CS178 Homework 1

Instructions

Welcome to CS 178!

This homework (and many subsequent ones) will involve data analysis and reporting on methods and results using Python code. You will submit a **single PDF file** that contains everything to Gradescope. This includes any text you wish to include to describe your results, the complete code snippets of how you attempted each problem, any figures that were generated, and scans of any work on paper that you wish to include. It is important that you include enough detail that we know how you solved the problem, since otherwise we will be unable to grade it.

Your homeworks will be given to you as Jupyter notebooks containing the problem descriptions and some template code that will help you get started. You are encouraged to modify these starter Jupyter notebooks to complete your assignment and to write your report. You may add additional cells (containing either code or text) as needed. This will help you not only ensure that all of the code for the solutions is included, but also will provide an easy way to export your results to a PDF file (for example, doing *print preview* and *printing to pdf*). I recommend liberal use of Markdown cells to create headers for each problem and sub-problem, explaining your implementation/answers, and including any mathematical equations. For parts of the homework you do on paper, scan it in such that it is legible (there are a number of free Android/iOS scanning apps, if you do not have access to a scanner), and include it as an image in the Jupyter notebook.

Several problems in this assignment require you to create plots. Use `matplotlib.pyplot` to do this, which is already imported for you as `plt`. Do not use any other plotting libraries, such as `pandas` or `seaborn`. Unless you are told otherwise, you should call `pyplot` plotting functions with their default arguments.

If you have any questions/concerns about the homework problems or using Jupyter notebooks, ask us on EdD. If you decide not to use Jupyter notebooks, but go with Microsoft Word or Latex to create your PDF file, make sure that all of the answers can be generated from the code snippets included in the document. Note that this may make grading more difficult.

Summary of Assignment: 100 total points

- Problem 1: Exploring a NYC Housing Dataset (30 points)
  - Problem 1.1: Numpy Arrays (5 points)
  - Problem 1.2: Feature Statistics (5 points)
  - Problem 1.3: Logical Indexing (5 points)
  - Problem 1.4: Histograms (5 points)
  - Problem 1.5: Scatter Plots (10 points)
- Problem 2: Building a Nearest Centroid Classifier (50 points)
  - Problem 2.1: Implementing Nearest Centroids (30 points)
  - Problem 2.2: Evaluating Nearest Centroids (20 points)
- Problem 3: Decision Boundaries (15 points)
  - Problem 3.1: Visualize 2D Centroid Classifier (5 points)
  - Problem 3.2: Visualize 2D Gaussian Bayes Classifier (5 points)
  - Problem 3.3: Analysis (5 points)
- Statement of Collaboration (5 points)

Before we get started, let's import some libraries that you will make use of in this assignment. Make sure that you run the code cell below in order to import these libraries.

Important: In the code block below, we set `seed=123`. This is to ensure your code has reproducible results and is important for grading. Do not change this. If you are not using the provided Jupyter notebook, make sure to also set the random seed as below.

In [1]:
```
# If you haven't installed numpy, pyplot, scikit, etc., do so:
!pip install -U scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\25181\anaconda3\lib\site-
packages (1.4.2)
Collecting scikit-learn
  Downloading scikit_learn-1.6.1-cp312-cp312-win_amd64.whl.metadata (15 kB)
Requirement already satisfied: numpy>=1.19.5 in c:\users\25181\anaconda3\lib\site
-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in c:\users\25181\anaconda3\lib\site-
packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\25181\anaconda3\lib\site
-packages (from scikit-learn) (1.4.2)
Collecting threadpoolctl>=3.1.0 (from scikit-learn)
  Using cached threadpoolctl-3.5.0-py3-none-any.whl.metadata (13 kB)
Downloading scikit_learn-1.6.1-cp312-cp312-win_amd64.whl (11.1 MB)
   ---------------------------------------- 0.0/11.1 MB ? eta -:--:--
   ---------------------------------------- 0.0/11.1 MB 1.3 MB/s eta 0:00:09
   - -------------------------------------- 0.3/11.1 MB 3.5 MB/s eta 0:00:04
   --- ------------------------------------ 1.0/11.1 MB 8.7 MB/s eta 0:00:02
   ------ --------------------------------- 1.9/11.1 MB 11.9 MB/s eta 0:00:01
   --------- ------------------------------ 2.7/11.1 MB 13.4 MB/s eta 0:00:01
   ------------ --------------------------- 3.5/11.1 MB 13.8 MB/s eta 0:00:01
   --------------- ------------------------ 4.4/11.1 MB 14.7 MB/s eta 0:00:01
   ----------------- ---------------------- 4.8/11.1 MB 14.0 MB/s eta 0:00:01
   -------------------- ------------------- 5.6/11.1 MB 14.9 MB/s eta 0:00:01
   ---------------------- ----------------- 6.3/11.1 MB 14.4 MB/s eta 0:00:01
   -------------------------- ------------- 7.3/11.1 MB 15.1 MB/s eta 0:00:01
   ----------------------------- ---------- 8.1/11.1 MB 15.3 MB/s eta 0:00:01
   ------------------------------- -------- 8.8/11.1 MB 15.2 MB/s eta 0:00:01
   --------------------------------- ------ 9.2/11.1 MB 15.5 MB/s eta 0:00:01
   ------------------------------------ --- 10.1/11.1 MB 15.3 MB/s eta 0:00:01
   -------------------------------------- - 10.7/11.1 MB 16.8 MB/s eta 0:00:01
   ---------------------------------------- 11.1/11.1 MB 15.6 MB/s eta 0:00:00
Downloading threadpoolctl-3.5.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, scikit-learn
  Attempting uninstall: threadpoolctl
    Found existing installation: threadpoolctl 2.2.0
    Uninstalling threadpoolctl-2.2.0:
      Successfully uninstalled threadpoolctl-2.2.0
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.4.2
    Uninstalling scikit-learn-1.4.2:
      Successfully uninstalled scikit-learn-1.4.2
Successfully installed scikit-learn-1.6.1 threadpoolctl-3.5.0
```

In [5]:

```python
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_openml
from sklearn.neighbors import KNeighborsClassifier, NearestCentroid
from sklearn.metrics import accuracy_score, zero_one_loss, confusion_matrix, Con
from sklearn.model_selection import train_test_split
from sklearn.inspection import DecisionBoundaryDisplay

import requests           # we'll use these for reading data from a url
from io import StringIO

# Fix the random seed for reproducibility
# !! Important !! : do not change this
seed = 123
np.random.seed(seed)
```

Problem 1: Exploring a NYC Housing Dataset

In this problem, you will explore some basic data manipulation and visualizations with a small dataset of real estate prices from NYC. For every datapoint, we are given several real-valued features which will be used to predict the target variable, y, representing in which borough the property is located. Let's first load in the dataset by running the code cell below:

In [7]:
```
# Load the features and labels from an online text file
url = 'https://ics.uci.edu/~ihler/classes/cs178/data/nyc_housing.txt'
with requests.get(url) as link:
    datafile = StringIO(link.text)
    nych = np.genfromtxt(datafile,delimiter=',')
    nych_X, nych_y = nych[:,:-1], nych[:,-1]
```

These data correspond to (a small subset of) property sales in New York in 2014. The target, y, represents the borough in which the property was located (0: Manhattan; 1: Bronx; 2: Staten Island). The observed features correspond to the property size (square feet), price (USD), and year built; the first two features have been log2-transformed (e.g., $x_1 = \log_2(\text{size})$) for convenience.

Problem 1.1 (5 points): Numpy Arrays

The variable `nych_X` is a numpy array containing the feature vectors in our dataset, and `nych_y` is a numpy array containing the corresponding labels.

- What is the shape of `nych_X` and `nych_y`? (Hint)
- How many datapoints are in our dataset, and how many features does each datapoint have?
- How many different classes (i.e. labels) are there?
- Print rows 3, 4, 5, and 6 of the feature matrix and their corresponding labels. Since Python is zero-indexed, we will count our rows starting at zero -- for example, by "row 0" we mean `nych_X[0, :]`, and "row 1" means `nych_X[1, :]`, etc. (Hint: you can do this in two lines of code with slicing).

In [44]:
```
print("shape of nych_X:", nych_X.shape)
print("shape of nych_y:", nych_y.shape)

num_classes = len(np.unique(nych_y))
print(f"Number of classes: {num_classes}")

row = [3,4,5,6]

print("feature matrix:",nych_X[row,:])
print("labels:",nych_y[row])
```

```
shape of nych_X: (300, 3)
shape of nych_y: (300,)
Number of classes: 3
feature matrix: [[  11.839204   19.416995 1980.      ]
 [  18.517396   25.357833 1973.      ]
 [  11.050529   19.041723 2014.      ]
 [  17.255803   26.280297 1917.      ]]
labels: [2. 1. 2. 0.]
```

In [30]:
```
#We have 300 datapoints, 3 features for each datapoint.
#we have 3 different class
```

Problem 1.2 (5 points): Feature Statistics

Let's compute some statistics about our features. You are allowed to use `numpy` to help you with this problem -- for example, you might find some of the `numpy` functions listed here or here useful.

- Compute the mean, variance, and standard deviation of each feature.
- Compute the minimum and maximum value for each feature.

Make sure to print out each of these values, and indicate clearly which value corresponds to which computation.

In [92]:
```
#from left to right, corresponding to property size (square feet), price (USD),
#mean
print("mean",np.mean(nych_X,axis=0))
#variance
print("variance",np.var(nych_X,axis=0))
#std
print("std",np.std(nych_X,axis=0))

#min
print("min",np.min(nych_X,axis=0))
#max
print("max",np.max(nych_X,axis=0))
```

```
mean [  14.11839247   21.90711615 1946.35333333]
variance [   6.60022492    8.87193012 1253.08182222]
std [ 2.56909029  2.97857854 35.39889578]
min [  10.366322   16.872675 1893.      ]
max [  20.152714   29.123861 2014.      ]
```

Problem 1.3 (5 points): Logical Indexing

Use numpy's logical (boolean) indexing to extract only those data corresponding to $y = 0$ (Manhattan). Then, compute the mean and standard deviation of *only these* data points. Then, do the same for $y = 1$ (Bronx).

Again, print out each of these vectors and indicate clearly which value corresponds to which computation.

In [96]:
```
#from left to right, corresponding to property size (square feet), price (USD),
Manhattan = nych_y==0
M_D=nych_X[Manhattan]
```

```
print("Manhattan mean",np.mean(M_D,axis=0))
print("Manhattan sd",np.std(M_D,axis=0))

Bronx = nych_y==1
B_D=nych_X[Bronx]
print("Bronx mean",np.mean(B_D,axis=0))
print("Bronx sd",np.std(B_D,axis=0))
```

```
Manhattan mean [  16.1489863    25.07251963 1926.94      ]
Manhattan sd [ 2.19416051  2.09812353 28.14562843]
Bronx mean [  14.60837771   21.4446885  1935.29      ]
Bronx sd [ 1.89678446  1.99063026 22.96619037]
```

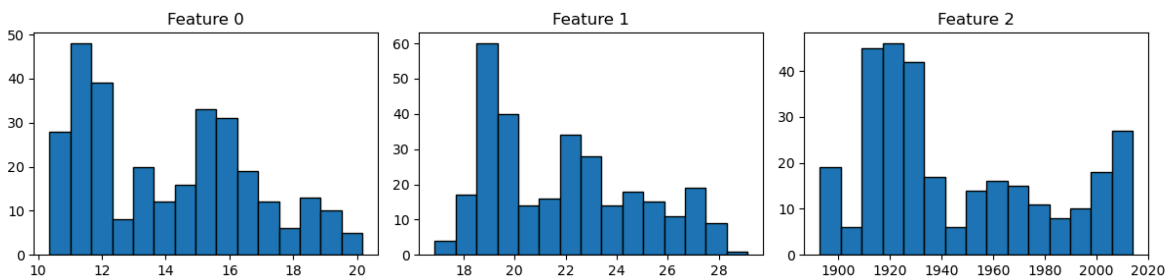
Problem 1.4 (5 points): Feature Histograms

Now, you will visualize the distribution of each feature with histograms. Use `matplotlib.pyplot` to do this, which is already imported for you as `plt`. Do not use any other plotting libraries, such as `pandas` or `seaborn`.

- For every feature in `nych_X`, plot a histogram of the values of the feature. Your plot should consist of a grid of subplots with 1 row and 3 columns.
- Include a title above each subplot to indicate which feature we are plotting. For example, you can call the first feature "Feature 0", the second feature "Feature 1", etc.

Some starter code is provided for you below. (Hint: `axes[0].hist(...)` will create a histogram in the first subplot.)

In [9]:
```
# Create a figure with 1 row and 3 columns
fig, axes = plt.subplots(1, 3, figsize=(12, 3))

### YOUR CODE STARTS HERE ###
# Loop
# property size (square feet), price (USD), and year built

for i in range(nych_X.shape[1]):
    #iterate among each feature
    axes[i].hist(nych_X[:, i], bins=15, edgecolor='black')
    #set the title
    axes[i].set_title(f"Feature {i}")


###  YOUR CODE ENDS HERE  ###

fig.tight_layout()
```



Problem 1.5 (10 points): Feature Scatter Plots

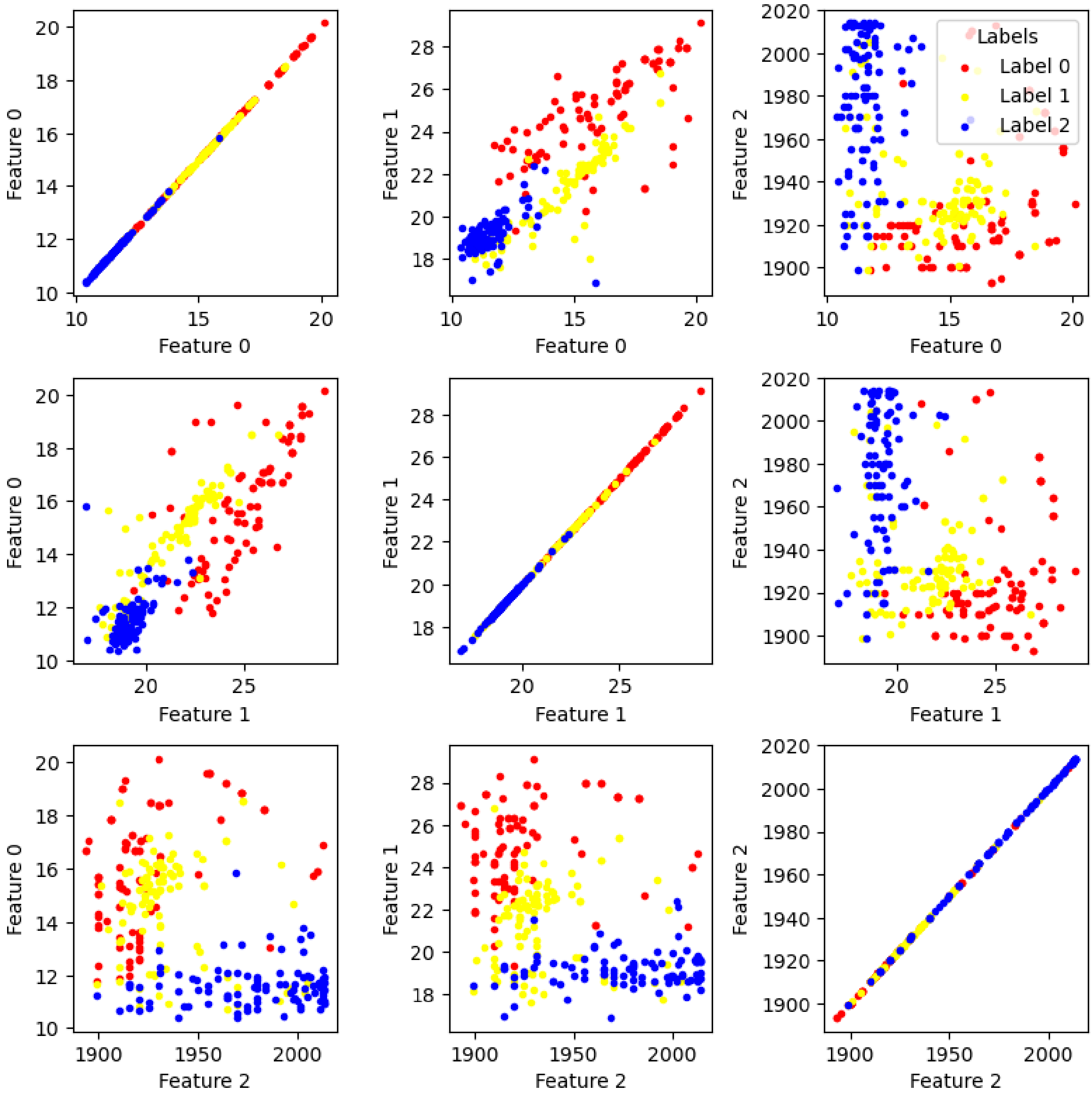
To help further visualize the NYC-Housing datset, you will now create several scatter plots of the features. Use `matplotlib.pyplot` to do this, which is already imported for you as `plt`. Do not use any other plotting libraries, such as `pandas` or `seaborn`.

- For every pair of features in `nych_X`, plot a scatter plot of the feature values, colored according to their labels. For example, plot all data points with $y = 0$ as blue, $y = 1$ as green, etc. Your plot should be a grid of subplots with 3 rows and 3 columns. (Hint: `axes[0, 0].scatter(...)` will create a scatter plot in the first column and first row).
- Include an x-label and a y-label on each subplot to indicate which features we are plotting. For example, you can call the first feature "Feature 0", the second feature "Feature 1", etc. (Hint: `axes[0, 0].set_xlabel(...)` might help you with the first subplot.)

Some starter code is provided for you below.

In [131…

```
# Create a figure with 3 rows and 3 columns
fig, axes = plt.subplots(3, 3, figsize=(8, 8))

### YOUR CODE STARTS HERE ###
# property size (square feet), price (USD), and year built
label_colors = ['red', 'yellow', 'blue']

for i in range(nych_X.shape[1]):
    for j in range(nych_X.shape[1]):
        #00 01 02 11... compare between features
        ax = axes[i, j]
        for label in np.unique(nych_y):
            #plot in different color based on y
            mask = nych_y == label
            nych_X[label_mask][:,i]
            ax.scatter(nych_X[mask][:,i], nych_X[mask][:,j],
                       label=f"Label {int(label)}", s=9,
                       c=label_colors[int(label)])

            ax.set_xlabel(f"Feature {i}")
            ax.set_ylabel(f"Feature {j}")
        #label for plot
        if i == 0 and j == 2:
            ax.legend(loc="upper right", title="Labels")




###  YOUR CODE ENDS HERE  ###

fig.tight_layout()
```



Problem 2: Nearest Centroid Classifiers

In this problem, you will implement a nearest centroid classifier and train it on the NYC data.

Problem 2.1 (30 points): Implementing a Nearest Centroid Classifier

In the code given below, we define the class `NearestCentroidClassifier` which has an unfinished implementation of a nearest centroid classifier. For this problem, you will complete this implementation. Your nearest centroid classifier will use the Euclidean distance, which is defined for two feature vectors x and x' as

$$d_E(x, x') = \sqrt{\sum_{j=1}^{d}(x_j - x'_j)^2}.$$

- Implement the method `fit`, which takes in an array of features `X` and an array of labels `y` and trains our classifier. You should store your computed centroids in the list `self.centroids`, and their y values in `self.classes_` (whose name is chosen to match `sklearn` conventions).
- Test your implementation of `fit` by training a `NearestCentroidClassifier` on the NYC data, and printing out the list of centroids. (These should match the means in Problem 1.3.)
- Implement the method `predict`, which takes in an array of feature vectors `X` and predicts their class labels based on the centroids you computed in the method `fit`.
- Print the predicted labels (using your `predict` function) and the true labels for the first ten data points in the NYCH dataset. Make sure to indicate which are the predicted labels and which are the true labels.

You are allowed to modify the given code as necessary to complete the problem, e.g. you may create helper functions.

In [149…

```
class NearestCentroidClassifier:
    def __init__(self):
        # A list containing the centroids; to be filled in with the fit method.
        self.centroids = []

    def fit(self, X, y):
        """ Fits the nearest centroid classifier with training features X and tr

        X: array of training features; shape (m,n), where m is the number of dat
            and n is the number of features.
        y: array training labels; shape (m, ), where m is the number of datapoin

        """
        # First, identify what possible classes exist in the training data set:
        self.classes_ = np.unique(y)

        ### YOUR CODE STARTS HERE ###
        # Hint: you should append to self.centroids with the corresponding centr
        # The centroid (mean vector) can be computed in a similar way to P1.2, f

        for cls in self.classes_:
            #all data belonging to one class
            X_cls = X[y == cls]

            centroid = np.mean(X_cls,axis=0)
            self.centroids.append(centroid)



        ###   YOUR CODE ENDS HERE   ###


    def predict(self, X):
```

```
        """ Makes predictions with the nearest centroid classifier on the featur

        X: array of features; shape (m,n), where m is the number of datapoints,
            and n is the number of features.

        Returns:
        y_pred: a numpy array of predicted labels; shape (m, ), where m is the n
        """
        ### YOUR CODE STARTS HERE ###
        # Hint: find the distance from each x[i] to the centroids, and predict t
        y_pred = []
        for ele in X:
            dis = [np.linalg.norm(ele - centroid) for centroid in self.centroids
            nearest_class = self.classes_[np.argmin(dis)]
            y_pred.append(round(nearest_class))



        ###  YOUR CODE ENDS HERE  ###

        return y_pred
```

Here is some code illustrating how to use your `NearestCentroidClassifier`. You can run this code to fit your classifier and to plot the centroids. You should write your implementation above such that you don't need to modify the code in the next cell. As a sanity check, you should find that the 3rd centroid (for Staten Island) has a "year build" coordinate value of around 1976.8 (i.e., the rightmost column).

In [151…

```
nc_classifier = NearestCentroidClassifier()  # Create a NearestCentroidClassifie
nc_classifier.fit(nych_X, nych_y)            # Fit to the NYC training data

print(nc_classifier.centroids)
```

```
[array([  16.1489863 ,   25.07251963, 1926.94      ]), array([  14.60837771,   2
1.4446885 , 1935.29      ]), array([  11.59781341,   19.20414033, 1976.83
])]
```

In [153…

```
# Print the predicted and true labels for the first ten data points in the NYCH
### YOUR CODE STARTS HERE ###

print("Predicted:",nc_classifier.predict(nych_X)[:10])

print("True:",nych_y[:10])



###  YOUR CODE ENDS HERE  ###
```

```
Predicted: [0, 2, 0, 2, 2, 2, 0, 0, 2, 1]
True: [1. 2. 0. 2. 1. 2. 0. 0. 1. 1.]
```

Problem 2.2 (20 points): Evaluating the Nearest Centroids Classifier

Now that you've implemented the nearest centroid classifier, it is time to evaluate its performance.

- Write a function `compute_error_rate` that computes the error rate (fraction of misclassifications) of a model's predictions. That is, your function should take in an array of true labels y and an array of predicted labels `y_pred`, and return the error rate of the predictions. You may use `numpy` to help you do this, but do not use `sklearn` or any other machine learning libraries.
- Write a function `compute_confusion_matrix` that computes the confusion matrix of a model's predictions. That is, your function should take in an array of true labels `y` and an array of predicted labels `y_pred`, and return the corresponding $C \times C$ confusion matrix as a numpy array, where C is the number of classes. You may use `numpy` to help you do this, but do not use `sklearn` or any other machine learning libraries.
- Verify that your implementations of `NearestCentroidClassifier`, `compute_error_rate`, and `compute_confusion_matrix` are correct. To help you do this, you are given the functions `eval_sklearn_implementation` and `eval_my_implementation`. The function `eval_sklearn_implementation` will use the relevant `sklearn` implementations to compute the error rate and confusion matrix of a nearest centroid classifier. The function `eval_my_implementation` will do the same, but using your implementations. If your code is correct, the outputs of the two functions should be the same.

In [155…

```
def compute_error_rate(y, y_pred):
    """ Computes the error rate of an array of predictions.

    y: true labels; shape (n, ), where n is the number of datapoints.
    y_pred: predicted labels; shape (n, ), where n is the number of datapoints.

    Returns:
    error rate: the error rate of y_pred compared to y; scalar expressed as a de
    """
    ### YOUR CODE STARTS HERE ###
    incorrect = np.sum(y != y_pred)
    total = len(y)
    error_rate = incorrect / total









    ###  YOUR CODE ENDS HERE  ###

    return error_rate
```

In [176…

```
def compute_confusion_matrix(y, y_pred):
    """ Computes the confusion matrix of an array of predictions.

    y: true labels; shape (n, ), where n is the number of datapoints.
    y_pred: predicted labels; shape (n, ), where n is the number of datapoints.

    Returns:
    confusion_matrix: a numpy array corresponding to the confusion matrix from y
        where C is the number of unique classes.  The (i,j)th entry is the numbe
        that are classified as being from class j.
```

```
    """

    ### YOUR CODE STARTS HERE ###

    # number of unique classes
    classes = np.unique(np.concatenate((y, y_pred)))
    num_classes = len(classes)

    #init
    confusion_matrix = np.zeros((num_classes, num_classes), dtype=int)

    #fill
    for true_y, pred_y in zip(y, y_pred):
        confusion_matrix[int(true_y),int(pred_y)] += 1

    return confusion_matrix




    ###  YOUR CODE ENDS HERE  ###

    return confusion_matrix
```

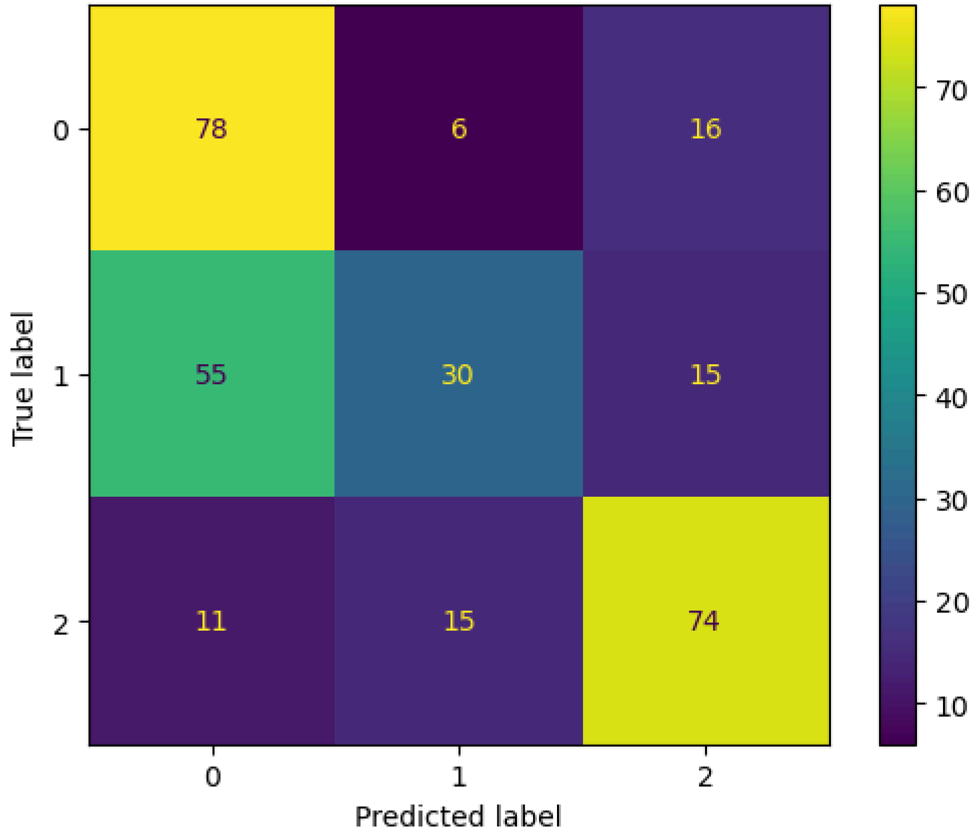
You can run the two code cells below to compare your answers to the implementations in `sklearn`. If your answers are correct, the outputs of these two functions should be the same. Do not modify the functions `eval_sklearn_implementation` and `eval_my_implementation`, but make sure that you read and understand this code.

In [160…

```
###############################################
### Results with the sklearn implementation ###
###############################################

def eval_sklearn_implementation(X, y):
    # Nearest centroid classifier implemented in sklearn
    sklearn_nearest_centroid = NearestCentroid()

    # Fit on training dataset
    sklearn_nearest_centroid.fit(X, y)

    # Make predictions on training and testing data
    sklearn_y_pred = sklearn_nearest_centroid.predict(X)

    # Evaluate error rate using the sklearn function zero_one_loss
    sklearn_err = zero_one_loss(y, sklearn_y_pred)

    print(f'Sklearn Results:')
    print(f'--- Error Rate (0/1): {sklearn_err}')

    # Evaluate confusion matrix using the sklearn function confusion_matrix
    sklearn_cm = confusion_matrix(y, sklearn_y_pred)
    sklearn_disp = ConfusionMatrixDisplay(confusion_matrix = sklearn_cm)
    sklearn_disp.plot();
```

```
# Call the function
eval_sklearn_implementation(nych_X, nych_y)
```

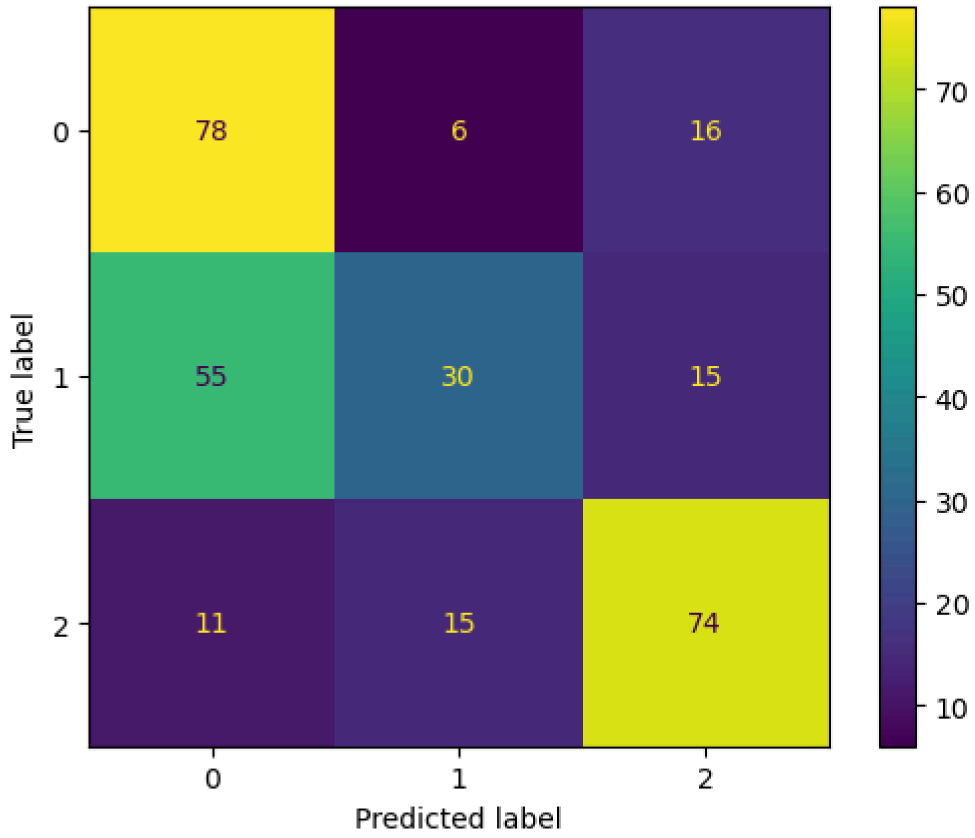
```
Sklearn Results:
--- Error Rate (0/1): 0.39333333333333333
```



In [178…

```
###########################################
### Results with your implementation ###
###########################################

def eval_my_implementation(X, y):
    # Now test your implementation of NearestCentroidClassifier
    nearest_centroid = NearestCentroidClassifier()

    # Fit on training dataset
    nearest_centroid.fit(X, y)

    # Make predictions on training and testing data
    y_pred = nearest_centroid.predict(X)

    # Evaluate error rate using your function compute_error_rate
    err = compute_error_rate(y, y_pred)

    print(f'Your Results:')
    print(f'--- Error Rate (0/1): {err}')

    # Evaluate confusion matrix using your function compute_confusion_matrix
    cm = compute_confusion_matrix(y, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix = cm)
    disp.plot();


# Call the function
eval_my_implementation(nych_X, nych_y)
```

```
Your Results:
--- Error Rate (0/1): 0.3933333333333333
```



Problem 3: Decision Boundaries

For the final problem of this homework, you will visualize the decision function and decision boundary of your nearest centroid classifier on 2D data, and compare it to the similar but more flexible Gaussian Bayes classifier discussed in class. Code for drawing the decision function (which simply evaluates the prediction on a grid) and superimposing the data points is provided.

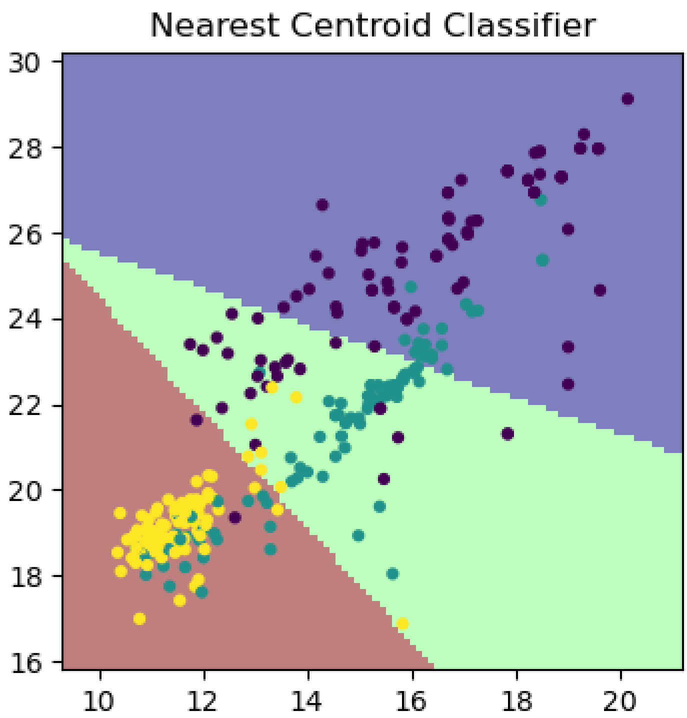
Problem 3.1 (5 points): Visualize 2D Centroid Classifier

We will use only the first two features of the NYCH data set, to facilitate visualization.

In [188…

```
# Plot the decision boundary for your classifier

# Some keyword arguments for making nice looking plots.
plot_kwargs = {'cmap': 'jet',     # another option: viridis
               'response_method': 'predict',
               'plot_method': 'pcolormesh',
               'shading': 'auto',
               'alpha': 0.5,
               'grid_resolution': 100}
```

```
figure, axes = plt.subplots(1, 1, figsize=(4,4))

learner = NearestCentroidClassifier()

### YOUR CODE STARTS HERE ###

nych_X2 = nych_X[:,:2]      # get just the first two features of X
learner.fit(nych_X2,nych_y)   # Fit "learner" to nych 2-feature data



###  YOUR CODE ENDS HERE  ###

DecisionBoundaryDisplay.from_estimator(learner, nych_X2, ax=axes, **plot_kwargs)
axes.scatter(nych_X2[:, 0], nych_X2[:, 1], c=nych_y, edgecolor=None, s=12)
axes.set_title(f'Nearest Centroid Classifier');
```



Problem 3.2 (5 points): Visualize a 2D Gaussian Bayes Classifier

In class, we discussed building a Bayes classifier using an estimate of the class-conditional probabilities $p(X|Y=y)$, for example, a Gaussian distribution. It turns out this is relatively easy to implement and fairly similar to your Nearest Centroid classifier (in fact, Nearest Centroid is a special case of this model).

An implementation of a Gaussian Bayes classifier is provided:

In [198…

```
class GaussianBayesClassifier:
    def __init__(self):
        """Initialize the Gaussian Bayes Classifier"""
        self.pY    = []          # class prior probabilities, p(Y=c)
        self.pXgY  = []          # class-conditional probabilities, p(X|Y=c)
        self.classes_ = []      # list of possible class values
```

```python
    def fit(self, X, y):
        """ Fits a Gaussian Bayes classifier with training features X and traini
            X, y : (m,n) and (m,) arrays of training features and target class v
        """
        from sklearn.mixture import GaussianMixture
        self.classes_ = np.unique(y)         # Identify the class labels; then
        for c in self.classes_:              # for each class:
            self.pY.append(np.mean(y==c))    #   estimate p(Y=c) (a float)
            model_c = GaussianMixture(1)     #
            model_c.fit(X[y==c,:])           #   and a Gaussian for p(X|Y=c)
            self.pXgY.append(model_c)        #

    def predict(self, X):
        """ Makes predictions with the nearest centroid classifier on the featur
            X : (m,n) array of features for prediction
            Returns: y : (m,) numpy array of predicted labels
        """
        pXY = np.stack(tuple(np.exp(p.score_samples(X)) for p in self.pXgY)).T
        pXY *= np.array(self.pY).reshape(1,-1)          # evaluate p(X=x|Y=c) * p
        pYgX = pXY/pXY.sum(1,keepdims=True)             # normalize to p(Y=c|X=x)
        return self.classes_[np.argmax(pYgX, axis=1)]  # find the max index & re
```

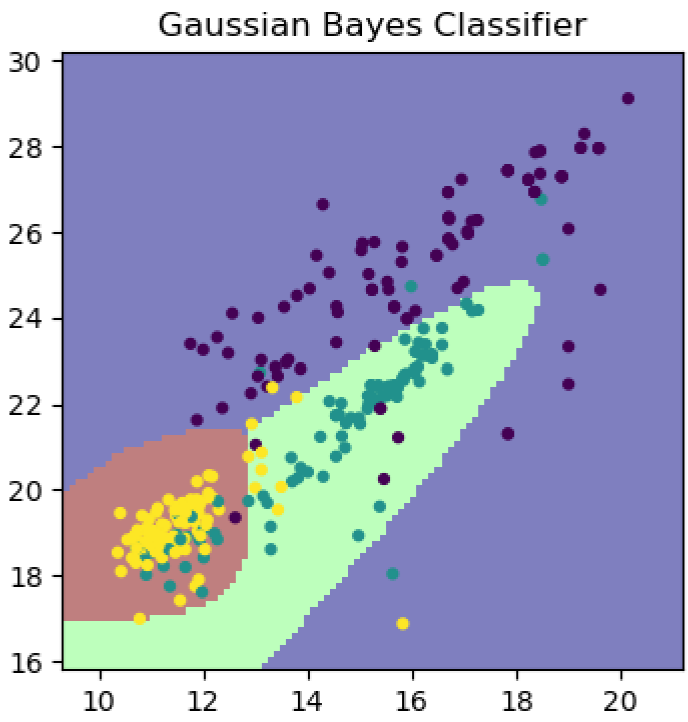
Using this learner, evaluate the predictions and error rate on the training data, and plot the decision boundary. The code should be the same as your Nearest Centroid, but using the new learner object.

In [200…

```python
# Plot the decision boundary for your classifier

# Some keyword arguments for making nice looking plots.
plot_kwargs = {'cmap': 'jet',     # another option: viridis
               'response_method': 'predict',
               'plot_method': 'pcolormesh',
               'shading': 'auto',
               'alpha': 0.5,
               'grid_resolution': 100}

figure, axes = plt.subplots(1, 1, figsize=(4,4))

learner = GaussianBayesClassifier()

### YOUR CODE STARTS HERE ###

nych_X2 =nych_X[:,:2]  # get just the first two features of X
learner.fit(nych_X2, nych_y) # Fit "learner" to nych 2-feature data

gbc_y_pred = learner.predict(nych_X2) # Use "learner" to predict on same data us

###  YOUR CODE ENDS HERE  ###

err = zero_one_loss(nych_y, gbc_y_pred)
print(f'Gaussian Bayes Error Rate (0/1): {err}')

DecisionBoundaryDisplay.from_estimator(learner, nych_X2, ax=axes, **plot_kwargs)
axes.scatter(nych_X2[:, 0], nych_X2[:, 1], c=nych_y, edgecolor=None, s=12)
axes.set_title(f'Gaussian Bayes Classifier');
```

```
Gaussian Bayes Error Rate (0/1): 0.15000000000000002
```

```
C:\Users\25181\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\25181\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\25181\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



Problem 3.3 (5 points): Analysis

Did the error increase or decrease? Why do you think this is?

In [ ]:
```
The error decrease from 0.393 to 0.15 when we switch from Nearest Centroids Clas
in our dataset, there are actually a lot of overlaps. Like its name, we would ex
as a apecial case of Gaussian Bayes Classifier. Thus, here Gaussian Bayes Classi
```

Statement of Collaboration (5 points)

It is **mandatory** to include a Statement of Collaboration in each submission, with respect to the guidelines below. Include the names of everyone involved in the discussions (especially in-person ones), and what was discussed.

All students are required to follow the academic honesty guidelines posted on the course website. For programming assignments, in particular, I encourage the students to organize (perhaps using EdD) to discuss the task descriptions, requirements, bugs in my code, and the relevant technical content before they start working on it. However, you should not discuss the specific solutions, and, as a guiding principle, you are not allowed to take anything written or drawn away from these discussions (i.e. no photographs of the blackboard, written notes, referring to EdD, etc.). Especially after you have started working on the assignment, try to restrict the discussion to EdD as much as possible, so that there is no doubt as to the extent of your collaboration.

In [ ]:
```
Jixuan Luo. only resources in ed and discussion material D00 && D01 are used as
```