.48003583950441076, 0.44667546369530897, 0.
2804683725923422, 0.1878256686660125, 0.15533107009331026]
```



In [ ]:

```
#plot eigenvalues and eigenvectors
# a = plt.figure(figsize=(10,10))
# first10Eigenvectors = z[:10].T

# plt.plot(first10Eigenvectors)
# plt.xlabel('Eigenvalues')
# plt.ylabel('Eigenvectors')
# plt.title('First 10 Eigenvalues and Eigenvectors')
# plt.legend(['1st', '2nd', '3rd', '4th', '5th', '6th', '7th', '8th', '9th', '10th'])
# plt.show()

# print(first10Eigenvectors.shape)

#plot top 3 eigenvalues this will give you an idea of the basis function employed by the PCA
plt.plot(v_sorted[0])
plt.plot(v_sorted[1])
plt.plot(v_sorted[2])
plt.plot(v_sorted[150])
plt.plot(v_sorted[219])
plt.xlabel('Eigenvalues')
plt.ylabel('Eigenvectors')
plt.title('First 3 Eigenvalues and Eigenvectors')
plt.legend(['1st', '2nd', '3rd', '150th', '220th'])
plt.show()
```



Student's comments to Exercise 1

Add comments to the results of Exercise 1 here (may use LateX for formulas if needed).

Exercise 2 – Classification using dimensionality reduction and whitening (optional)

In this exercise you will apply a simple 2-class linear discriminant analysis (LDA) classifier to the data belonging to the two classes, before and after **whitening** (i.e. applying PCA plus rescaling each coefficient to unit variance: $y = \Lambda^{-\frac{1}{2}} W^T x$. Note that matrix Λ is obtained as one of the outputs of *eig.m*). This classifier makes the Naïve Bayes Classifier assumption that the features are statistically independent, thus the shared covariance matrix is taken as $\Sigma = I$. We also assume that class 0 and class 1 are equiprobable. Thus, letting μ_0 and μ_1 be the mean of vectors in class 0 and 1, we define $x_0 = 0.5(\mu_0 + \mu_1)$ and $w = \mu_1 - \mu_0$. A test vector x is classified into class 1 or 0 depending on whether $sign(w^T(x - x_0))$ is equal to +1 or -1.

When the classifier is applied to the PCA coefficients, you simply replace μ_0 and μ_1 with their reduced versions (subtracting μ and applying the same PCA matrix computed over the training set), and recalculate x_0 and w accordingly.

Task: Divide the Indian Pines dataset into training and test sets (e.g. 75% of the data of each class to be used as training data, and 25% as test data). Train this classifier on the training data, and apply it to the test data. In particular, you should perform the following:

- Plot the mean vector of class 0 and 1 – this will give you a visual description of the differences among vectors of either class.
- Apply the classifier to the original data (without PCA) and compute its accuracy
- Apply the classifier to the PCA coefficients for different values of K, and compute its accuracy
- Apply the classifier to the original data where only the first K features have been retained, and compute its accuracy (this is a more brutal way to reduce dimensionality)
- Try to classify the data using a Support Vector Machine and compare the results

Plot the mean vector of class 0 and 1 – this will give you a visual description of the differences among vectors of either class.

In [ ]:
```
#Insert code here
```

Apply the classifier to the original data (without PCA) and compute its accuracy

In [ ]:
```
#For this section you can use the code you developed for the previous Lab or the sklearn function LinearDiscriminantAnalysis
```

Apply the classifier to the PCA coefficients for different values of K, and compute its accuracy

In [ ]:
```
#Insert code Here
```

Apply the classifier to the original data where only a subset of K features (selected randomly) have been retained, and compute its accuracy (this is a more brutal way to reduce dimensionality)

In [ ]:
```
#Insert code here
```

Try to classify the data using a Support Vector Machine and compare the results

In [ ]:
```
#For this section you can use the sklearn library

from sklearn import svm

# Train = bothClassTrain
# Test = bothClassTest
# Train_Label = bothClassTrain

# svc = svm.SVC()
# svc.fit(Train, Train_Label)
```

```
# predict = svc.predict(Test)

#print(metrics.accuracy_score(predict, Test_Label))
```

In [ ]:
```
bothClassTrain
```

Out[ ]:
```
array([[ 1.18668e+02, -5.83320e+01,  6.56668e+02, ..., -2.16332e+02,
        -4.74332e+02, -3.33320e+01],
       [-1.03319e+02, -2.31319e+02,  4.19681e+02, ..., -1.11319e+02,
        -3.26319e+02,  1.75681e+02],
       [-1.71800e+00, -8.37180e+01,  6.84282e+02, ..., -2.65718e+02,
        -4.87180e+01,  3.22820e+01],
       ...,
       [ 4.61500e+00, -4.38500e+00,  2.66150e+01, ...,  6.15000e-01,
        -7.38500e+00, -3.38500e+00],
       [ 3.38000e+00, -5.62000e+00,  3.38000e+00, ..., -1.62000e+00,
        -6.20000e-01,  8.38000e+00],
       [-1.01370e+01, -1.13700e+00,  8.86300e+00, ..., -4.13700e+00,
        -4.13700e+00,  4.86300e+00]])
```

Student's comments to Exercise 2

Add comments to the results of Exercise 2 here (may use LateX for formulas if needed).

STATISTICAL LEARNING AND NEURAL NETWORKS, A.A. 2022/2023

COMPUTER LAB 4 - Kalman filter

Duration: 3 hours

Introduction:

In this lab, you are provided with the set of coordinates (x,y – horizontal and vertical) describing the trajectories of pedestrians moving acrooss a scene. Your task is to simulate the observed positions of the pedestrians by adding observation noise, then to track the subjects using a Kalman filter. In other words, you need to estimate the next (x,y) positions, from the observations of the previous positions.

Simulating the observed coordinates

Choose one of the trajectories in the dataset. This data will be considered the real trajectory. Generate the observed directory by adding observation noise $\delta_t \sim\ \mathcal{N}(0,\sigma^{2}_{R})$ to the (x,y) coordinates.

Designing the Kalman filter

Your task is to **design a Kalman filter** based on a constant velocity model, which tracks the next (x,y) position of the object, from the observation of the previous positions. The code must be based on the following model.

- The state vector contains coordinates and velocities: $z_t^T = (z_{1t}, z_{2t}, v_{1t}, v_{2t})$ (see slides). The object has initial coordinates (0,0) and velocity (Δ, Δ).
- Only the coordinates (but not the velocities) are observed. This leads to a linear dynamical system with:

$$A = \begin{pmatrix} 1 & 0 & \Delta & 0 \\ 0 & 1 & 0 & \Delta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

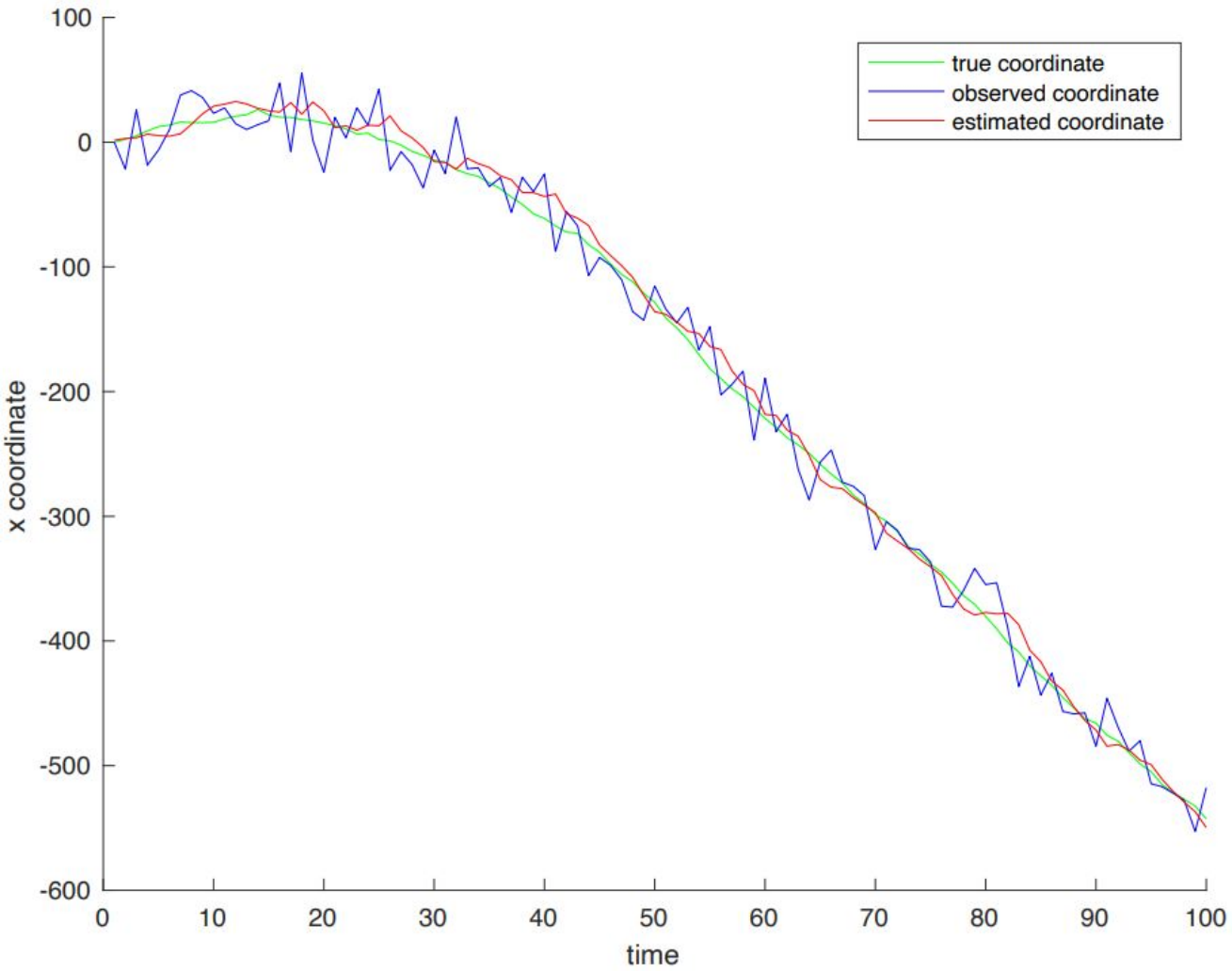
and B = D = 0.

- Σ_Q, and Σ_R should be set to:

$$\Sigma_Q = \sigma_Q^2 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\Sigma_R = \sigma_R^2 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

To see if your Kalman filter is working well, you should plot the estimated position of the object over time with respect to the true position (i.e., the first two entries of the state vector) and the observed position. Depending on the chosen parameters, for each coordinate the graph may look something like this:



Suggestion: when implementing your Kalman filter, you will have to choose initial values for μ_t and Σ_t. Provided that you do not make very unreasonable assumptions, the Kalman filter will update those estimates from observed data, so the initial choices are not very critical.

Test your Kalman filter modifying the values of some of the parameters, including standard deviations σ_Q and σ_R, initial values for μ_t and Σ_t and the value of Δ.

In [ ]:
```
# from google.colab import drive
# drive.mount('/content/drive')

# %pip install ndjson
```

In [ ]:
```
import ndjson
import pandas as pd
import numpy as np

#Dataset origin: https://paperswithcode.com/dataset/trajnet-1
#The crowds_students001 file is loaded, and formatted as a list of numpy vectors

with open('crowds_students001_trackonly.ndjson') as f:
  data = ndjson.load(f)

p='-1'

person_dict = []

for ii in range(len(data)):
  if(p!=data[ii]['track']['p']):
    p=data[ii]['track']['p']
    person_dict.append([])
  person_dict[p].append((data[ii]['track']['x'],data[ii]['track']['y']))

person_dict_numpy = []
for ii in range(148):
  person_dict_numpy.append(np.array(person_dict[ii], dtype=float))
# print(person_dict_numpy)
# print(len(person_dict_numpy))
```

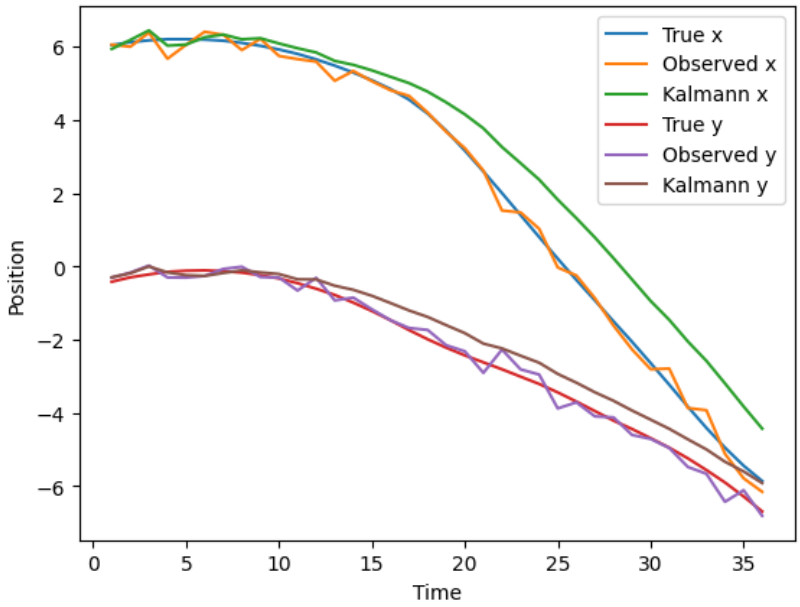
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

# Implement the kalman filter as a function using the numpy library, the matrix products can be done using the np.dot functi
# The matrix inversion can be done using the np.linalg.inv

def kalman_filter(y, mu_tmin1, sigma_tmin1, A, C, Q, R):
  # Predict the state at the next time step
  mu_tilde = np.dot(A, mu_tmin1) # (4x4) * (4x1) = (4x1)
  sigma_tilde = np.dot(np.dot(A, sigma_tmin1), A.T) + Q # (4x4) * (4x4) * (4x4) = (4x4)
  y_hat = np.dot(C, mu_tilde) # (2x4) * (4x1) = (2x1)
```

```
  # Calculate the Kalman gain
  S = np.dot(np.dot(C, sigma_tilde), C.T) + R # (2x4) * (4x4) * (4x2) + (2x2) = (2x2)
  K = np.dot(np.dot(sigma_tilde, C.T), np.linalg.inv(S)) # (4x4) * (4x2) * (2x2) = (4x2)

  # Update the estimate of the state
  #r = np.subtract(y, y_hat)
  y = y.reshape(-1, 1) # reshape y to be a column vector (2x1) instead of a row vector (1x2) | same as y = y[:, np.newaxis] a
  mu_est = mu_tilde + np.dot(K, (y - y_hat)) # (4x1) + (4x2) * (2x1) = (4x1)
  sigma_est = np.dot((np.eye(4) - np.dot(K, C)), sigma_tilde) # (4x4) - (4x2) * (2x4) * (4x4) = (4x4)

  return mu_est, sigma_est

true_positions = person_dict_numpy[42] #choose a single trajectory by taking an element of the list person_dict_numpy, selec

#Define parameter delta
Delta = 1

A = np.array([[1, 0, Delta, 0],
              [0, 1, 0, Delta],
              [0, 0, 1, 0 ],
              [0, 0, 0, 1 ]])


# Define the measurement matrix
C = np.array([[1, 0, 0, 0],  # x is measured directly
              [0, 1, 0, 0]]) # y is measured directly

# Set the standard deviation of the measurement noise

sigma_R = 0.2
sigma_Q = 0.001

# Define the process noise covariance matrix
Q = (sigma_Q**2)*np.eye(4) # 4x4

# Define the measurement noise covariance matrix
R = (sigma_R**2)*np.eye(2) # 2x2

# Set the initial state and covariance
mu_0 = np.array([[0,0,0,0]]).T # 4x1
# sigma_0 = [1, 0, 1, 0], [0, 1, 0, 1], [0, 0, 1, 0], [0, 0, 0, 1] # 4x4
sigma_0 = np.eye(4) # 4x4

# Iterate over the observed coordinates
y_est = []
y_est.append((mu_0, sigma_0))


#data initialization
TSeries = []
XKalman = []
YKalman = []
XObs = []
YObs = []
XTrue = []
YTrue = []

for t in range(1,len(true_positions)):
  print(t, y_est[t-1][0].shape, y_est[t-1][1].shape, A.shape, C.shape, Q.shape, R.shape)
  print(y_est[t-1][0], y_est[t-1][1])
  # Get the observed coordinates at time t
  # Note: the observed position is simulated by adding gaussian noise to the true_positions
  y = true_positions[t] + np.random.normal(0,sigma_R,2)

  #apply the kalman filter on the observed coordinates
  y_est.append(kalman_filter(y, y_est[t-1][0], y_est[t-1][1], A, C, Q, R))
  XKalman.append(y_est[t][0][0]) # x coordinate
  YKalman.append(y_est[t][0][1]) # y coordinate
  TSeries.append(t)
  XObs.append(y[0])
  YObs.append(y[1])
  XTrue.append(true_positions[t][0])
  YTrue.append(true_positions[t][1])

#Plot the trajectory of the x coordinate over time and the trajectory of the y coordinate over time into two separate plot
plt.plot(TSeries, XTrue, label='True x')
plt.plot(TSeries, XObs, label='Observed x')
plt.plot(TSeries, XKalman, label='Kalmann x')
plt.plot(TSeries, YTrue, label='True y')
plt.plot(TSeries, YObs, label='Observed y')
plt.plot(TSeries, YKalman, label='Kalmann y')
plt.xlabel('Time')
plt.ylabel('Position')
plt.legend()
plt.show()


#Use the matplotlib library to plot the true positions, observed positions y and the results of the kalman filtering
#You should obtain a plot which resembles the one in the figure
#Plot the trajectory of the x coordinate over time and the trajectory of the y coordinate over time into two separate plot
```

```
1 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[0]
 [0]
 [0]
 [0]] [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
2 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 5.93379174]
 [-0.29950394]
 [ 2.96689439]
 [-0.14975189]] [[0.03921569 0.         0.01960783 0.        ]
 [0.         0.03921569 0.         0.01960783]
 [0.01960783 0.         0.50980516 0.        ]
 [0.         0.01960783 0.         0.50980516]]
3 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 6.18019301]
 [-0.18853168]
 [ 0.51845414]
 [ 0.08489951]] [[0.03745319 0.         0.03370782 0.        ]
 [0.         0.03745319 0.         0.03370782]
 [0.03370782 0.         0.06367216 0.        ]
 [0.         0.03370782 0.         0.06367216]]
4 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 6.43986200e+00]
 [-2.84763824e-03]
 [ 3.68933581e-01]
 [ 1.43130665e-01]] [[0.03232768 0.         0.01867825 0.        ]
 [0.         0.03232768 0.         0.01867825]
 [0.01867825 0.         0.01820097 0.        ]
 [0.         0.01867825 0.         0.01820097]]
5 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 6.02494938]
 [-0.1641828 ]
 [ 0.04001219]
 [ 0.01536922]] [[0.02748887 0.         0.01153502 0.        ]
 [0.         0.02748887 0.         0.01153502]
 [0.01153502 0.         0.00756691 0.        ]
 [0.         0.01153502 0.         0.00756691]]
6 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 6.05401129]
 [-0.24230492]
 [ 0.03641365]
 [-0.01535437]] [[0.02369457 0.         0.00778663 0.        ]
 [0.         0.02369457 0.         0.00778663]
 [0.00778663 0.         0.00384942 0.        ]
 [0.         0.00778663 0.         0.00384942]]
7 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 6.25306299]
 [-0.260696  ]
 [ 0.08030376]
 [-0.01617387]] [[0.02075032 0.         0.00559976 0.        ]
 [0.         0.02075032 0.         0.00559976]
 [0.00559976 0.         0.00222144 0.        ]
 [0.         0.00559976 0.         0.00222144]]
8 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 6.32734298e+00]
 [-1.81634822e-01]
 [ 7.89250660e-02]
 [ 5.62311273e-03]] [[0.0184286  0.         0.00421786 0.        ]
 [0.         0.0184286  0.         0.00421786]
 [0.00421786 0.         0.00139773 0.        ]
 [0.         0.00421786 0.         0.00139773]]
9 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 6.19825006]
 [-0.10747727]
 [ 0.03759397]
 [ 0.01924022]] [[0.01656125 0.         0.00329056 0.        ]
 [0.         0.01656125 0.         0.00329056]
 [0.00329056 0.         0.00093677 0.        ]
 [0.         0.00329056 0.         0.00093677]]
10 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 6.22626063e+00]
 [-1.64208192e-01]
 [ 3.59115787e-02]
 [ 5.90331265e-03]] [[0.01503126 0.         0.00263877 0.        ]
 [0.         0.01503126 0.         0.00263877]
 [0.00263877 0.         0.00065889 0.        ]
 [0.         0.00263877 0.         0.00065889]]
11 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 6.08374877e+00]
 [-2.08945449e-01]
 [ 7.85161798e-03]
 [-2.06073291e-03]] [[0.01375703 0.         0.00216351 0.        ]
 [0.         0.01375703 0.         0.00216351]
 [0.00216351 0.         0.00048153 0.        ]
 [0.         0.00216351 0.         0.00048153]]
12 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 5.95384896]
 [-0.3532878 ]
 [-0.01177281]
 [-0.02233054]] [[0.01268067 0.         0.00180652 0.        ]
 [0.         0.01268067 0.         0.00180652]
 [0.00180652 0.         0.00036307 0.        ]
 [0.         0.00180652 0.         0.00036307]]
```

```
13 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 5.83822667]
 [-0.35729595]
 [-0.02529868]
 [-0.01994414]] [[0.01176028 0.         0.00153172 0.        ]
 [0.         0.01176028 0.         0.00153172]
 [0.00153172 0.         0.00028099 0.        ]
 [0.         0.00153172 0.         0.00028099]]
14 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 5.60826891]
 [-0.52841731]
 [-0.0498581 ]
 [-0.03808565]] [[0.01096489 0.         0.0013158  0.        ]
 [0.         0.01096489 0.         0.0013158 ]
 [0.0013158  0.         0.00022236 0.        ]
 [0.         0.0013158  0.         0.00022236]]
15 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 5.50046458]
 [-0.64021084]
 [-0.05630758]
 [-0.04628942]] [[0.0102712 0.        0.0011432 0.       ]
 [0.        0.0102712 0.        0.0011432]
 [0.0011432 0.        0.0001794 0.       ]
 [0.        0.0011432 0.        0.0001794]]
16 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 5.34966622]
 [-0.80403093]
 [-0.06611865]
 [-0.05849274]] [[0.00966134 0.         0.00100315 0.        ]
 [0.         0.00966134 0.         0.00100315]
 [0.00100315 0.         0.00014723 0.        ]
 [0.         0.00100315 0.         0.00014723]]
17 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 5.17490708]
 [-1.00102311]
 [-0.07669576]
 [-0.07197688]] [[0.00912143 0.         0.00088805 0.        ]
 [0.         0.00912143 0.         0.00088805]
 [0.00088805 0.         0.00012269 0.        ]
 [0.         0.00088805 0.         0.00012269]]
18 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 5.00199644]
 [-1.20465753]
 [-0.08551951]
 [-0.08405105]] [[0.0086405  0.         0.00079241 0.        ]
 [0.         0.0086405  0.         0.00079241]
 [0.00079241 0.         0.00010367 0.        ]
 [0.         0.00079241 0.         0.00010367]]
19 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 4.76912414]
 [-1.38042428]
 [-0.0983017 ]
 [-0.09200697]] [[8.20980959e-03 0.00000000e+00 7.12163960e-04 0.00000000e+00]
 [0.00000000e+00 8.20980959e-03 0.00000000e+00 7.12163960e-04]
 [7.12163960e-04 0.00000000e+00 8.87156458e-05 0.00000000e+00]
 [0.00000000e+00 7.12163960e-04 0.00000000e+00 8.87156458e-05]]
20 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 4.47579006]
 [-1.60576847]
 [-0.11436503]
 [-0.10298894]] [[7.82228451e-03 0.00000000e+00 6.44261902e-04 0.00000000e+00]
 [0.00000000e+00 7.82228451e-03 0.00000000e+00 6.44261902e-04]
 [6.44261902e-04 0.00000000e+00 7.68162403e-05 0.00000000e+00]
 [0.00000000e+00 6.44261902e-04 0.00000000e+00 7.68162403e-05]]
21 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 4.14935412]
 [-1.82271714]
 [-0.13100732]
 [-0.11193194]] [[7.47215409e-03 0.00000000e+00 5.86377968e-04 0.00000000e+00]
 [0.00000000e+00 7.47215409e-03 0.00000000e+00 5.86377968e-04]
 [5.86377968e-04 0.00000000e+00 6.72456319e-05 0.00000000e+00]
 [0.00000000e+00 5.86377968e-04 0.00000000e+00 6.72456319e-05]]
22 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 3.76702341]
 [-2.1085991 ]
 [-0.14986052]
 [-0.12498092]] [[7.15466329e-03 0.00000000e+00 5.36712181e-04 0.00000000e+00]
 [0.00000000e+00 7.15466329e-03 0.00000000e+00 5.36712181e-04]
 [5.36712181e-04 0.00000000e+00 5.94754382e-05 0.00000000e+00]
 [0.00000000e+00 5.36712181e-04 0.00000000e+00 5.94754382e-05]]
23 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 3.2579624 ]
 [-2.2398444 ]
 [-0.17569743]
 [-0.12543152]] [[6.86586020e-03 0.00000000e+00 4.93854098e-04 0.00000000e+00]
 [0.00000000e+00 6.86586020e-03 0.00000000e+00 4.93854098e-04]
 [4.93854098e-04 0.00000000e+00 5.31146958e-05 0.00000000e+00]
 [0.00000000e+00 4.93854098e-04 0.00000000e+00 5.31146958e-05]]
24 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 2.81645674]
 [-2.43899453]
 [-0.1940832 ]
 [-0.13053058]] [[6.60243417e-03 0.00000000e+00 4.56685657e-04 0.00000000e+00]
 [0.00000000e+00 6.60243417e-03 0.00000000e+00 4.56685657e-04]
 [4.56685657e-04 0.00000000e+00 4.78698757e-05 0.00000000e+00]
 [0.00000000e+00 4.56685657e-04 0.00000000e+00 4.78698757e-05]]
```

```
25 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 2.36834775]
 [-2.63049711]
 [-0.21102644]
 [-0.13459734]] [[6.36159108e-03 0.00000000e+00 4.24311134e-04 0.00000000e+00]
 [0.00000000e+00 6.36159108e-03 0.00000000e+00 4.24311134e-04]
 [4.24311134e-04 0.00000000e+00 4.35176625e-05 0.00000000e+00]
 [0.00000000e+00 4.24311134e-04 0.00000000e+00 4.35176625e-05]]
26 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 1.82149687]
 [-2.93588473]
 [-0.23268242]
 [-0.14561093]] [[6.14095634e-03 0.00000000e+00 3.96005891e-04 0.00000000e+00]
 [0.00000000e+00 6.14095634e-03 0.00000000e+00 3.96005891e-04]
 [3.96005891e-04 0.00000000e+00 3.98860885e-05 0.00000000e+00]
 [0.00000000e+00 3.96005891e-04 0.00000000e+00 3.98860885e-05]]
27 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 1.31614303]
 [-3.17499136]
 [-0.24972541]
 [-0.15145476]] [[5.93849862e-03 0.00000000e+00 3.71178381e-04 0.00000000e+00]
 [0.00000000e+00 5.93849862e-03 0.00000000e+00 3.71178381e-04]
 [3.71178381e-04 0.00000000e+00 3.68412465e-05 0.00000000e+00]
 [0.00000000e+00 3.71178381e-04 0.00000000e+00 3.68412465e-05]]
28 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 0.78773817]
 [-3.43699885]
 [-0.26664933]
 [-0.15816852]] [[5.75246924e-03 0.00000000e+00 3.49341619e-04 0.00000000e+00]
 [0.00000000e+00 5.75246924e-03 0.00000000e+00 3.49341619e-04]
 [3.49341619e-04 0.00000000e+00 3.42777906e-05 0.00000000e+00]
 [0.00000000e+00 3.49341619e-04 0.00000000e+00 3.42777906e-05]]
29 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[ 0.22274023]
 [-3.66874167]
 [-0.28429421]
 [-0.16251984]] [[5.58135372e-03 0.00000000e+00 3.30091519e-04 0.00000000e+00]
 [0.00000000e+00 5.58135372e-03 0.00000000e+00 3.30091519e-04]
 [3.30091519e-04 0.00000000e+00 3.21120527e-05 0.00000000e+00]
 [0.00000000e+00 3.30091519e-04 0.00000000e+00 3.21120527e-05]]
30 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[-0.36071562]
 [-3.93581586]
 [-0.30156329]
 [-0.16855523]] [[5.42383268e-03 0.00000000e+00 3.13090282e-04 0.00000000e+00]
 [0.00000000e+00 5.42383268e-03 0.00000000e+00 3.13090282e-04]
 [3.13090282e-04 0.00000000e+00 3.02769923e-05 0.00000000e+00]
 [0.00000000e+00 3.13090282e-04 0.00000000e+00 3.02769923e-05]]
31 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[-0.94551292]
 [-4.18428429]
 [-0.31755551]
 [-0.17306736]] [[5.27874997e-03 0.00000000e+00 2.98053525e-04 0.00000000e+00]
 [0.00000000e+00 5.27874997e-03 0.00000000e+00 2.98053525e-04]
 [2.98053525e-04 0.00000000e+00 2.87184466e-05 0.00000000e+00]
 [0.00000000e+00 2.98053525e-04 0.00000000e+00 2.87184466e-05]]
32 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[-1.45949564]
 [-4.43453648]
 [-0.32842621]
 [-0.17733893]] [[5.14508666e-03 0.00000000e+00 2.84740219e-04 0.00000000e+00]
 [0.00000000e+00 5.14508666e-03 0.00000000e+00 2.84740219e-04]
 [2.84740219e-04 0.00000000e+00 2.73923185e-05 0.00000000e+00]
 [0.00000000e+00 2.84740219e-04 0.00000000e+00 2.73923185e-05]]
33 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[-2.04890396]
 [-4.72023698]
 [-0.34261071]
 [-0.18322844]] [[5.02193954e-03 0.00000000e+00 2.72944769e-04 0.00000000e+00]
 [0.00000000e+00 5.02193954e-03 0.00000000e+00 2.72944769e-04]
 [2.72944769e-04 0.00000000e+00 2.62624450e-05 0.00000000e+00]
 [0.00000000e+00 2.72944769e-04 0.00000000e+00 2.62624450e-05]]
34 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[-2.5799412 ]
 [-4.99654648]
 [-0.35268715]
 [-0.18820611]] [[4.90850338e-03 0.00000000e+00 2.62490723e-04 0.00000000e+00]
 [0.00000000e+00 4.90850338e-03 0.00000000e+00 2.62490723e-04]
 [2.62490723e-04 0.00000000e+00 2.52989670e-05 0.00000000e+00]
 [0.00000000e+00 2.62490723e-04 0.00000000e+00 2.52989670e-05]]
35 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[-3.19517355]
 [-5.33391209]
 [-0.36652612]
 [-0.19606843]] [[4.80405610e-03 0.00000000e+00 2.53225745e-04 0.00000000e+00]
 [0.00000000e+00 4.80405610e-03 0.00000000e+00 2.53225745e-04]
 [2.53225745e-04 0.00000000e+00 2.44770730e-05 0.00000000e+00]
 [0.00000000e+00 2.53225745e-04 0.00000000e+00 2.44770730e-05]]
36 (4, 1) (4, 4) (4, 4) (2, 4) (4, 4) (2, 2)
[[-3.82329117]
 [-5.59833772]
 [-0.38014023]
 [-0.19962597]] [[4.70794642e-03 0.00000000e+00 2.45017568e-04 0.00000000e+00]
 [0.00000000e+00 4.70794642e-03 0.00000000e+00 2.45017568e-04]
 [2.45017568e-04 0.00000000e+00 2.37760213e-05 0.00000000e+00]
 [0.00000000e+00 2.45017568e-04 0.00000000e+00 2.37760213e-05]]
```



In [ ]:
```
# #plot TrueCoordinates, ObservedCoordinates and KalmanEstimate at one plot as lat,long , not seperated
# plt.plot(XTrue, YTrue, label='True')
# plt.plot(XObs, YObs, label='Observed')
# plt.plot(XKalman, YKalman, label='Kalmann')
# plt.xlabel('X')
# plt.ylabel('Y')
# plt.legend()
# plt.show()
```