CS273A Homework 1

Due: Monday, October 7th 2024 (11:59 PM)

Name: Langtian Qin
Student ID: 80107838
Email: langtiq@uci.edu

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 [5]:
```
# If you haven't installed numpy, pyplot, scikit, etc., do so:
!pip install -U scikit-learn
```

```
Requirement already satisfied: scikit-learn in /opt/anaconda3/lib/python3.
12/site-packages (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in /opt/anaconda3/lib/python
3.12/site-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /opt/anaconda3/lib/python3.
12/site-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /opt/anaconda3/lib/python
3.12/site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /opt/anaconda3/lib/
python3.12/site-packages (from scikit-learn) (3.5.0)
```

In [6]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_openml
from sklearn.neighbors import NearestCentroid
from sklearn.metrics import zero_one_loss, confusion_matrix, ConfusionMat
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 [8]:
```
# Load the features and labels from an online text file
url = 'https://ics.uci.edu/~ihler/classes/cs273/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.

Note: All answers will be written in blue font, and the code will be provided after the answer (if available).

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).

1) the shape of nych_X is (300, 3) and the shape of nych_Y is (300,).

In [14]:
```
# 1) What is the shape of nych_X and nych_y
print("shape of nych_X:", nych_X.shape, "\nshape of nych_Y:", nych_y.shap
```

```
shape of nych_X: (300, 3)
shape of nych_Y: (300,)
```

2) There are 300 data points are in our dataset, and each datapoint have 3 features.

3) There are 3 classes (label), i.e., 0: Manhattan; 1: Bronx; 2: Staten Island.

4) rows 3, 4, 5, and 6 of the feature matrix and their corresponding labels are shown below.

In [21]:
```
# row 3 -> nych_X[3,:] row 6 -> nych_X[6,:]
print("rows_features:\n", nych_X[3:7])
```

```
print("rows_labels:\n",nych_y[3:7])
```

```
rows_features:
 [[  11.839204   19.416995 1980.      ]
 [  18.517396   25.357833 1973.      ]
 [  11.050529   19.041723 2014.      ]
 [  17.255803   26.280297 1917.      ]]
rows_labels:
 [2. 1. 2. 0.]
```

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.

1) the mean, variance, and standard deviation of each feature are shown below.

In [25]:
```
# Calculte the mean
mean_x = np.mean(nych_X, 0)
print("mean value:", mean_x)

# Calculate the variance
var_x = np.var(nych_X, 0)
print("variance value:", var_x)

# Calcualte the standard deviation
std_x = np.std(nych_X, 0)
print("standard deviation value:", std_x)
```

```
mean value: [  14.11839247   21.90711615 1946.35333333]
variance value: [   6.60022492    8.87193012 1253.08182222]
standard deviation value: [ 2.56909029  2.97857854 35.39889578]
```

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.

1) the mean and standard deviation of Manhattan features (y=0) are shown below.

In [29]:
```
# Mean value of Manhattan features (y=0)
print("mean value of Manhattan features (y=0):", np.mean(nych_X[nych_y==0
```

```
# Standard deviation value of Manhattan features (y=0)
print("standard deviation value of Manhattan features (y=0):", np.std(nyc
```

```
mean value of Manhattan features (y=0): 656.05383531
standard deviation value of Manhattan features (y=0): 898.8082244444478
```

2) the mean and standard deviation of Bronx features (y=1) are shown below.

In [32]:
```
# Mean value of Bronx features (y=1)
print("mean value of Bronx features (y=1):", np.mean(nych_X[nych_y==1]))

# Standard deviation value of Bronx features (y=1)
print("standard deviation value of Bronx features (y=1):", np.std(nych_X[
```

```
mean value of Bronx features (y=1): 657.1143554033333
standard deviation value of Bronx features (y=1): 903.9096270337484
```

Problem 1.4 (5 points): Feature Histograms

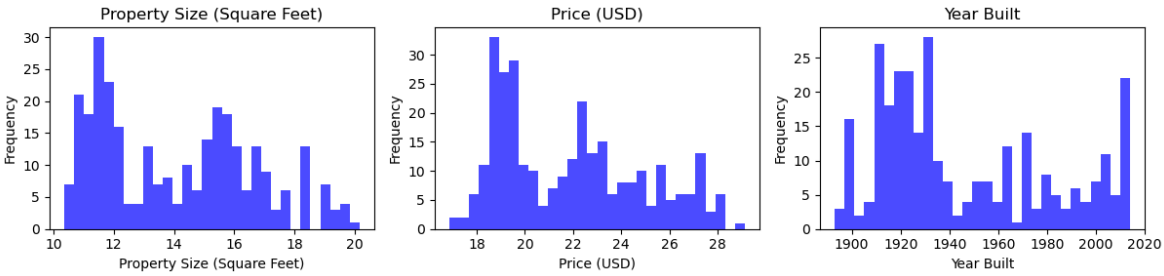
Now, you will visualize the distribution of each feature with histograms. Use `matplotlib.pyplot` to do this, which is already imported for you as `plt`. Do not use any other plotting libraries, such as `pandas` or `seaborn`.

- 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.)

1The distribution histograms of features are shown below respectively.)

In [46]:
```
# Create a figure with 1 row and 3 columns
fig, axes = plt.subplots(1, 3, figsize=(12, 3))

### YOUR CODE STARTS HERE ###
# feature names
features = ["Property Size (Square Feet)", "Price (USD)", "Year Built"]

# Plot the distribution of each feature
for i in range(3):
    axes[i].hist(nych_X[:, i], bins=30, color='b', alpha=0.7)
    axes[i].set_title(features[i])
    axes[i].set_xlabel(features[i])
    axes[i].set_ylabel('Frequency')

plt.tight_layout()
plt.show()

###  YOUR CODE ENDS HERE  ###

fig.tight_layout()
```



Problem 1.5 (5 points): Feature Scatter Plots

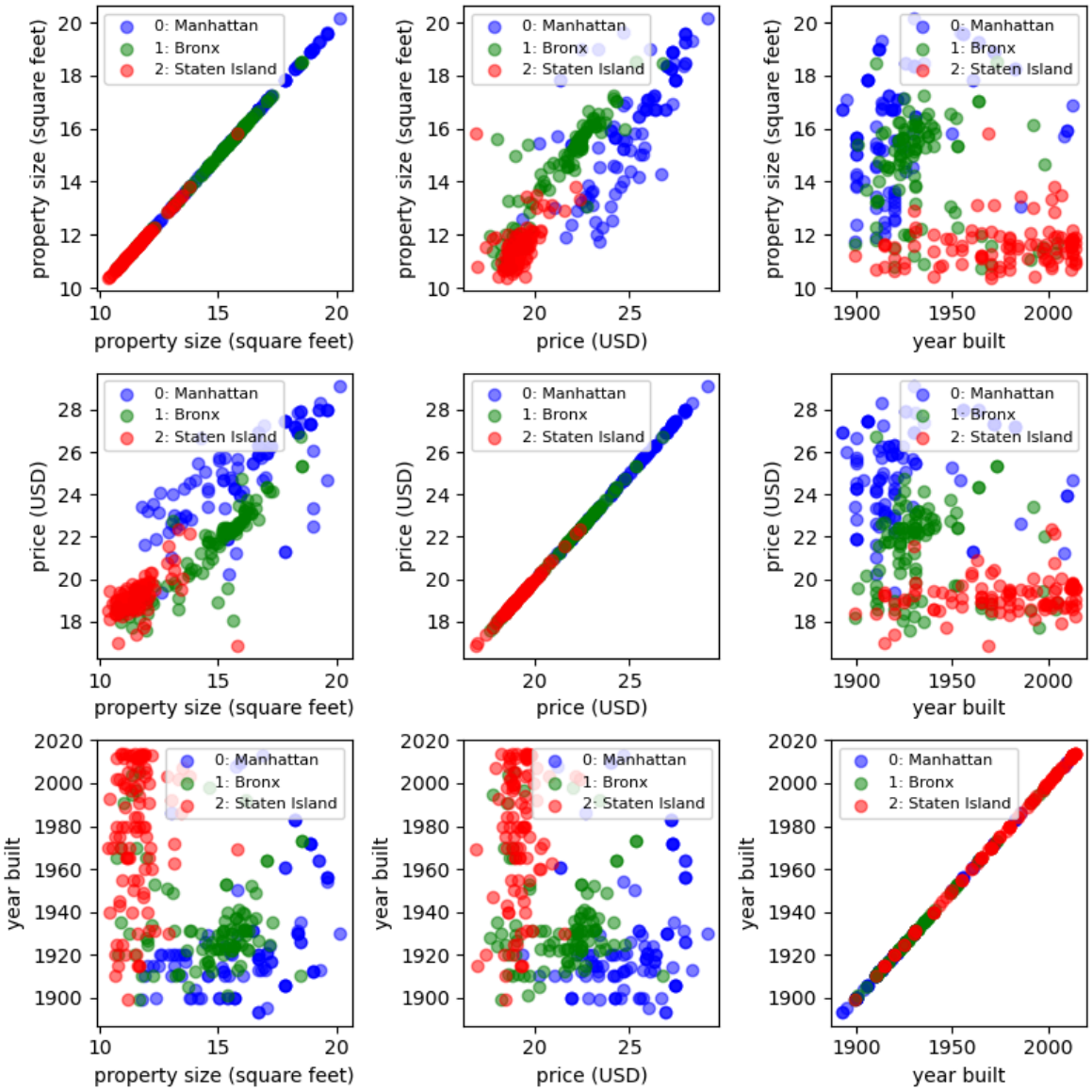
To help further visualize the NYC-Housing datset, you will now create several scatter plots of the features. Use `matplotlib.pyplot` to do this, which is already imported for you as `plt`. Do not use any other plotting libraries, such as `pandas` or `seaborn`.

- 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, with the plot in position (i, j) showing feature x_i versus x_j, with the class labels indicated by color. (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 [92]:
```
# Create a figure with 3 rows and 3 columns
fig, axes = plt.subplots(3, 3, figsize=(8, 8))

### YOUR CODE STARTS HERE ###
feature_names = ['property size (square feet)', 'price (USD)', 'year buil
num_features = len(feature_names)

# Use different colors for different labels
colors = ['blue', 'green', 'red']
labels = ['0', '1', '2']
labels_name = [ "0: Manhattan", "1: Bronx", "2: Staten Island"]

scatter_handles = []
# Plot scatter plots for each pair of features
for i in range(num_features):
    for j in range(num_features):
        ax = axes[i, j]
        for label in np.unique(nych_y):
            scatter = ax.scatter(nych_X[nych_y == label, j], nych_X[nych_
                                 label=labels_name[label.astype(int)], al
            scatter_handles.append(scatter)
        ax.set_xlabel(feature_names[j])
        ax.set_ylabel(feature_names[i])
        ax.legend(loc='best', fontsize=8)
```

```
###  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 (20 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 [98]:
```
class NearestCentroidClassifier:
    def __init__(self):
        # A list containing the centroids; to be filled in with the fit m
        self.centroids = []

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

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

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

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

        for cls in self.classes_:
            # Select the datapoints of class 'cls'
            X_class = X[y == cls]

            # Compute the centroid as the mean of these points
            centroid = np.mean(X_class, axis=0)

            # Append to the centroids list
            self.centroids.append(centroid)
```

```
        # Convert centroids to a numpy array for easier distance computat
        self.centroids = np.array(self.centroids)

        ###  YOUR CODE ENDS HERE  ###


    def predict(self, X):
        """ Makes predictions with the nearest centroid classifier on the

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

        Returns:
        y_pred: a numpy array of predicted labels; shape (m, ), where m i
        """
        ### YOUR CODE STARTS HERE ###
        # Hint: find the distance from each x[i] to the centroids, and pr

        # Initialize an empty list to store predictions
        y_pred = []

        # For each point in X, find the closest centroid
        for x in X:
            # Compute the Euclidean distance between x and all centroids
            distances = np.linalg.norm(self.centroids - x, axis=1)

            # Find the index of the nearest centroid
            nearest_centroid_idx = np.argmin(distances)

            # Append the corresponding class to the predictions
            y_pred.append(self.classes_[nearest_centroid_idx])

        ###  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 [100…
```
nc_classifier = NearestCentroidClassifier()  # Create a NearestCentroidCl
nc_classifier.fit(nych_X, nych_y)            # Fit to the NYC training da

print(nc_classifier.centroids)
```

```
[[  16.1489863    25.07251963 1926.94      ]
 [  14.60837771   21.4446885  1935.29      ]
 [  11.59781341   19.20414033 1976.83      ]]
```

In [116…
```
# Print the predicted and true labels for the first ten data points in th
### YOUR CODE STARTS HERE ###

predicted_labels = nc_classifier.predict(nych_X[0:10])

# Print the predicted and true labels for the first ten data points
for i in range(10):
```

```
    print(f"Data Point {i + 1}: Predicted Label = {predicted_labels[i]},



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

```
Data Point 1: Predicted Label = 0.0, True Label = 1.0
Data Point 2: Predicted Label = 2.0, True Label = 2.0
Data Point 3: Predicted Label = 0.0, True Label = 0.0
Data Point 4: Predicted Label = 2.0, True Label = 2.0
Data Point 5: Predicted Label = 2.0, True Label = 1.0
Data Point 6: Predicted Label = 2.0, True Label = 2.0
Data Point 7: Predicted Label = 0.0, True Label = 0.0
Data Point 8: Predicted Label = 0.0, True Label = 0.0
Data Point 9: Predicted Label = 2.0, True Label = 1.0
Data Point 10: Predicted Label = 1.0, True Label = 1.0
```

Problem 2.2 (15 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 [118…

```
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 datap

    Returns:
    error rate: the error rate of y_pred compared to y; scalar expressed
```

```
    """
    ### YOUR CODE STARTS HERE ###
    # Calculate the number of incorrect predictions
    incorrect_predictions = np.sum(y != y_pred)

    # Calculate the total number of predictions
    total_predictions = y.shape[0]

    # Compute the error rate
    error_rate = incorrect_predictions / total_predictions

    ###  YOUR CODE ENDS HERE  ###

    return error_rate
```

In [204…

```
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 datap

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

    ### YOUR CODE STARTS HERE ###
    classes = np.unique(y)
    num_classes = len(classes)

    # Initialize the confusion matrix with zeros
    confusion_matrix = np.zeros((num_classes, num_classes), dtype=int)

    # Compute the confusion matrix
    for true_label in range(num_classes):
        for pred_label in range(num_classes):
            confusion_num = 0
            for i in range(len(y)):
                if y[i] == true_label and y_pred[i] == pred_label:
                    confusion_num += 1
            confusion_matrix[true_label, pred_label] = confusion_num


    ###  YOUR CODE ENDS HERE  ###

    return confusion_matrix
```

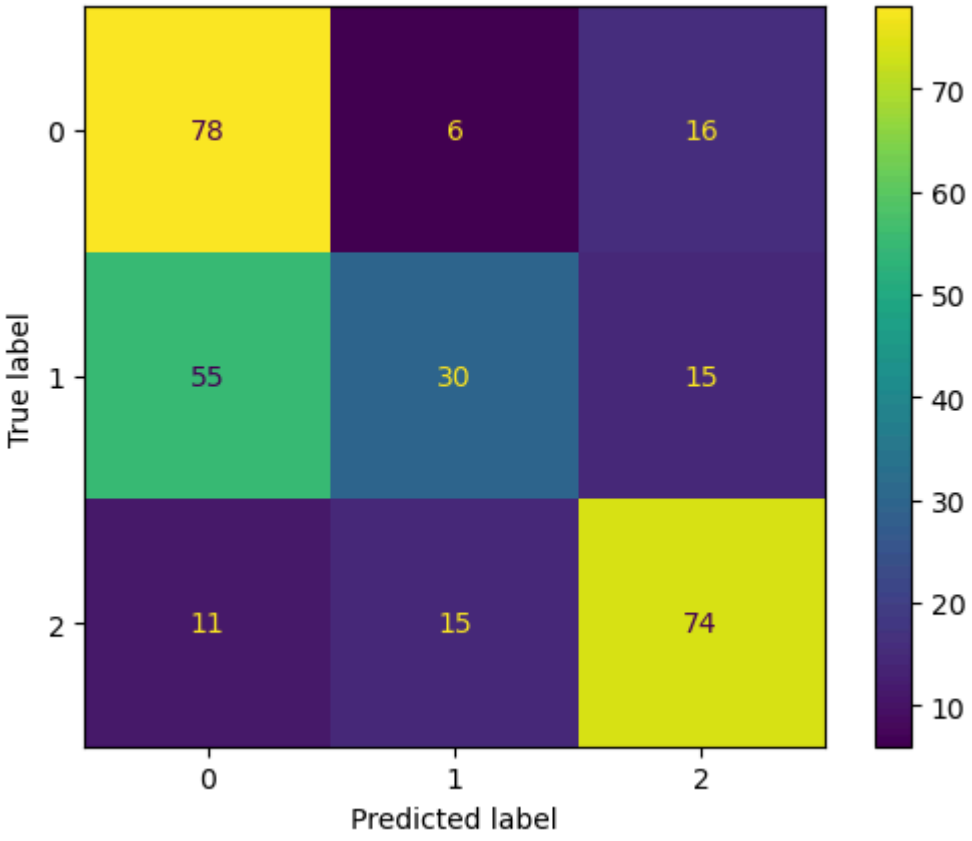
You can run the two code cells below to compare your answers to the implementations in `sklearn`. If your answers are correct, the outputs of these two functions should be the same. Do not modify the functions `eval_sklearn_implementation` and `eval_my_implementation`, but make sure that you read and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

In [207…

```
################################################
### 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 accuracies using the sklearn function accuracy_score
    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_matr
    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): 0.3933333333333333
```

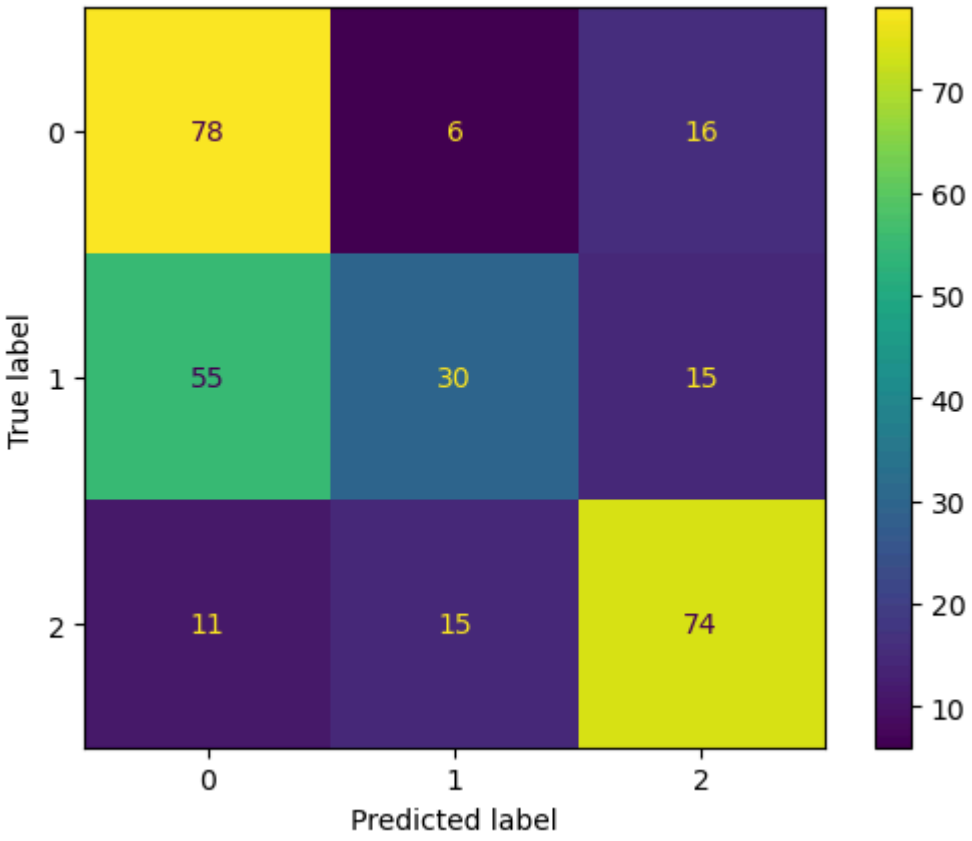


In [209…

```
##########################################
### 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 accuracies using your function compute_accuracy
    err = zero_one_loss(y, y_pred)

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

    # Evaluate confusion matrix using your function compute_confusion_mat
    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

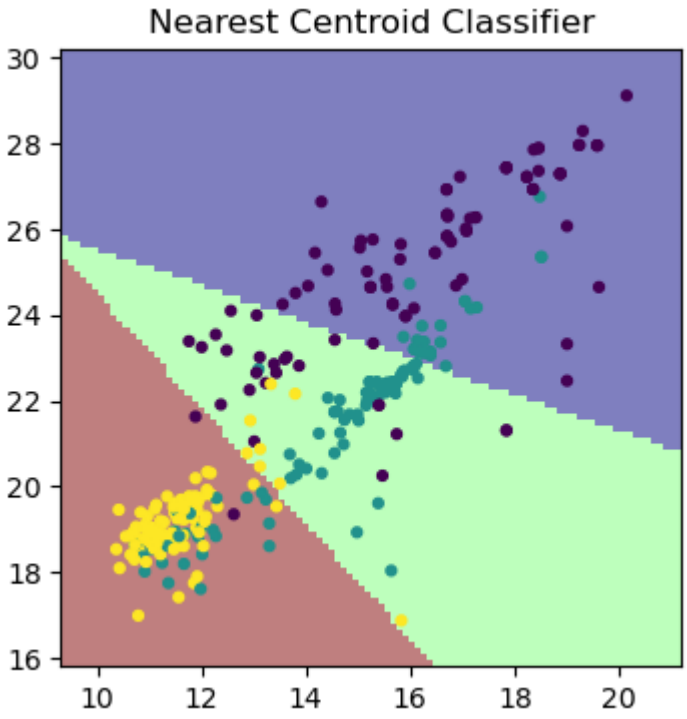
For the final problem of this homework, you will visualize the decision function and decision boundary of your nearest centroid classifier on 2D data, and compare it to the similar but more flexible Gaussian Bayes classifier discussed in class. Code for drawing the decision function (which simply evaluates 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 [215…

```
# 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_
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:

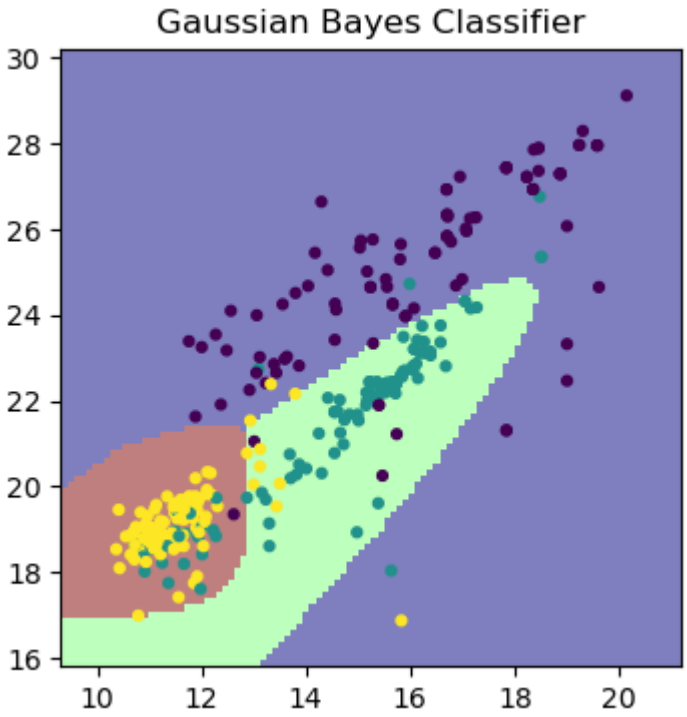
In [218…
```
class GaussianBayesClassifier:
    def __init__(self):
        """Initialize the Gaussian Bayes Classifier"""
        self.pY   = []         # class prior probabilities, p(Y=c)
        self.pXgY = []         # class-conditional probabilities, p(X|Y=c
        self.classes_ = []     # list of possible class values

    def fit(self, X, y):
        """ Fits a Gaussian Bayes classifier with training features X and
            X, y : (m,n) and (m,) arrays of training features and target
        """
        from sklearn.mixture import GaussianMixture
        self.classes_ = np.unique(y)         # Identify the class labels;
        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
            self.pXgY.append(model_c)        #

    def predict(self, X):
        """ Makes predictions with the nearest centroid classifier on the
            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.pXg
        pXY *= np.array(self.pY).reshape(1,-1)         # evaluate p(X=x|Y
        pYgX = pXY/pXY.sum(1,keepdims=True)            # normalize to p(Y
        return self.classes_[np.argmax(pYgX, axis=1)]  # find the max ind
```

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 [222…
```
# 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

###  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_
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
```



Problem 3.3 (5 points): Analysis

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

1) The Gaussian Bayesian classifier significantly reduces the error compared to the centroid classifier. This is because the centroid classifier assumes that the class distribution is relatively uniform and the shapes are similar. The adaptability to non-uniform class distribution is poor, and their decision boundaries are usually straight lines or planes. The Gaussian Bayesian classifier uses Gaussian distribution to obtain the data distribution of each class, which can effectively utilize the differences between features, allowing it to handle datasets with more complex class distribution.

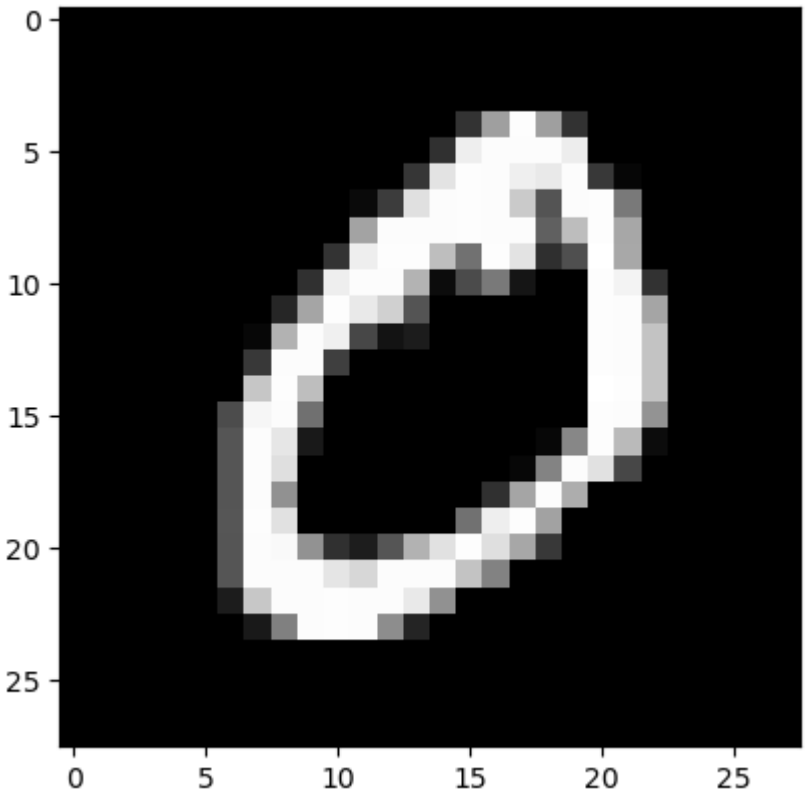
Problem 4: MNIST Data

Next, let us apply our learners to a higher-dimensional data set, the MNIST dataset. The MNIST dataset is an image dataset consisting of 70,000 hand-written digits (from 0 to 9), each of which is a 28x28 grayscale image. For each image, we also have a label, corresponding to which digit is written. Run the following code cell to load the MNIST dataset:

In [229…
```
# Load the features and labels for the MNIST dataset
# This might take a minute to download the images.
mnist_X, mnist_y = fetch_openml('mnist_784', as_frame=False, return_X_y=T

# Convert labels to integer data type
mnist_y = mnist_y.astype(int)
```

Each data point in the MNIST dataset is 768-dimensional, with each feature corresponding to a pixel intensity of a 28×28 scan of a digit. To visualize a data point, we can re-shape the feature vector into the shape of the image, and then display it using `imshow`:

In [231…
```
plt.imshow( mnist_X[1,:].reshape(28,28) ,cmap='gray');
```



Problem 4.1 (5 points): Training on MNIST

First, let us train a nearest centroid classifier on the MNIST data. For this problem, we will go ahead and use the scikit-learn implementation, just so that it's not dependent on your earlier problem solution.

In [240…
```
mnist_nearest_centroid = NearestCentroid()


### YOUR CODE STARTS HERE ###
mnist_nearest_centroid.fit(mnist_X, mnist_y)
# fit mnist_nearest_centroid to your mnist data
# dir(mnist_nearest_centroid)
###  YOUR CODE ENDS HERE  ###
```

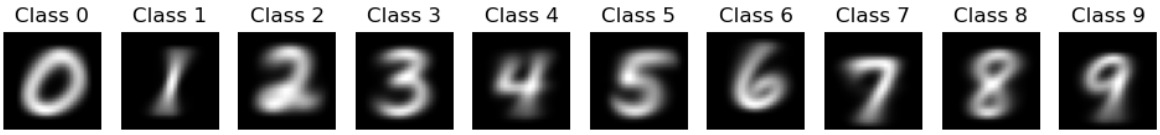
Out[240…
```
NearestCentroid()
```

Problem 4.2 (5 points): Visualizing the centroids

If you look at the trained model with, say, `dir(mnist_nearest_centroid)`, you will see that the centroids are stored in `mnist_nearest_centroid.centroids_`.

Each centroid is a vector in the same 28 x 28 vector space as the original images. So, we can visualize the centroid in the same way that we visualized a data point. Run through all ten centroids and draw them (suing `imshow`):

In [244…
```
# Create a figure with 1 row and 3 columns
fig, axes = plt.subplots(1, 10, figsize=(12, 3))

for i,c in enumerate(mnist_nearest_centroid.classes_):
    pass
    ### YOUR CODE STARTS HERE ###
    # Reshape the centroid vector into a 28x28 image
    centroid_image = mnist_nearest_centroid.centroids_[i].reshape(28, 28)
    # display centroid for class c using axes[i].imshow()
    axes[i].imshow(centroid_image, cmap='gray')
    axes[i].set_title(f"Class {c}")
    axes[i].axis('off')  # Hide the axes

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



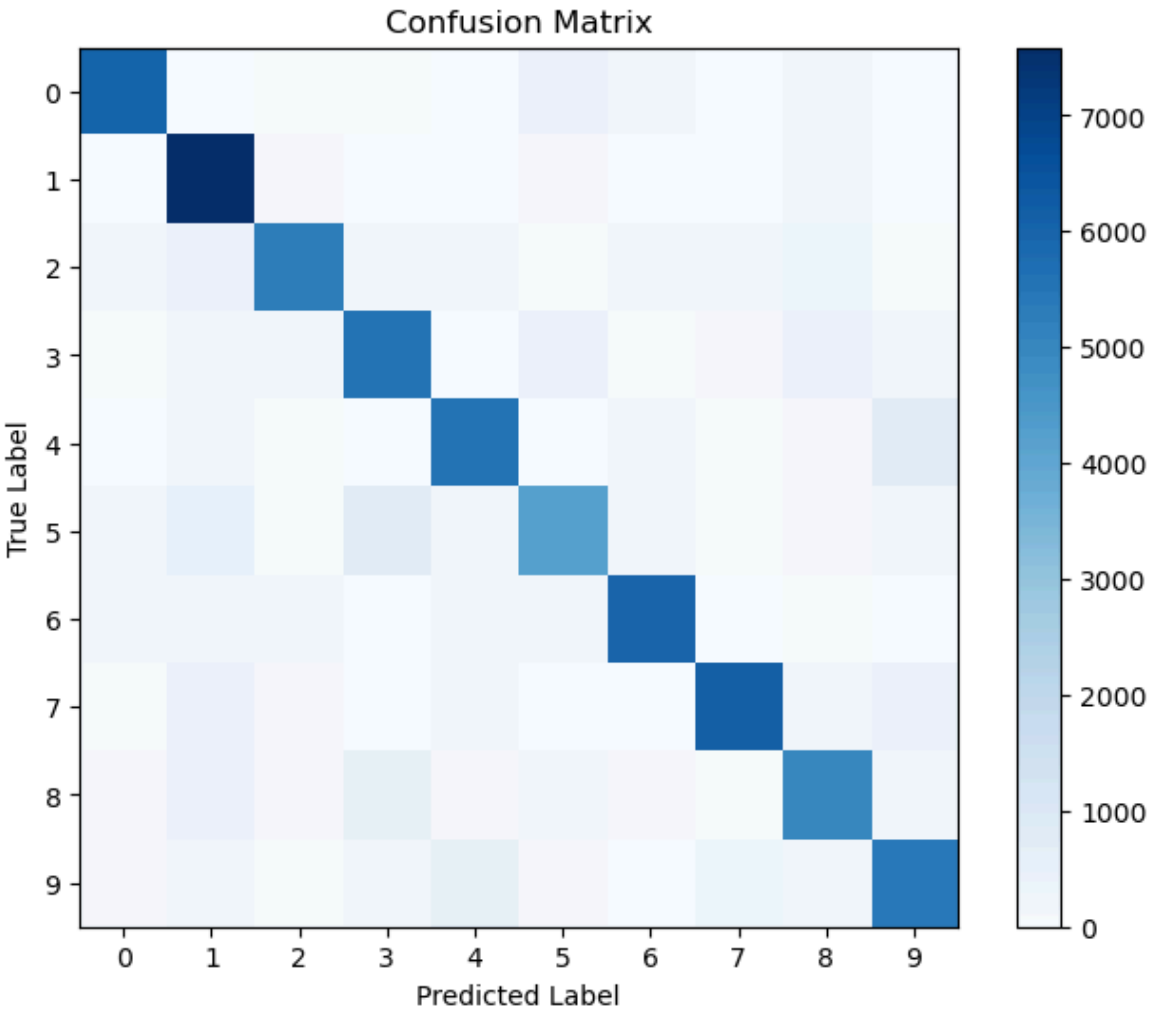
Problem 4.3 (10 points): MINST Error Rate and Confusion Matrix

Now, use `scikit`'s functions to compute the error rate of your nearest centroid classifier, and also the confusion matrix.

In [278…
```
### YOUR CODE STARTS HERE ###
# Fit the Nearest Centroid Classifier to your MNIST training data
mnist_nearest_centroid.fit(mnist_X, mnist_y)

# Predict the labels on the test set
mnist_y_pred = mnist_nearest_centroid.predict(mnist_X)
```

```python
# Compute the error rate (1 - accuracy)
error_rate = zero_one_loss(mnist_y, mnist_y_pred)
print(f"Error Rate: {error_rate:.4f}")

# Compute the confusion matrix
conf_matrix = confusion_matrix(mnist_y, mnist_y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Optionally, plot the confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.colorbar()
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.xticks(range(10), labels=range(10))  # Labels for digits 0-9
plt.yticks(range(10), labels=range(10))
plt.show()

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

```
Error Rate: 0.1905
Confusion Matrix:
[[6020    4   58   30   16  451  191   24   93   16]
 [   0 7587   67   14    3   82   12    7   99    6]
 [ 120  428 5309  231  206   39  204  145  275   33]
 [  55  210  229 5540   14  402   56   84  383  168]
 [  12  163   31    0 5529   12  124   30   85  838]
 [ 101  506   32  826  168 4251  136   53   78  162]
 [  99  258  169    4  141  227 5948    0   30    0]
 [  36  375   87    6  170   21    4 6123   90  381]
 [  62  402   88  621   78  241   61   35 4975  262]
 [  80  204   50  106  608   63    9  320  137 5381]]
```



What are some of the most common mistakes? What are some uncommon mistakes? Thinking about the data and problem, do these make sense?

1) Common errors include confusion between numbers 3 and 8, and confusion between numbers 9 and 4. An uncommon mistake is to mistake the number 1 for another number. This makes sense because common errors usually occur between numbers with similar shapes, while uncommon errors occur between numbers with significant differences in shape. The centroid classifier often performs worse than other more complex classifiers when dealing with classification tasks with complex boundaries

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.

1) This assignment is completed independently.