cs273-hw1-Abdul_Kalam_Syed

October 10, 2025

1 CS273A Homework 1

1.0.1 Due: Monday, October 6th 2025 (11:59 PM)

1.1 Instructions

Welcome to CS 273A!

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.

1.1.1 Summary of Assignment: 100 total points

- Problem 1: Exploring a NYC Housing Dataset (25 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 (5 points)
- Problem 2: Building a Nearest Centroid Classifier (35 points)
  - Problem 2.1: Implementing Nearest Centroids (20 points)
  - Problem 2.2: Evaluating Nearest Centroids (15 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)
- Problem 4: MNIST data (20 points)
  - Problem 4.1: Training the model (5 points)
  - Problem 4.2: Visualizing the centroids (5 points)
  - Problem 4.3: Error rate and confusion matrix (10 points)
- Statement of Collaboration (5 points)

<img src="data:image/svg+xml,%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22UTF-8%22%20standa

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.

[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\syedz\anaconda3\lib\site-packages (1.7.2)
Requirement already satisfied: numpy>=1.22.0 in
c:\users\syedz\anaconda3\lib\site-packages (from scikit-learn) (1.24.3)
Requirement already satisfied: scipy>=1.8.0 in
c:\users\syedz\anaconda3\lib\site-packages (from scikit-learn) (1.11.1)
Requirement already satisfied: joblib>=1.2.0 in
c:\users\syedz\anaconda3\lib\site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in
c:\users\syedz\anaconda3\lib\site-packages (from scikit-learn) (3.6.0)
```

[2]:
```
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,␣
↪ConfusionMatrixDisplay
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)
```

1.2 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:

[3]:
```
# 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.

1.2.1 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).

[4]:
```
# The shape of nych_X is
print(f"nych_X shape: {nych_X.shape}")
# The shape of nych_y is
print(f"nych_y shape: {nych_y.shape}\n")

# How many data points in our dataset?
print(f"Number of data points: {nych_X.shape[0]}")
# How many features in our dataset?
```

```
print(f"Number of features: {nych_X.shape[1]}\n")

# How many different classes (labels) are there?
print(f"Number of unique labels: {len(np.unique(nych_y))}\n") # This is already␣
↪mentioned in the dataset description, three labels, the target "y"

# Print rows 3, 4, 5, and 6 of the features and correspoinding labels
print(f"Printing row 3, 4, 5, and 6:\n{nych_X[3:7,:]}")
print(f"Corresponding labels:\n{nych_y[3:7]}\n")
```

```
nych_X shape: (300, 3)
nych_y shape: (300,)

Number of data points: 300
Number of features: 3

Number of unique labels: 3

Printing row 3, 4, 5, and 6:
[[  11.839204   19.416995 1980.      ]
 [  18.517396   25.357833 1973.      ]
 [  11.050529   19.041723 2014.      ]
 [  17.255803   26.280297 1917.      ]]
Corresponding labels:
[2. 1. 2. 0.]
```

1.2.2 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.

[5]:
```
# # Mean of each feature
# print(f"Mean of size: {np.mean(nych_X[:,0])}")
# print(f"Mean of price: {np.mean(nych_X[:,1])}")
# print(f"Mean of year built: {np.mean(nych_X[:,2])}\n")

# # Variance of each feature
# print(f"Variance of size: {np.var(nych_X[:,0])}")
# print(f"Variance of price: {np.var(nych_X[:,1])}")
# print(f"Variance of year built: {np.var(nych_X[:,2])}\n")

# # Standard deviation of each feature
```

```python
# print(f"Standard deviation of size: {np.std(nych_X[:,0])}")
# print(f"Standard deviation of price: {np.std(nych_X[:,1])}")
# print(f"Standard deviation of year built: {np.std(nych_X[:,2])}\n")

# # Minimum value of each feature
# print(f"Minimum value of size: {np.min(nych_X[:,0])}")
# print(f"Minimum value of price: {np.min(nych_X[:,1])}")
# print(f"Minimum value of year built: {np.min(nych_X[:,2])}\n")

# # Maximum value of each feature
# print(f"Maximum value of size: {np.max(nych_X[:,0])}")
# print(f"Maximum value of price: {np.max(nych_X[:,1])}")
# print(f"Maximum value of year built: {np.max(nych_X[:,2])}\n")

# Making this more efficient by using a loop
feature_names = ['size', 'price', 'year built']
for i in range(nych_X.shape[1]):
    print(f"Feature: {feature_names[i]}")
    print(f"  Mean: {np.mean(nych_X[:,i])}")
    print(f"  Variance: {np.var(nych_X[:,i])}")
    print(f"  Standard Deviation: {np.std(nych_X[:,i])}")
    print(f"  Minimum: {np.min(nych_X[:,i])}")
    print(f"  Maximum: {np.max(nych_X[:,i])}\n")
```

```
Feature: size
  Mean: 14.118392473333333
  Variance: 6.60022491794569
  Standard Deviation: 2.5690902899559
  Minimum: 10.366322
  Maximum: 20.152714

Feature: price
  Mean: 21.907116153333334
  Variance: 8.871930118164771
  Standard Deviation: 2.9785785398684337
  Minimum: 16.872675
  Maximum: 29.123861

Feature: year built
  Mean: 1946.3533333333332
  Variance: 1253.0818222222222
  Standard Deviation: 35.39889577687731
  Minimum: 1893.0
  Maximum: 2014.0
```

1.2.3 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.

[6]:
```
# # Datapoint correspoinding to y = 0 (Manhattan)
# manh = nych_X[nych_y==0,:]
# #print(manh)
# # Mean of each feature for Manhattan
# print(f"Mean of size for y=0 (Manhattan): {np.mean(manh[:,0])}")
# print(f"Mean of price for y=1 (Manhattan): {np.mean(manh[:,1])}")
# print(f"Mean of year built for y=2 (Manhattan): {np.mean(manh[:,2])}\n")

# # Standard deviation of each feature for Manhattan
# print(f"Standard deviation of size for y=0 (Manhattan): {np.std(manh[:,0])}")
# print(f"Standard deviation of price for y=1 (Manhattan): {np.std(manh[:,1])}")
# print(f"Standard deviation of year built for y=2 (Manhattan): {np.std(manh[:
 ↪,2])}\n")



# # Datapoint correspoinding to y = 1 (Bronx)
# bronx = nych_X[nych_y==1,:]
# #print(bronx)
# # Mean of each feature for Bronx
# print(f"Mean of size for y=1 (Bronx): {np.mean(bronx[:,0])}")
# print(f"Mean of price for y=1 (Bronx): {np.mean(bronx[:,1])}")
# print(f"Mean of year built for y=1 (Bronx): {np.mean(bronx[:,2])}\n")

# # Standard deviation of each feature for Bronx
# print(f"Standard deviation of size for y=1 (Bronx): {np.std(bronx[:,0])}")
# print(f"Standard deviation of price for y=1 (Bronx): {np.std(bronx[:,1])}")
# print(f"Standard deviation of year built for y=1 (Bronx): {np.std(bronx[:
 ↪,2])}\n")

# # Datapoint correspoinding to y = 2 (Staten Island)
# staten = nych_X[nych_y==2,:]
# #print(staten)
# # Mean of each feature for Staten Island
# print(f"Mean of size for y=2 (Staten Island): {np.mean(staten[:,0])}")
# print(f"Mean of price for y=2 (Staten Island): {np.mean(staten[:,1])}")
# print(f"Mean of year built for y=2 (Staten Island): {np.mean(staten[:,2])}\n")

# # Standard deviation of each feature for Staten Island
```

```python
# print(f"Standard deviation of size for y=2 (Staten Island): {np.std(staten[:
 ↪,0])}")
# print(f"Standard deviation of price for y=2 (Staten Island): {np.std(staten[:
 ↪,1])}")
# print(f"Standard deviation of year built for y=2 (Staten Island): {np.
 ↪std(staten[:,2])}\n")

# Making this more efficient by using a loop
target_labels = ['Manhattan', 'Bronx', 'Staten Island']
feature_names = ['size', 'price', 'year built']
for j in range(3):
    subset = nych_X[nych_y==j,:]
    print(f"Statistics for y={j} ({target_labels[j]}):")
    for i in range(nych_X.shape[1]):
        #print(f"  Feature: {feature_names[i]}")
        print(f"  Mean of {feature_names[i]}: {np.mean(subset[:,i])}")
        print(f"  Standard Deviation of {feature_names[i]}: {np.std(subset[:
 ↪,i])}")
    print()
```

```
Statistics for y=0 (Manhattan):
  Mean of size: 16.148986300000004
  Standard Deviation of size: 2.1941605135432343
  Mean of price: 25.072519630000002
  Standard Deviation of price: 2.0981235310287794
  Mean of year built: 1926.94
  Standard Deviation of year built: 28.145628434980804

Statistics for y=1 (Bronx):
  Mean of size: 14.60837771
  Standard Deviation of size: 1.8967844611136042
  Mean of price: 21.444688499999998
  Standard Deviation of price: 1.9906302553041209
  Mean of year built: 1935.29
  Standard Deviation of year built: 22.966190367581646

Statistics for y=2 (Staten Island):
  Mean of size: 11.597813410000004
  Standard Deviation of size: 0.8196508202645331
  Mean of price: 19.20414033
  Standard Deviation of price: 0.8434230157638462
  Mean of year built: 1976.83
  Standard Deviation of year built: 31.80441950421356
```

1.2.4 Problem 1.4 (5 points): Feature Histograms

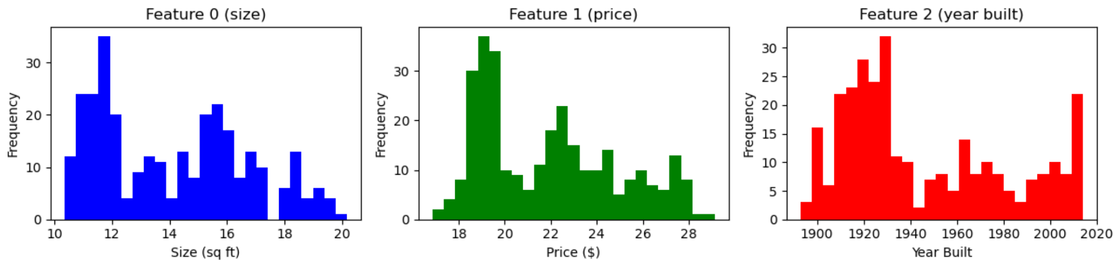
Now, you will visualize the distribution of each feature with histograms. Use `matplotlib.pyplot` to do this, which is already imported for you as `plt`. Do not use any other plotting libraries, such as `pandas` or `seaborn`.

- 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.)

[7]:
```
# Create a figure with 1 row and 3 columns
fig, axes = plt.subplots(1, 3, figsize=(12, 3))

### YOUR CODE STARTS HERE ###
# # Histogram of each feature
# axes[0].hist(nych_X[:,0], bins=25, color='blue')
# axes[0].set_title('Feature 0 (Size)')
# axes[0].set_xlabel('Size (sq ft)')
# axes[0].set_ylabel('Frequency')

# axes[1].hist(nych_X[:,1], bins=25, color='green')
# axes[1].set_title('Feature 1 (Price)')
# axes[1].set_xlabel('Price ($)')
# axes[1].set_ylabel('Frequency')

# axes[2].hist(nych_X[:,2], bins=25, color='red')
# axes[2].set_title('Feature 2 (Year Built)')
# axes[2].set_xlabel('Year Built')
# axes[2].set_ylabel('Frequency')

# Making this more efficient by using a loop
colors = ['blue', 'green', 'red']
for i in range(3):
    axes[i].hist(nych_X[:,i], bins=25, color=colors[i])
    axes[i].set_title(f'Feature {i} ({feature_names[i]})')
    if i == 0:
        axes[i].set_xlabel('Size (sq ft)')
    elif i == 1:
        axes[i].set_xlabel('Price ($)')
    elif i == 2:
        axes[i].set_xlabel('Year Built')
    else:
        axes[i].set_xlabel('Year Built')
    axes[i].set_ylabel('Frequency')
```

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

fig.tight_layout()
```



1.2.5 Problem 1.5 (5 points): Feature Scatter Plots

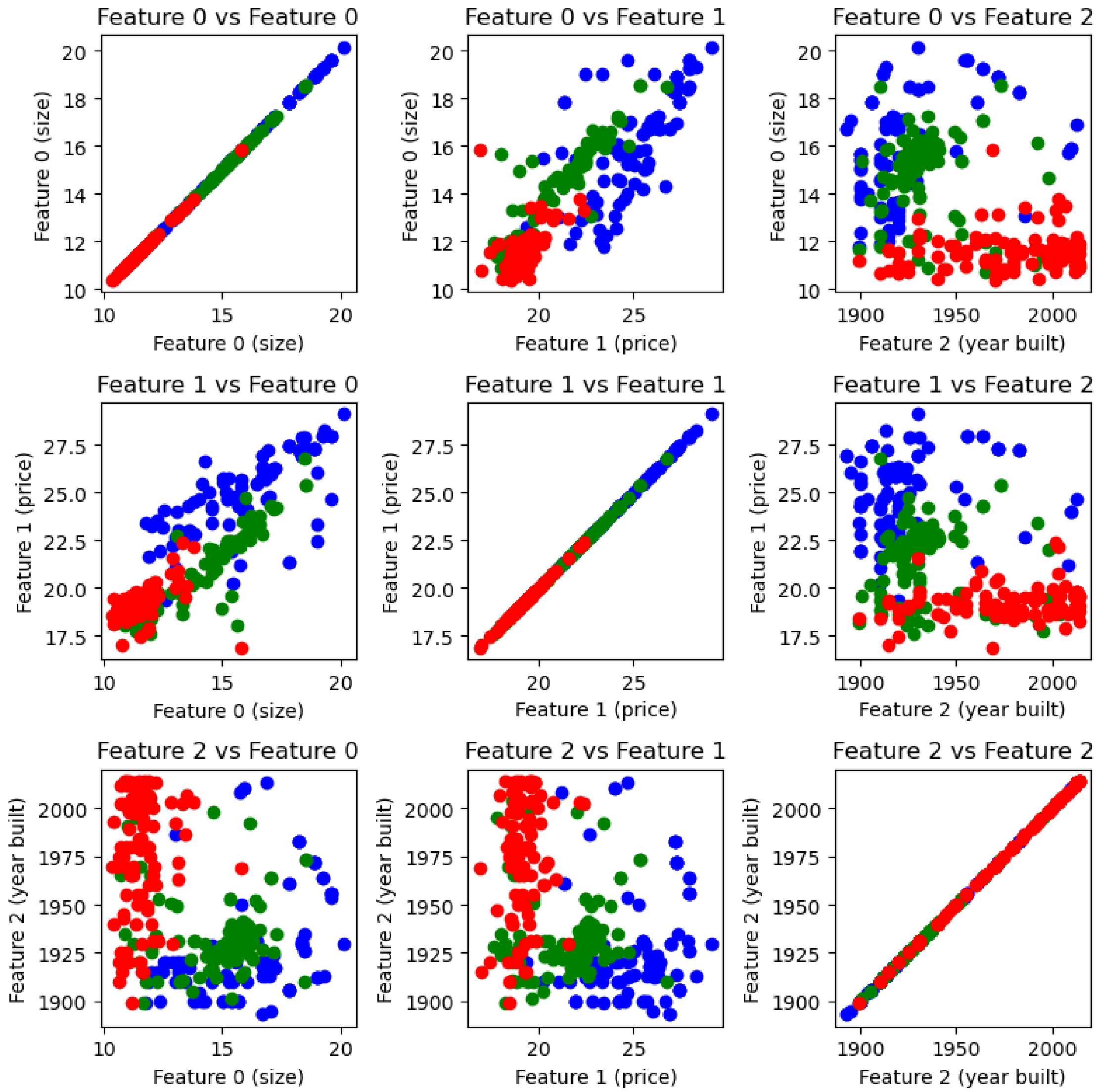
To help further visualize the NYC-Housing datset, you will now create several scatter plots of the features. Use `matplotlib.pyplot` to do this, which is already imported for you as `plt`. Do not use any other plotting libraries, such as `pandas` or `seaborn`.

- 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.

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

### YOUR CODE STARTS HERE ###
# Scatter plot of each pair of features, colored by label
colors = ['blue', 'green', 'red']
for i in range(3):
    for j in range(3):
        for k in range(3):
            subset = nych_X[nych_y==k,:]
            axes[i,j].scatter(subset[:,j], subset[:,i], color=colors[k])
        axes[i, j].set_xlabel(f'Feature {j} ({feature_names[j]})')
        axes[i, j].set_ylabel(f'Feature {i} ({feature_names[i]})')
        axes[i, j].set_title(f'Feature {i} vs Feature {j}')

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

```
fig.tight_layout()
```



<img src="data:image/svg+xml,%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22UTF-8%22%20standa

---

1.3 Problem 2: Nearest Centroid Classifiers

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

1.3.1 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.

[9]:
```
class NearestCentroidClassifier:
    def __init__(self):
        # A list containing the centroids; to be filled in with the fit method.
        self.centroids = []
        # Creating a array containing the unique class labels, so it can be␣
↪used in predict method
        self.classes_ = None

    def fit(self, X, y):
        """ Fits the nearest centroid classifier with training features X and␣
↪training labels y.

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

        """
        # 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␣
↪centroid for each class.
        # The centroid (mean vector) can be computed in a similar way to P2.2,␣
↪for example.
```

```python
        for c in self.classes_:
            class_data = X[y == c] # adding the features corresponding to class␣
↪c (y)
            centroid = np.mean(class_data, axis=0) # calculating the mean of␣
↪the features for class c
            self.centroids.append(centroid) # appending the centroid to the␣
↪list of centroids

        # print(f"Centroids:\n{self.centroids}\n")
        self.centroids = np.array(self.centroids) # converting the list of␣
↪centroids to a numpy array for later computations
        # print(f"Centroids as numpy array:\n{self.centroids}\n")

        ###  YOUR CODE ENDS HERE  ###


    def predict(self, X):
        """ Makes predictions with the nearest centroid classifier on the␣
↪features in X.

        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␣
↪number of datapoints.
        """
        ### YOUR CODE STARTS HERE ###
        # Hint: find the distance from each x[i] to the centroids, and predict␣
↪the closest.
        y_pred = []
        for i in X:
            pred_data = np.mean((i - self.centroids)**2, axis=1) # calculating␣
↪the mean squared distance from the point to each centroid
            pred_class = self.classes_[np.argmin(pred_data)] # finding the␣
↪class corresponding to the closest centroid by geting the index of the␣
↪minimum distance
            y_pred.append(pred_class) # appending the predicted class to the␣
↪list

        y_pred = np.array(y_pred) # converting the list of predicted classes to␣
↪a numpy array

        ###  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).

[10]:
```
nc_classifier = NearestCentroidClassifier()  # Create a␣
 ↪NearestCentroidClassifier object
nc_classifier.fit(nych_X, nych_y)            # Fit to the NYC training data

print(nc_classifier.centroids)
```

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

[11]:
```
# Print the predicted and true labels for the first ten data points in the NYCH␣
 ↪testing set
### YOUR CODE STARTS HERE ###

print("First 10 Predicted labels: ", nc_classifier.predict(nych_X[:10]))
print("First 10 True labels:      ", nych_y[:10])

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

```
First 10 Predicted labels:  [0. 2. 0. 2. 2. 2. 0. 0. 2. 1.]
First 10 True labels:       [1. 2. 0. 2. 1. 2. 0. 0. 1. 1.]
```

1.3.2 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 yand 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.

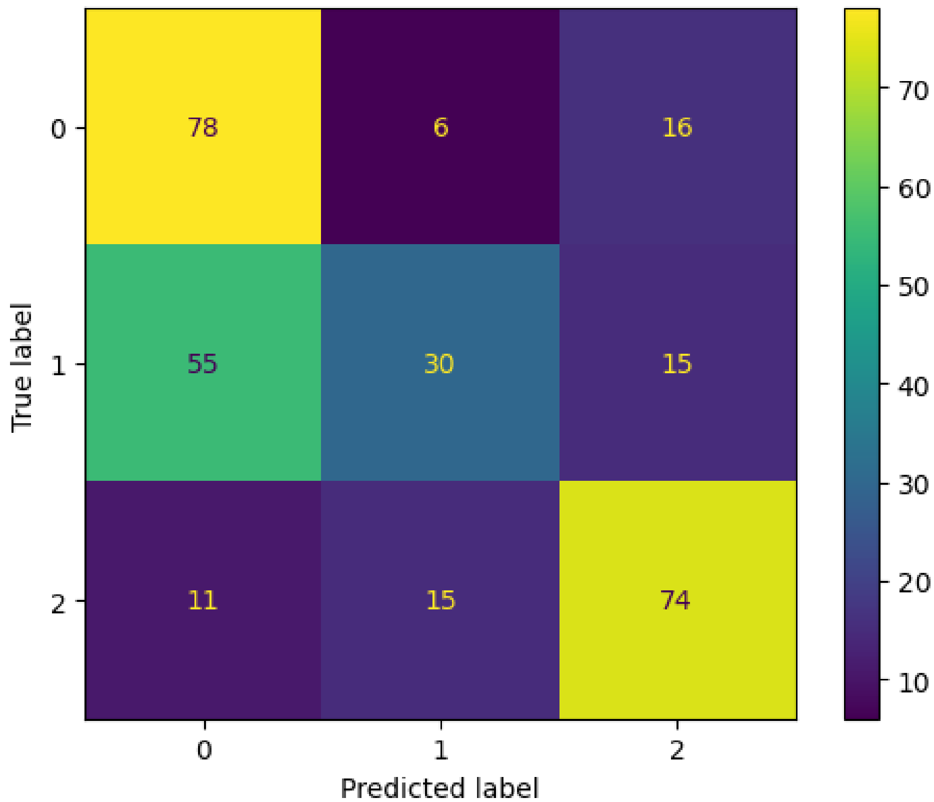
[12]:
```python
def compute_error_rate(y, y_pred):
    """ Computes the error rate of an array of predictions.

    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␣
 ↪decimal (e.g. 0.5)
    """
    ### YOUR CODE STARTS HERE ###
    # print(np.mean(y != y_pred))
    return np.mean(y != y_pred)  # this would calculate the error rate by␣
 ↪finding the mean of the array where y and y_pred are not equal

    ###  YOUR CODE ENDS HERE  ###

    #return error_rate
```

[13]:
```python
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 and y_pred; shape (C, C),
        where C is the number of unique classes.  The (i,j)th entry is the␣
 ↪number of examples of class i
        that are classified as being from class j.
    """

    ### YOUR CODE STARTS HERE ###
    # confusion matrix without using sklearn
    C = (np.unique(y)).size # number of unique classes
    confusion_matrix = np.zeros((C, C), dtype=int) # initializing the confusion␣
 ↪matrix with zeros, setting the data type to int because the entries will be␣
 ↪counts
    for i, j in zip(y, y_pred):
        confusion_matrix[int(i), int(j)] += 1 # adding 1 to the (i,j)th entry␣
 ↪for true label i and predicted label j

    ###  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.

[14]:
```
################################################
### 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_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)
```

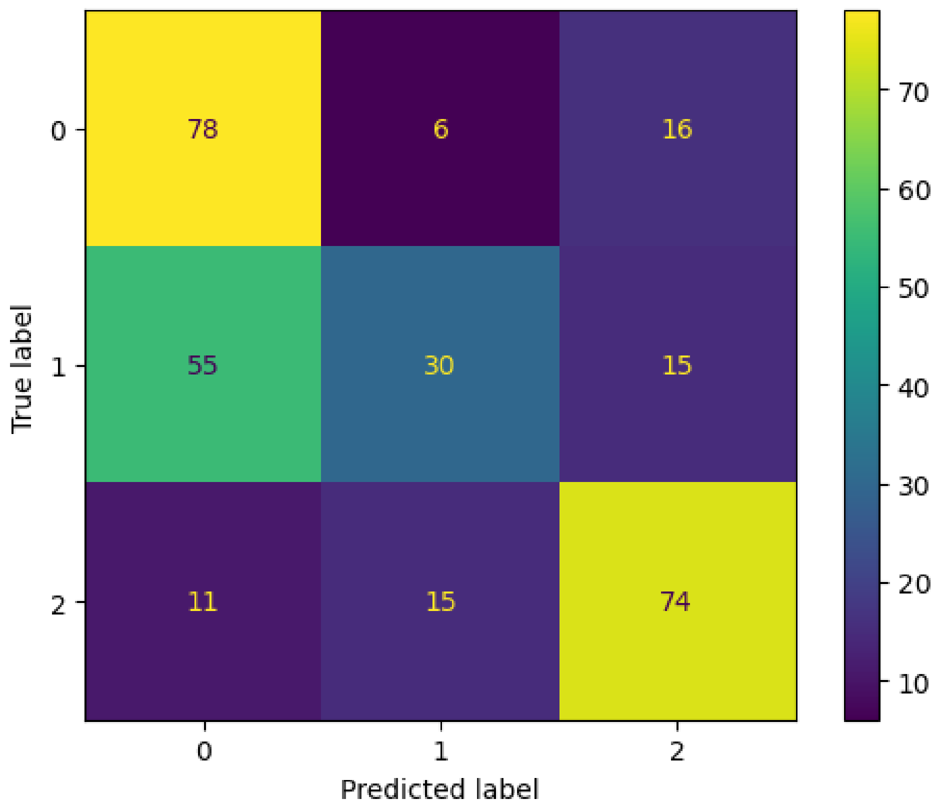
```
Sklearn Results:
--- Error Rate (0/1): 0.39333333333333333
```



[15]:
```
#########################################
### 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)
    # The line above can also be replaced with the compute_error_rate function␣
↪that I wrote earlier
    # 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): 0.39333333333333333
```



<img src="data:image/svg+xml,%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22UTF-8%22%20standa

---

1.4 Problem 3: Decision Boundaries

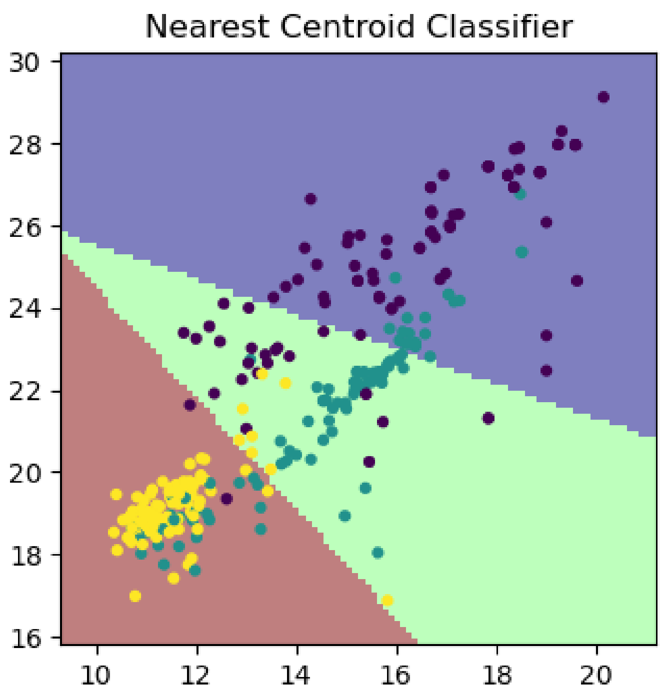
For the final problem of this homework, you will visualize the decision function and decision boundary of your nearest centroid classifier on 2D data, and compare it to the similar but more flexible

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.

1.4.1 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.

[16]:
```
# 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');
```



1.4.2 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:

[17]:
```
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␣
↪training labels y.
            X, y : (m,n) and (m,) arrays of training features and target class␣
↪values
        """
        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␣
↪features in X.
            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(Y=c)
        pYgX = pXY/pXY.sum(1,keepdims=True)             # normalize to␣
↪p(Y=c|X=x) (not required)
        return self.classes_[np.argmax(pYgX, axis=1)]  # find the max index &␣
↪return its class ID
```

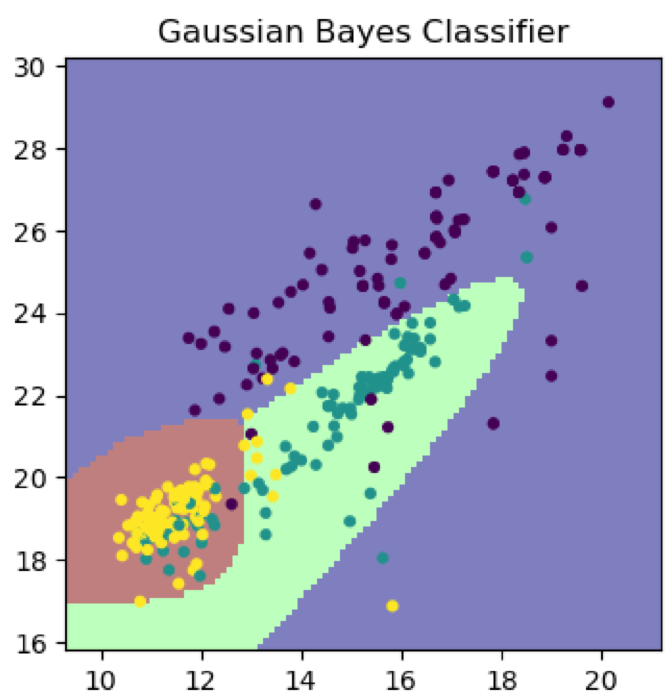
Using this learner, evaluate the predictions and error rate on the training data, and plot the decision boundary. The code should be the same as your Nearest Centroid, but using the new learner object.

[18]:
```
# 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␣
↪used in training

###  YOUR CODE ENDS HERE  ###

err = zero_one_loss(nych_y, gbc_y_pred)
```

```python
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');
```

```
c:\Users\syedz\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\syedz\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\syedz\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(
```

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



1.4.3 Problem 3.3 (5 points): Analysis

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

The error decreased when using Guassian Bayes classifier. We can visually see the decision boundaries are curved and the better aligned. On the other hand, Nearest Centroid classifer only draws straight lines, making it less likely to adpot to changes, as it is only using the mean, while Guassian Bayes classifer uses additional data, like the variance.

<img src="data:image/svg+xml,%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22UTF-8%22%20standa

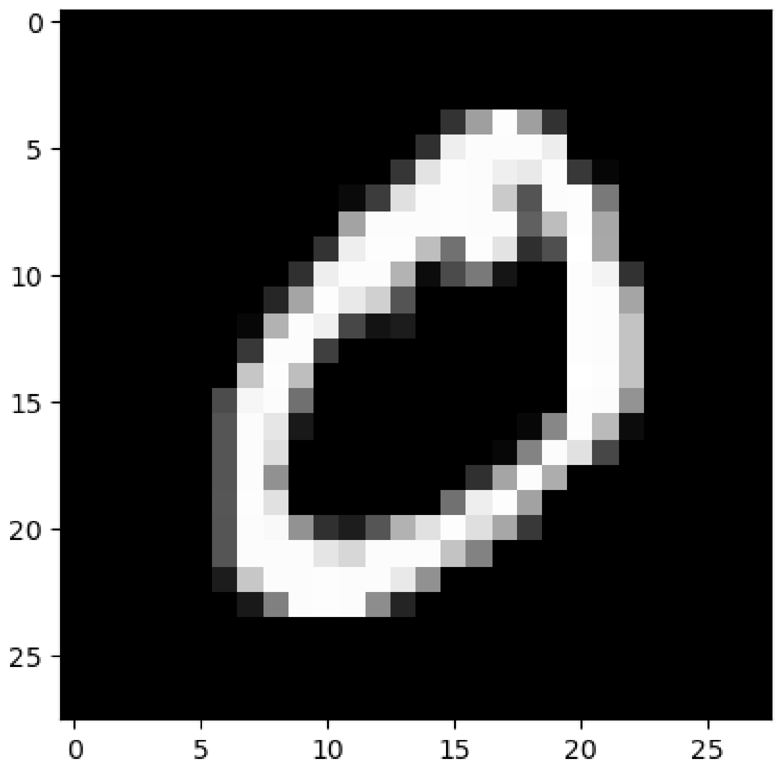
1.5 Problem 4: MNIST Data

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:

[19]:
```
# 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=True,␣
 ↪parser='auto')

# 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`:

[20]:
```
plt.imshow( mnist_X[1,:].reshape(28,28) ,cmap='gray');
```



1.5.1 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.

[21]:
```
mnist_nearest_centroid = NearestCentroid()


### YOUR CODE STARTS HERE ###

# fit mnist_nearest_centroid to your mnist data
mnist_nearest_centroid.fit(mnist_X, mnist_y)

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

```
c:\Users\syedz\anaconda3\Lib\site-
packages\sklearn\neighbors\_nearest_centroid.py:244: UserWarning:
self.within_class_std_dev_ has at least 1 zero standard deviation.Inputs within
the same classes for at least 1 feature are identical.
  warnings.warn(
```

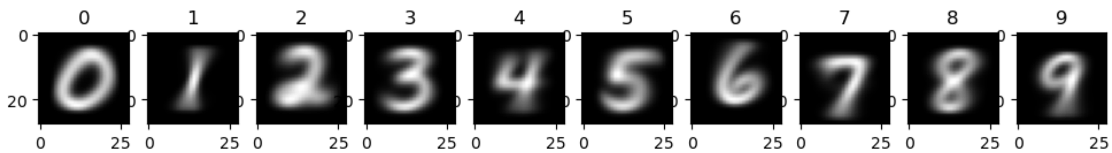
[21]:
```
NearestCentroid()
```

1.5.2 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`):

[22]:
```
# 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 ###

    # display centroid for class c using axes[i].imshow()
    axes[i].imshow(mnist_nearest_centroid.centroids_[i].reshape(28, 28),␣
↪cmap='gray')
    axes[i].set_title(f'{c}')

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



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

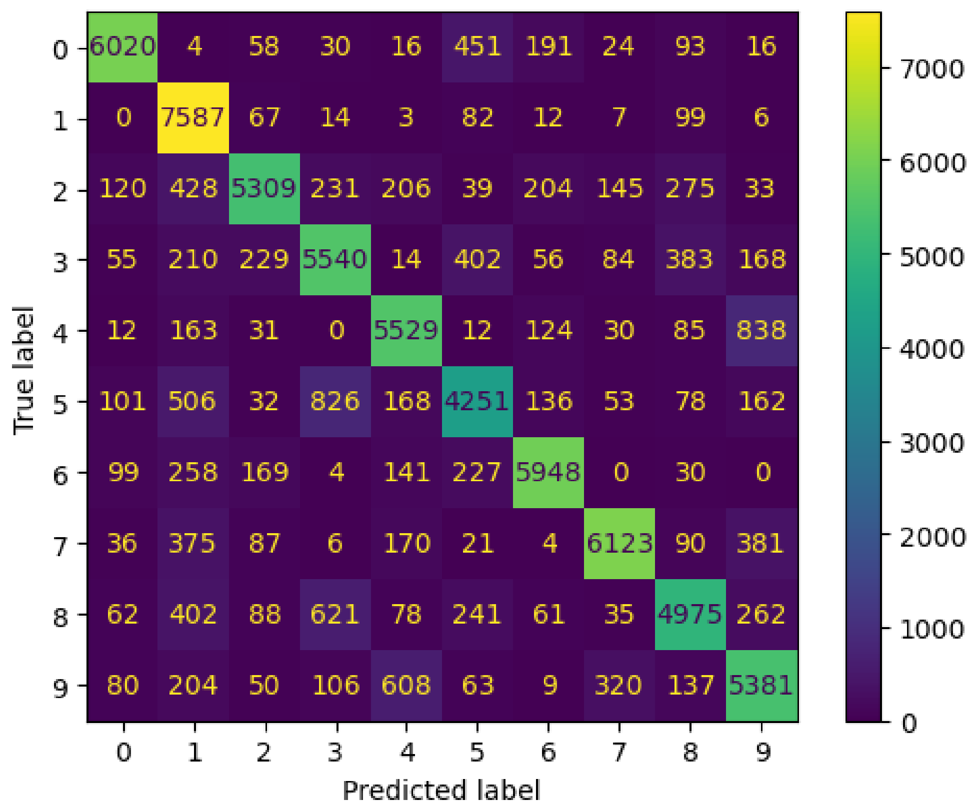
Now, use `scikit`'s functions to compute the error rate of your nearest centroid classifier, and also the confusion matrix.

[23]:
```
### YOUR CODE STARTS HERE ###

# Predict on the MNIST data
mnist_y_pred = mnist_nearest_centroid.predict(mnist_X)

# Computing error rate
err = zero_one_loss(mnist_y, mnist_y_pred)
print(f"Nearest Centroid Error Rate (0/1): {err}")

# Computing confusion matrix
cm = confusion_matrix(mnist_y, mnist_y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix = cm)
disp.plot()
```

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

Nearest Centroid Error Rate (0/1): 0.19052857142857138

[23]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x205698e9950>



Looking at the confusion matrix, what are some of the most common mistakes (true vs. predicted class)? What are some uncommon mistakes? Thinking about the data and task, do these make sense? (Are these reasonable classses to confuse?)

Some of the common mistakes include confusing 4 and 9 between each other, which is warranted, as these numbers could look similar. 5 and 8 are confused to be 3 quite a lot as well, however, not as high the other way around, but it is still warranted as they could look similar, especially if the images are blurred like we have seen in the samples above. Some uncommon mistakes include 0 and 1, not many mistakes here. 6 and 7, and more surprisingly 6 and 9, they are quite different, but the same if directions are changed. The confusion matrix does make sense overall.

<img src="data:image/svg+xml,%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22UTF-8%22%20standa

1.5.4 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.

(*Note: If you did not collaborate with anyone, you may simply state that.*)

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.

I have not collaborated with anyone on this assignment. I did read some comments of EdD about some people not being able to load the MNIST dataset, but it worked for me as intended.