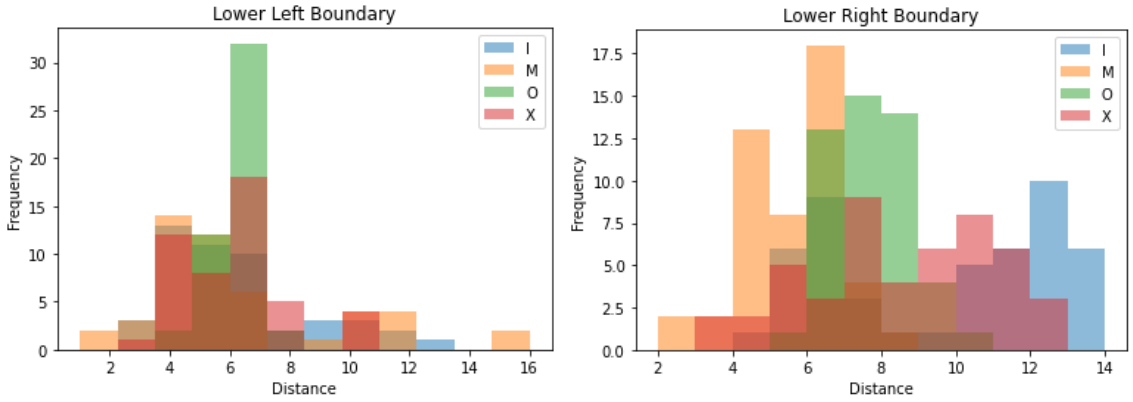
Brenden Hein

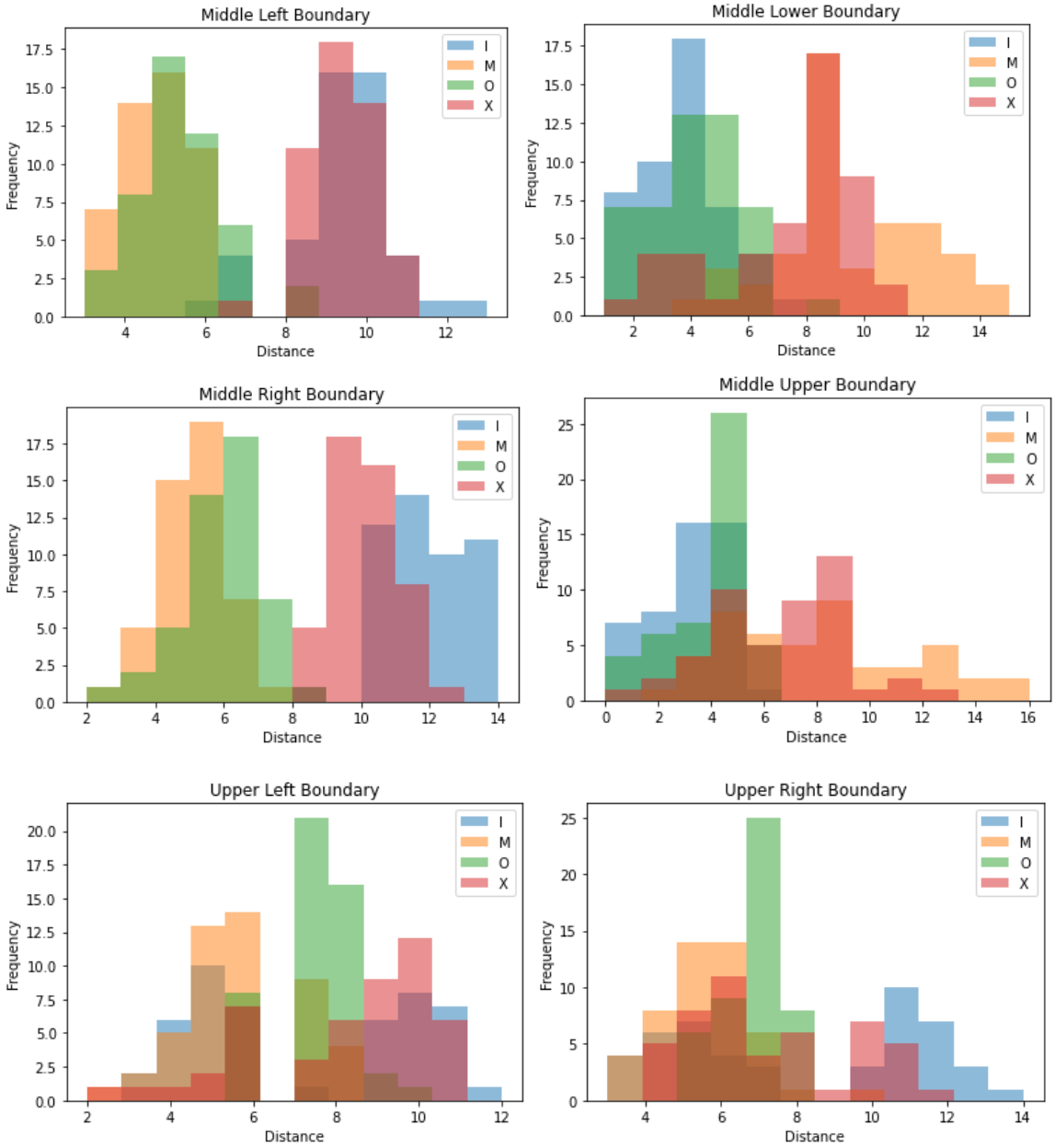
2/1/2021

Dr. Ross

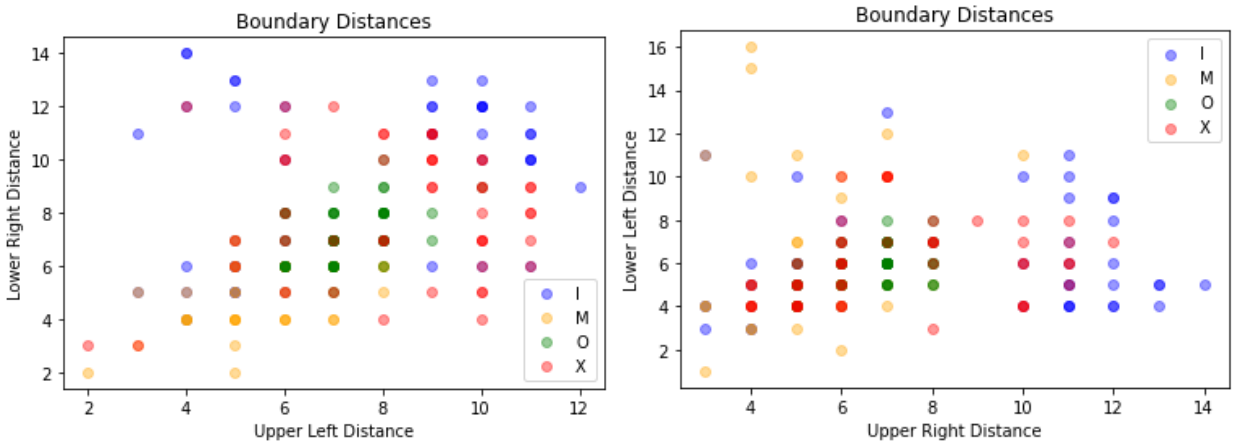
Homework 1

1. Question 1
    a. For each class, the mean pattern vector is…
        i. X1 = [7.333, 9.208, 8.188, 5.938, 9.312, 11.438, 3.146, 3.771]
        ii. X2 = [5.667, 5.125, 5.375, 6.062, 4.646, 4.604, 7.896, 9.438]
        iii. X3 = [7.312, 7.208, 6.729, 5.979, 5.333, 5.479, 3.729, 4.208]
        iv. X4 = [8.271, 8.083, 7.25, 6.0, 9.188, 9.625, 6.417, 7.438]
    b. For each class, the pattern that is the farthest away from the mean is…
        i. Pattern 21 = [12, 9, 7, 13, 13, 8, 3, 2] (Distance = 10.084)
        ii. Pattern 26 = [6, 6, 4, 16, 4, 4, 4, 14] (Distance = 11.76)
        iii. Pattern 33 = [6, 6, 5, 5, 4, 4, 0, 2] (Distance = 5.466)
        iv. Pattern 33 = [2, 3, 12, 7, 8, 12, 7, 8] (Distance = 9.82)
    c. The 8 histograms, which can be seen on the next page, show off the 8 features in the feature set. For each feature, a comparison of all the distance measurements for the 4 letter classes can be visualized. And as it can be noted in these histograms, some of these are not so great options to classify these patterns. The worst of these appears to be the lower left boundary, as the different letters seem to all huddle around the lower side. Intuitively, this makes sense, as each of 4 letters protrudes out to the lower left, so there is little interclass variability. Two other bad options would be the upper left and upper right boundaries. While they are not as clustered as the lower left boundary, some of the patterns due tend to be very similar (such as “M”, “O”, and “X” for the upper right boundary). On the other hand, there are some much better options seen in our histograms. The best one is the middle right boundary. While it is not perfect, there is noticeable delineation between the 4 letters, which makes creating a decision boundary with a low enough error rate much more manageable. The middle lower is not a bad option either, although there is noticeable invariance with the “I” and the “O.”

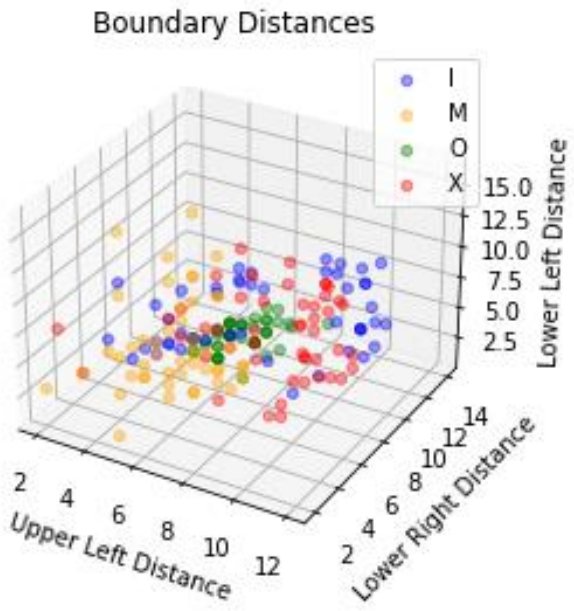




d. Based on these two scatterplots, it appears that the upper left/lower right group (x1, x2) is noticeably better than the upper right/lower left group (x3, x4). The latter is cluttered, with "X" (red sweeping across the board for the upper right distance. On top of this, "I" (blue) seems to be very polarizing for the upper right distance, and yellow is scattered, as well. On the other hand, the upper left/lower right pair (x1, x2) seems to be a much better fit. While it is not perfect, there is a decent delineation between the groups (outside of X's mild scatteredness though out the middle around "O" (green)). However, there seems to be an almost positive linear pattern going up and to the right starting with "M" (yellow), then going to "O" and "X", and finally ending with "I." This pattern makes it substantially easier to create a decision boundary, making the first scatterplot the much better option.



e. Based on this scatter plot, there are appears to be some overlap, although it is relatively mild. In the middle of the plot, there appears to be overlap with "I", "O", and "X." On top of this, on the bottom left side, there is some overlap with "I" and "M", although it is scattered. Finally, "I" and "X" do have a tiny amount of overlap on the bottom right side. Despite this mild overlapping, this model seems to have a very sound decision boundary, and it is a decent potential option for classifying the 4 letters.



2. The best learning schemes - supervised, unsupervised, or reinforcement – for each of the following problems is…
   a. The best learning schema for teaching a computer to play chess would be reinforcement learning. When training a model to solve a chess game, it is bound to make mistakes. For example, early in the training process, it might make a poor move that exposes the king to a checkmate; obviously, this is not a good move. However, by imposing penalties for moves like these can help the system learn what moves are decent and which ones are not, the computer can learn how to play a good game of chess after countless attempts at trial and error.

b. The best learning schema for grouping seashells is unsupervised learning. When feeding your system different seashells, you are not letting the model know that these seashells are parts of certain groups I.e., classes. The model not only has to group these seashells, but they also have the job of figuring what features to even extract. This makes it unsupervised; it does not know that the images it is receiving are part of class A, class B, class C, etc.… It is the model's job to figure out what class each seashell Is a part of.
c. If classes of what cars each sideview image of the car is provided, this is supervised learning. If this were an unsupervised schema, it could discover, say n, classes. However, unless you are providing the system with the classes for each make/model pair, it could not actually name each class. That is why a supervised model is better. It is not inly provided a feature set to make its job easier, but I can categorize each image into a named class instead of an unknown cluster.
d. The best learning schema for predicting rain best on current weather conditions would be a supervised model. Firstly, you are providing a set of features (precipitation, humidity, temperature, wind, pressure, etc.) for the class to extract for each data point passed in. On top of this, you a providing 2 classes (rain or no rain); the system is not discovering them. These points really push towards a supervised learning model.
e. The best learning schema for automatically segmenting an image into distinct reasons is an unsupervised learning model. First, the application does not know what classes it is categorizing into; it must discover these classes. The hope is that this model will get to a point where it not only can delineate between colors, but it can also distinguish between textures such as rough, hairy, grassy, etc.… Again, it must discover these classes by analyzing many data points and finding correlations between them.

3. Each term's definition with an example is…
    a. Overfitting: Creating a training model that fits too tightly to the training data. This presents an issue because if you overfit your line, it is based around the training data, not the underlying model, which increases the error rate past a certain point. An example of this is distinguishing between an image of a female or a male. By overfitting, it becomes reliant on that training data. So, if you introduce an image that fits the underlying model given the feature set but does not correctly classify the image given your overfitted model, you run into misclassification issues where girls will be flagged as guys and vice-versa.
    b. Reject Option: This is where the system encounters an unknown and does not know what it should classify it as. This is completely okay, assuming your reject rate is small enough to where you are still classifying most of the datapoints you are receiving. An example of this is if you are classifying an image of a fish based on lightness and width into 3 classes: seabass, salmon, and trout. An image of a salmon is passed to it, but it is a smaller salmon with more colorful scales, so the system is confounded on if it is a salmon or a trout. Thus, it exercises its power to reject it, and sends it on its way, unclassified.
    c. Decision Boundary: This is what is used to separate classes in nth-dimensional space. This boundary creates decision regions, and points that fall inside these regions are grouped together in a known (supervised) or unknown (unsupervised) class. For

example, if you had a 2-dimensional feature set for apples and oranges (lightness and size) and a linear decision boundary between the two, then any point that falls right/above the boundary is classified as orange and the opposite for classifying it as an apple.

d. Segmentation: Removing noise from a class that you do not need and grouping together similar parts of a pattern. For example, if you are classifying between dogs and wolves, it might be a good idea to segment the animal from the background, so you do not end up with a snow detection system instead of a wolf detection system.
e. Invariant Representation: Invariance is a lack of variability between 2 or more pieces of data. In terms of this course, when dealing with intraclass variability, you want high invariance, so the patterns in your class are all very similar. On the other hand, you want low interclass invariance, as you want the different classes to be different from each other. An example of this involves classifying waveforms of speaking different letters. You want there to be little invariance between speaking the letter "p" and speaking the letter "b." However, you want significant invariance when multiple people say "P" so that the variability within the class to be very high.

4. After reading the paper "Bird Species Recognition Using Support Vector Machines…
    a. This system involved…
        i. The system used an audio recorder to gather songs and calls of different birds in field conditions.
        ii. The article describes a segmentation method that involved segmenting into different syllables of the birds' calls and songs using an iterative time-domain algorithm, grouping together similarly sounding syllables to prevent a border effect. The segmented syllables were represented by both the mel-cepstrum method and a descriptive signal parameters method once extracted.
        iii. For the descriptive parameters, the spectral features extracted from the birds' sounds were the spectral centroid, the signal bandwidth, the spectral roll-off frequency, the spectral flux, the spectral flatness, and the frequency range. The temporal features extracted were the zero-crossing rate, the short time energy, the syllable temporal duration, and the modulation spectrum. Finally, there was the MFCC features.
        iv. The classification model used involves a binary decision tree which contains binary support vector machines classifiers at each node. At each node, only 2 classes are considered for classification of a certain pattern. The patterns make their way down the tree until all classes, but one, are rejected, and the pattern is then matched to that surviving class.
        v. The study ended up using 6 features for the classification of syllables.
        vi. The number of classes present were 14 different Canadian birds between the 2 datasets.
    b. Classifier training was accomplished by extracting individual syllables from the calls and songs of different birds and using a binary decision tree with support vector machines at every node in the tree to classify each sample. There were 1,270 and 2,278 patterns available in the training set for dataset 1 and dataset 2, respectively. The training patterns were labeled by using a SVM to compare 2 classes at each node, with the

winning class moving on. The class that makes it to the bottom of the tree is the one the pattern gets mapped to.

c. The performance of the system was evaluated by using different parametric representations of the testing data and calculating the percentage of correctly classified points using each of those features. Syllables used in the testing phase were also different than that of the training phase for accuracy. The metrics used to classify performance were a comp method, MFCC, MFCC Δ, MFCC ΔΔ, a mixture model for all the MFC-coefficients, and a reference method.

d. I think the system did work well. When looking at the results, it appears dataset 2 did extraordinarily well, correctly classifying sounds >95% of the time across the board. While dataset 1 seemed to look a little worse, it is still sound. The thing to note about data set 1 was that it had less species, and those birds were more closely related, making it harder to classify the similar sounding syllables in dataset 1.

5.

$$\int_0^{10} k * x^3 * (10 - x) = 1$$

$$\int_0^{10} 10kx^3 - kx^4 = 1$$

$$\frac{10kx^4}{4} - \frac{kx^5}{5}]_0^{10} = 1$$

$$\frac{10 * 10^4 k}{4} - \frac{10^5 k}{5} - [0 - 0] = 1$$

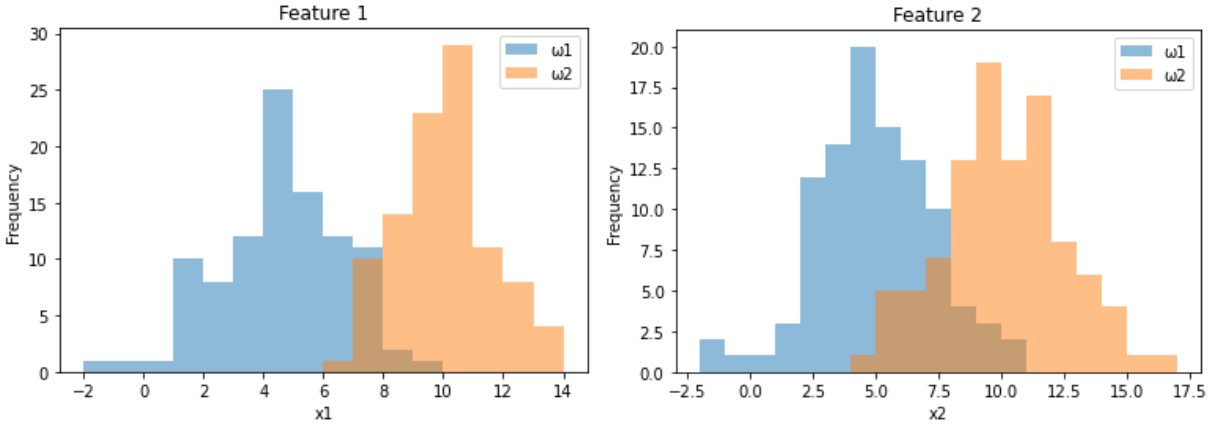
$$10^5 k * \left(\frac{1}{4} - \frac{1}{5}\right) = 1$$

$$\frac{10^5 k}{20} = 1$$

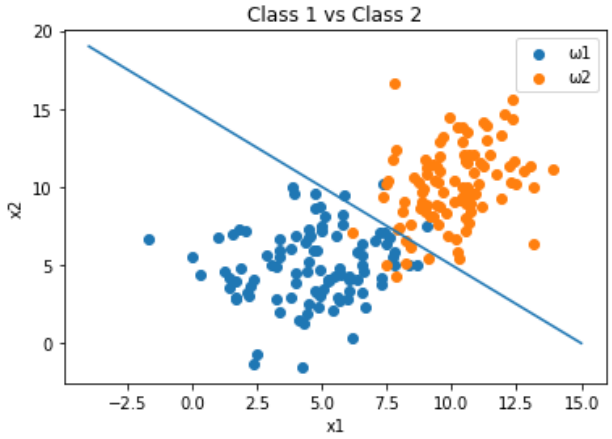
$$k = \frac{20}{100000} = \frac{\mathbf{1}}{\mathbf{5000}}$$

6. Given a 2-dimensional dataset…

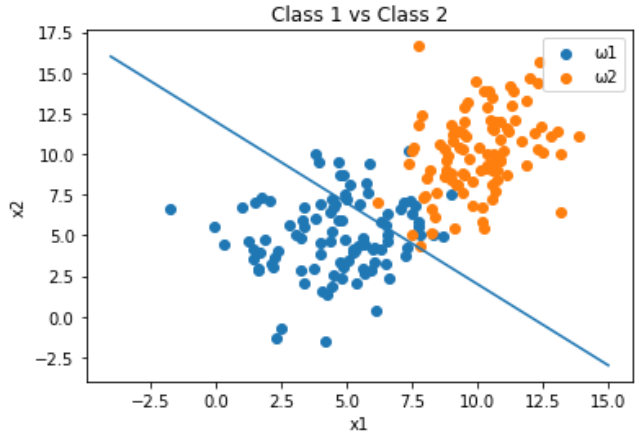
a. X1 is more discriminatory than x2. As you can see from the histograms, there is very little overlay between class 1 and class 2. However, for feature 2, while the overlapping is still relatively small, it is significantly larger than for feature 1.



b. The error rate based on this data set is **.05.**



c. The error rate based on this data set is **.135.**



d. The classifier that performed better for this data set was the first one. While they both had relatively low error rates, the first one of 5% is still significantly smaller than 13.5%, making it the better classifier.

Appendix

- Question 1 code

```
import pprint
import scipy.spatial as sc
import matplotlib.pyplot as plt
import pandas as pd

DIMENSIONS = 8
PATTERNS = 48
TITLES = ["Upper Left Boundary", "Lower Right Boundary", \
          "Upper Right Boundary", "Lower Left Boundary", \
          "Middle Left Boundary", "Middle Right Boundary", \
          "Middle Upper Boundary", "Middle Lower Boundary"]
NAME_MAP = {1: "I", 2: "M", 3: "O", 4: "X"}


def mean_pattern_vector(data):
    """Computes the sum of the data points"""
    d_avgs = {}
    print("\nMean Pattern Vectors:\n")
    for c, vals in data.items():
        d_sum = [0] * DIMENSIONS
        for d in vals:
            for i in range(DIMENSIONS):
                d_sum[i] += d[i]

        # Computes the average
        d_avg = [round(el / PATTERNS, 3) for el in d_sum]
        d_avgs[c] = d_avg
        print("Class " + str(c) + ":", d_avg)
    return d_avgs


def draw_scatter_3d(x, y, z, cols, data):
    """This function draws a 3d scatter plot"""
    patterns = []
    for c, vals in data.items():
        for v in vals:
            patterns.append([NAME_MAP[c], v[x], v[y], v[z]])

    df = pd.DataFrame(patterns, columns=cols)

    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    ax.scatter(list(df[cols[1]])[0:48], list(df[cols[2]])[0:48],
               list(df[cols[3]])[0:48], c='blue', label="I", alpha=.4)
    ax.scatter(list(df[cols[1]])[48:96], list(df[cols[2]])[48:96],
               list(df[cols[3]])[48:96], c='orange', label="M", alpha=.4)
    ax.scatter(list(df[cols[1]])[96:144], list(df[cols[2]])[96:144],
               list(df[cols[3]])[96:144], c='green', label="O", alpha=.4)
    ax.scatter(list(df[cols[1]])[144:192], list(df[cols[2]])[144:192],
               list(df[cols[3]])[144:192], c='red', label="X", alpha=.4)

    ax.set_xlabel(cols[1] + " Distance")
    ax.set_ylabel(cols[2] + " Distance")
    ax.set_zlabel(cols[3] + " Distance")
    ax.set_title("Boundary Distances")
    ax.legend()


def draw_scatter(x, y, cols, data):
    """This function draws a scatter plot for 2 features, grouped by class"""
    patterns = []
    for c, vals in data.items():
        for v in vals:
            patterns.append([NAME_MAP[c], v[x], v[y]])
```

```
    df = pd.DataFrame(patterns, columns=cols)

    fig, ax = plt.subplots()
    ax.scatter(x=list(df[cols[1]])[0:48], y=list(df[cols[2]])[0:48], c='blue',
               label="I", alpha=.4)
    ax.scatter(x=list(df[cols[1]])[48:96], y=list(df[cols[2]])[48:96],
               c='orange', label="M", alpha=.4)
    ax.scatter(x=list(df[cols[1]])[96:144], y=list(df[cols[2]])[96:144],
               c='green', label="O", alpha=.4)
    ax.scatter(x=list(df[cols[1]])[144:192], y=list(df[cols[2]])[144:192],
               c='red', label="X", alpha=.4)

    ax.set_xlabel(cols[1] + " Distance")
    ax.set_ylabel(cols[2] + " Distance")
    ax.set_title("Boundary Distances")
    ax.legend()


def euclidean_distance(data, d_avgs):
    """Computes the Euclidean Distance Formula"""
    print("\nFarthest Patterns From The Mean:\n")
    for c, vals in data.items():
        max_dist, max_num = 0, -1
        for i, d in enumerate(vals):
            dist = sc.distance.euclidean(d, d_avgs[c])
            if dist > max_dist:
                max_dist = dist
                max_num = i
        print("Class " + str(c) + ":", vals[max_num], \
              "(Distance = " + str(round(max_dist, 3)) +")")


def main():
    # Initializes the data
    file = open("imox_data.txt", "r")
    data = {1: [], 2: [], 3: [], 4: []}
    d_avgs = {}

    # Parses the file
    for line in file:
        line = line.strip().split()
        line = [int(float(d)) for d in line]
        data[line[-1]].append(line[:-1])

    d_avgs = mean_pattern_vector(data)
    euclidean_distance(data, d_avgs)

    # Converts the data into pandas dataframes
    for i in range(DIMENSIONS):
        feature_cols = {"I": [], "M": [], "O": [], "X": []}

        # Gets the features
        for c, vals in data.items():
            for v in vals:
                feature_cols[NAME_MAP[c]].append(v[i])

        # Plots the histograms for each of the 8 features
        df = pd.DataFrame(feature_cols)
        plot = df.plot.hist(bins=12, alpha=.5, title=TITLES[i])
        plot.set_xlabel("Distance")

    draw_scatter(0, 1, ["Letter", "Upper Left", "Lower Right"], data)
    draw_scatter(2, 3, ["Letter", "Upper Right", "Lower Left"], data)
    draw_scatter_3d(0, 1, 3, ["Letter", "Upper Left", "Lower Right", \
                              "Lower Left"], data)


if __name__ == "__main__":
    main()
```

- Question 6 code

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math


def plot_histograms(data):
    """This function plots the histograms for p(x1|w) and p(x2|w) for each
    class"""
    data_w = {}
    for i in range(2):
        for c, vals in data.items():
            data_w["\u03C9" + str(c)] = [v[i] for v in vals]

        # Plots the histograms
        df = pd.DataFrame(data_w)
        mini = math.floor(min(df["\u03C91"].min(), df["\u03C92"].min()))
        maxi = math.ceil(max(df["\u03C91"].max(), df["\u03C92"].max()))
        plot = df.plot.hist(bins=(maxi-mini), alpha=.5,
                            title="Feature " + str(i + 1), range=(mini, maxi))
        plot.set_xlabel("x" + str(i + 1))
        data_w = {}


def error_rate(data, con):
    """This function finds the percentage of misclassified datapoints"""
    length, incorrect = 0, 0
    for c, vals in data.items():
        length += len(vals)
        for v in vals:
            # Where doe the point fall??
            res = v[0] + v[1] - con
            what_class = 1 if res < 0 else 2

            # Is the result correct
            if what_class != c:
                incorrect += 1

    return incorrect / length

def plot_scatter(data, con):
    """This function plots a scatter plot of the 2 data points for each class"""
    fig, ax = plt.subplots()

    # Plots the 2 points
    for c, vals in data.items():
        ax.scatter(*zip(*vals), label="\u03C9" + str(c))

    # Plot the decision boundary
    x = np.arange(-4, 16)
    ax.plot(x, con-x)

    # Plot nicities
    ax.legend()
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_title("Class 1 vs Class 2")

    print("Error Rate for Decision Boundary:", error_rate(data, con))


def main():
    file = open("data.txt", "r")
    data = {1: [], 2: []}

    # Parses the file
    for line in file:
        line = line.strip().split()
        data[int(line[2])].append([float(line[0]), float(line[1])])
```

```
    plot_histograms(data)
    plot_scatter(data, 15)
    plot_scatter(data, 12)


if __name__ == "__main__":
    main()
```