Assignment 2 - Supervised Machine Learning Fundamentals

Jiechen Li

Netid: jl1254

Names of students you worked with on this assignment: Yabei Zeng, ChatGPT for concepts, formula clarification and grammar check.

Instructions for all assignments can be found here.

Note: this assignment falls under collaboration Mode 2: Individual Assignment – Collaboration Permitted. Please refer to the syllabus for additional information.

Total points in the assignment add up to 90; an additional 10 points are allocated to presentation quality.

Learning Objectives:

By successfully completing this assignment you will be able to...

- Explain the bias-variance tradeoff of supervised machine learning and the impact of model flexibility on algorithm performance
- Perform supervised machine learning training and performance evaluation
- Implement a k-nearest neighbors machine learning algorithm from scratch in a style similar to that of popular machine learning tools like `scikit-learn`
- Describe how KNN classification works, the method's reliance on distance measurements, and the impact of higher dimensionality on computational speed
- Apply regression (linear regression) and classification (KNN) supervised learning techniques to data and evaluate the performance of those methods
- Construct simple feature transformations for improving model fit in linear models
- Fit a `scikit-learn` supervised learning technique to training data and make predictions using it

In [ ]:
```
# MAC USERS TAKE NOTE:
# For clearer plots in Jupyter notebooks on macs, run the following line of code:
%config InlineBackend.figure_format = 'retina'
```

Conceptual Questions on Supervised Learning

1

[4 points] For each part below, indicate whether we would generally expect the performance of a flexible statistical learning method to be *better* or *worse* than an inflexible method. Justify your answer.

1. The sample size n is extremely large, and the number of predictors p is small.
2. The number of predictors p is extremely large, and the number of observations n is small.
3. The relationship between the predictors and response is highly non-linear.
4. The variance of the error terms, i.e. $\sigma^2 = Var(\epsilon)$, is extremely high

ANSWER

> 1. **Better**: When we have a lot of data but not too many features to consider, a flexible approach is usually a good choice. It can learn the details and patterns in the data without getting confused or sidetracked.

> 2. **Worse**: If we have lots and lots of features but not many data points, a flexible method might start paying too much attention to random quirks in the data that don't really matter. It's like trying to read too much into a short story.

> 3. **Better**: When the connection between our data (like height and weight, or study time and test scores) isn't a straight line but more of a curve, we need a flexible approach that can follow those twists and turns to make better predictions.

> 4. **Worse**: If our data is really noisy (imagine trying to have a conversation next to a construction site), a super flexible method will try to pay attention to all that noise, which isn't helpful. A less flexible method is like having earplugs; it won't hear the noise as much and can focus on the actual conversation.

2

[6 points] For each of the following, (i) explain if each scenario is a classification or regression problem AND why, (ii) indicate whether we are most interested in inference or prediction for that problem AND why, and (iii) provide the sample size n and number of predictors p indicated for each scenario.

(a) We collect a set of data on the top 500 firms in the US. For each firm we record profit, number of employees, industry and the CEO salary. We are interested in understanding which factors affect CEO salary.

(b) We are considering launching a new product and wish to know whether it will be a success or a failure. We collect data on 20 similar products that were previously launched. For each product we have recorded whether it was a success or failure, price charged for the product, marketing budget, competition price, and ten other variables.

(c) We are interested in predicting the % change in the US dollar in relation to the weekly changes in the world stock markets. Hence we collect weekly data for all of 2012. For each week we record the % change in the dollar, the % change in the US market, the % change in the British market, and the % change in the German market.

ANSWER

> **(a)**
>
> - **Regression**: This is a regression problem because the outcome variable (y) we are interested in (CEO salary) is a continuous variable.
> - **Inference**: We are interested in inference because the goal is to understand the relationship between the predictors (profit, number of employees, industry) and the outcome (CEO salary), rather than predicting the CEO salary for a particular firm.
> - **Sample Size (n) and Number of Predictors (p)**: The sample size n is 500 (the number of firms). The number of predictors p is at least 3 (profit, number of employees, industry), but could be more if "industry" encompasses multiple binary variables (one for each industry category).

> **(b)**
>
> - **Classification**: This is a classification problem because the outcome variable we want to predict is categorical (success or failure of a product launch).
> - **Prediction**: We are most interested in prediction because we want to forecast the outcome (success or failure) of a new product launch.
> - **Sample Size (n) and Number of Predictors (p)**: The sample size n is 20 (the number of products). The number of predictors p is at least 13 (price, marketing budget, competition price, and ten other variables).

(c)

- **Regression**: This is a regression problem because the outcome variable (% change in the US dollar) is a continuous variable.
- **Prediction**: We are interested in prediction because the aim is to forecast future values of the % change in the US dollar based on the % changes in the world stock markets.
- **Sample Size (n) and Number of Predictors (p)**: The sample size n would be the number of weeks in 2012, which is 52. The number of predictors p is at least 3 (% change in the US market, the % change in the British market, and the % change in the German market).

Practical Questions

3

[6 points] Classification using KNN. The table below provides a training dataset containing six observations (a.k.a. samples) ($n = 6$) each with three predictors (a.k.a. features) ($p = 3$), and one qualitative response variable (a.k.a. target).

Table 1. Training dataset with $n = 6$ observations in $p = 3$ dimensions with a categorical response, y

Obs.	x_1	x_2	x_3	y
1	0	3	0	Red
2	2	0	0	Red
3	0	1	3	Red
4	0	1	2	Blue
5	-1	0	1	Blue
6	1	1	1	Red

We want to use the above training dataset to make a prediction, $\hat{y}$, for an unlabeled test data observation where $x_1 = x_2 = x_3 = 0$ using K-nearest neighbors. You are given some code below to get you started. *Note: coding is only required for part (a), for (b)-(d) please provide your reasoning based on your answer to part (a)*.

(a) Compute the Euclidean distance between each observation and the test point, $x_1 = x_2 = x_3 = 0$. Present your answer in a table similar in style to Table 1 with observations 1-6 as the row headers.

(b) What is our prediction, $\hat{y}$, when $K = 1$ for the test point? Why?

(c) What is our prediction, $\hat{y}$, when $K = 3$ for the test point? Why?

(d) If the Bayes decision boundary (the optimal decision boundary) in this problem is highly nonlinear, then would we expect the *best* value of K to be large or small? Why?

In [ ]:
```
import numpy as np

X = np.array([[0, 3, 0], [2, 0, 0], [0, 1, 3], [0, 1, 2], [-1, 0, 1], [1, 1, 1]])
y = np.array(["r", "r", "r", "b", "b", "r"])
```

ANSWER:

(a)

Compute the Euclidean distance between each observation and the test point, $x_1 = x_2 = x_3 = 0$:

Euclidean distance formula:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2}$$

$\mathbf{p}$ and $\mathbf{q}$ are two points in Euclidean space. In this case, we're calculating the distance from the test point (0,0,0) to each observation in the training set.

In [ ]:
```
import numpy as np

# training data
X = np.array([[0, 3, 0], [2, 0, 0], [0, 1, 3], [0, 1, 2], [-1, 0, 1], [1, 1, 1]])

# test data point
test_point = np.array([0, 0, 0])

# Euclidean distances
distances = np.sqrt(np.sum((X - test_point) ** 2, axis=1))

for i, distance in enumerate(distances, start=1):
    print(f"Observation {i}: {distance}")
```

```
Observation 1: 3.0
Observation 2: 2.0
Observation 3: 3.1622776601683795
Observation 4: 2.23606797749979
Observation 5: 1.4142135623730951
Observation 6: 1.7320508075688772
```

Obs.	**Distance**	y
1	3.0	Red
2	2.0	Red
3	3.16	Red
4	2.23	Blue
5	1.41	Blue
6	1.73	Red

(b)

> Prediction $\hat{y}$ when $K = 1$: For $K = 1$, the KNN looks for the one nearest neihbor to the test point. In this case, Observation 5 is the neareat data to the test point based on the training data. The Observation 5 has the minimum distance of 1.41. So, $\hat{y}$ is Blue when $K = 1$.

(c)

> Prediction $\hat{y}$ when $K = 3$: For $K = 3$, the KNN looks for the three nearest neihbor to the test point and predict the class based on the majority class among these neighbors. In this case, Observation 2, 5, and 6 are the three neareat data to the test point based on the training data. The Observation 2 is Red with distance as 2.0, Observation 5 is Blue with distance as 1.41, and Observation 6 is Red with distance as 1.73. Then, the majority class is Red, so $\hat{y}$ is Red when $K = 3$.

(d)

> If the Bayes decision boundary is highly nonlinear, I would expect a smaller value of K to perform better. First, because a smaller value of K makes the KNN more sensitive to local structure of the data. Second, in the context of a highly nonlinear decision boundary, using a large K can lead to over-smoothing, where the model fails to capture the complexity of the boundary and potentially misclassifies observations that are near the boundary. Last, in the context of a highly nonlinear decision boundary, a moderately small K is often a good trade-off to capture the complexity of the boundary without overfitting to the noise.

4

[18 points] Build your own classification algorithm.

(a) Build a working version of a binary KNN classifier using the skeleton code below. We'll use the `sklearn` convention that a supervised learning algorithm has the methods `fit` which trains your algorithm (for KNN that means storing the data) and `predict` which identifies the K nearest neighbors and determines the most common class among those K neighbors. *Note: Most classification algorithms typically also have a method `predict_proba` which outputs the confidence score of each prediction, but we will explore that in a later assignment.*

(b) Load the datasets to be evaluated here. Each includes training features ($\mathbf{X}$), and test features ($\mathbf{y}$) for both a low dimensional dataset ($p = 2$ features/predictors) and a higher dimensional dataset ($p = 100$ features/predictors). For each of these datasets there are $n = 1000$ observations of each. They can be found in the `data` subfolder in the `assignments` folder on github. Each file is labeled similar to

`A2_X_train_low.csv`, which lets you know whether the dataset is of features, X, targets, y; training or testing; and low or high dimensions.

(c) Train your classifier on first the low dimensional dataset and then the high dimensional dataset with $k = 5$. Evaluate the classification performance on the corresponding test data for each of those trained models. Calculate the time it takes each model to make the predictions and the overall accuracy of those predictions for each corresponding set of test data - state each.

(d) Compare your implementation's accuracy and computation time to the scikit learn KNeighborsClassifier class. How do the results and speed compare to your implementation? *Hint: your results should be identical to that of the scikit-learn implementation.*

(e) Some supervised learning algorithms are more computationally intensive during training than testing. What are the drawbacks of the prediction process being slow? In what cases in practice might slow testing (inference) be more problematic than slow training?

In [ ]:
```
# # Skeleton code for part (a) to write your own kNN classifier

# class Knn:
# # k-Nearest Neighbor class object for classification training and testing
#     def __init__(self):

#     def fit(self, x, y):
#         # Save the training data to properties of this class

#     def predict(self, x, k):
#         y_hat = [] # Variable to store the estimated class label for
#         # Calculate the distance from each vector in x to the training data

#         # Return the estimated targets
#         return y_hat

# # Metric of overall classification accuracy
# #  (a more general function, sklearn.metrics.accuracy_score, is also available)
# def accuracy(y,y_hat):
#     nvalues = len(y)
#     accuracy = sum(y == y_hat) / nvalues
#     return accuracy
```

ANSWER:

In [ ]:
```
# Question (a)
# Skeleton code for part (a) to write your own kNN classifier
import numpy as np
```

```
class Knn:
    # k-Nearest Neighbor class object for classification training and testing
    def __init__(self):
        self.train_x = None
        self.train_y = None

    def fit(self, x, y):
        # Save the training data to properties of this class
        self.train_x = x
        self.train_y = y

    def predict(self, x, k):
        y_hat = []  # Variable to store the estimated class label for
        # Calculate the distance from each vector in x to the training data
        column = x.shape[0]
        for i in range(column):
            test_point = x[i, :]
            distances = np.sqrt(np.sum((x - test_point) ** 2, axis=1))
            k_indices = np.argsort(distances)[
                :k
            ]  # sort distacne and get k nearest indices
            k_nearest_labels = self.train_y[k_indices]  # get lable of k nearest
            # convert to integer if necessary
        if k_nearest_labels.dtype != "int":
            k_nearest_labels = k_nearest_labels.astype(int)
            most_common = np.argmax(np.bincount(k_nearest_labels))
            y_hat.append(most_common)
        # Return the estimated targets
        return y_hat


# Metric of overall classification accuracy
#  (a more general function, sklearn.metrics.accuracy_score, is also available)
def accuracy(y, y_hat):
    nvalues = len(y)
    accuracy = sum(y == y_hat) / nvalues
    return accuracy
```

In [ ]:

```
# Question (b)

import pandas as pd
import time
import warnings
```

```
warnings.filterwarnings("ignore")

pd.set_option("mode.copy_on_write", True)

# load the datasets
A2_X_train_high = pd.read_csv(
    "https://github.com/kylebradbury/ids705/raw/main/assignments/data/A2_X_train_high.csv"
)
A2_y_train_high = pd.read_csv(
    "https://github.com/kylebradbury/ids705/raw/main/assignments/data/A2_y_train_high.csv"
)

A2_X_train_low = pd.read_csv(
    "https://github.com/kylebradbury/ids705/raw/main/assignments/data/A2_X_train_low.csv"
)
A2_y_train_low = pd.read_csv(
    "https://github.com/kylebradbury/ids705/raw/main/assignments/data/A2_y_train_low.csv"
)

A2_X_test_high = pd.read_csv(
    "https://github.com/kylebradbury/ids705/raw/main/assignments/data/A2_X_test_high.csv"
)
A2_y_test_high = pd.read_csv(
    "https://github.com/kylebradbury/ids705/raw/main/assignments/data/A2_y_test_high.csv"
)

A2_X_test_low = pd.read_csv(
    "https://github.com/kylebradbury/ids705/raw/main/assignments/data/A2_X_test_low.csv"
)
A2_y_test_low = pd.read_csv(
    "https://github.com/kylebradbury/ids705/raw/main/assignments/data/A2_y_test_low.csv"
)
```

```
In [ ]: # Question (c)

        # fit classification in training data
        # convert data to numpy arrays
        X_train_low = A2_X_train_low.to_numpy()
        y_train_low = A2_y_train_low.to_numpy().ravel()
        X_test_low = A2_X_test_low.to_numpy()
        y_test_low = A2_y_test_low.to_numpy().ravel()

        knn = Knn()
        knn.fit(X_train_low, y_train_low)
```

```
# timer
start_time_low = time.time()

# predictions
y_pred_low = knn.predict(X_test_low, k=5)

# calculate the time and accuracy
time_taken_low = time.time() - start_time_low
acc_low = accuracy(y_test_low, y_pred_low)

print(f"Low Time taken for KNN: {time_taken_low} seconds")
print(f"Low Accuracy for KNN: {acc_low}")


# convert data to numpy arrays
X_train_high = A2_X_train_high.to_numpy()
y_train_high = A2_y_train_high.to_numpy().ravel()
X_test_high = A2_X_test_high.to_numpy()
y_test_high = A2_y_test_high.to_numpy().ravel()

knn.fit(X_train_high, y_train_high)

# timer
start_time_high = time.time()

# predictions
y_pred_low = knn.predict(X_test_low, k=5)

# calculate the time and accuracy
time_taken_high = time.time() - start_time_high
acc_high = accuracy(y_test_low, y_pred_low)

print(f"High Time taken for KNN: {time_taken_high} seconds")
print(f"High Accuracy for KNN: {acc_high}")
```

```
Low Time taken for KNN: 0.05380082130432129 seconds
Low Accuracy for KNN: 0.5005005005005005
High Time taken for KNN: 0.03771519660949707 seconds
High Accuracy for KNN: 0.5005005005005005
```

In [ ]:
```
# Question (d)

from sklearn.neighbors import KNeighborsClassifier

# initialize the sklearn KNN classifier
sklearn_knn = KNeighborsClassifier(n_neighbors=5)
```

```
# train and test on low dimensional data
sklearn_knn.fit(X_train_low, y_train_low)  # fit the model on the training data
start_time_low_sklearn = time.time()
y_pred_low_sklearn = sklearn_knn.predict(
    X_test_low
)  # make predictions on the test data
time_taken_low_sklearn = time.time() - start_time_low_sklearn
accuracy_low_sklearn = accuracy(y_test_low, y_pred_low_sklearn)  # calculate accuracy
print(f"Low Time taken for Scikit-learn: {time_taken_low_sklearn:.2f} seconds")
print(f"Low Accuracy for Scikit-learn: {accuracy_low_sklearn:.2f}")

# train and test on high dimensional data
sklearn_knn.fit(X_train_high, y_train_high)  # fit the model on the training data
start_time_high_sklearn = time.time()
y_pred_high_sklearn = sklearn_knn.predict(
    X_test_high
)  # make predictions on the test data
time_taken_high_sklearn = time.time() - start_time_high_sklearn
accuracy_high_sklearn = accuracy(y_test_high, y_pred_high_sklearn)  # calculate accuracy
print(f"High Time taken for Scikit-learn: {time_taken_high_sklearn:.2f} seconds")
print(f"High Accuracy for Scikit-learn: {accuracy_high_sklearn:.2f}")
```

```
Low Time taken for Scikit-learn: 0.01 seconds
Low Accuracy for Scikit-learn: 0.92
High Time taken for Scikit-learn: 0.02 seconds
High Accuracy for Scikit-learn: 0.99
```

(e)

> The drawbacks of slow prediction can lead to costly problems, risks, and losses, especially in real-time applications and decision-making scenarios. For example, cloud platforms often use a pay-as-you-go method; therefore, slow inference can lead to higher operational costs. The stock market and other financial sectors could be threatened by fraud due to a slow prediction process.

> In cases where there is a need to continuously deal with large-scale datasets, slow testing (inference) can be more problematic than slow training. This is because delays may cause data loss, leading to significant challenges in data processing and analysis.

5

[20 points] Bias-variance tradeoff: exploring the tradeoff with a KNN classifier. This exercise will illustrate the impact of the bias-variance tradeoff on classifier performance by investigating how model flexibility impacts classifier decision boundaries. For this problem, please us Scikit-learn's KNN implementation rather than your own implementation, as you did at the end of the last question.

(a) Create a synthetic dataset (with both features and targets). Use the `make_moons` module with the parameter `noise=0.35` to generate 1000 random samples.

(b) Visualize your data: scatterplot your random samples with each class in a different color.

(c) Create 3 different data subsets by selecting 100 of the 1000 data points at random three times (with replacement). For each of these 100-sample datasets, fit three separate k-Nearest Neighbor classifiers with: $k = \{1, 25, 50\}$. This will result in 9 combinations (3 datasets, each with 3 trained classifiers).

(d) For each combination of dataset and trained classifier plot the decision boundary (similar in style to Figure 2.15 from *Introduction to Statistical Learning*). This should form a 3-by-3 grid. Each column should represent a different value of k and each row should represent a different dataset.

(e) What do you notice about the difference between the decision boundaries in the rows and the columns in your figure? Which decision boundaries appear to best separate the two classes of data with respect to the training data? Which decision boundaries vary the most as the training data change? Which decision boundaries do you anticipate will generalize best to unseen data and why?

(f) Explain the bias-variance tradeoff using the example of the plots you made in this exercise and its implications for training supervised machine learning algorithms.

Notes and tips for plotting decision boundaries (as in part d):

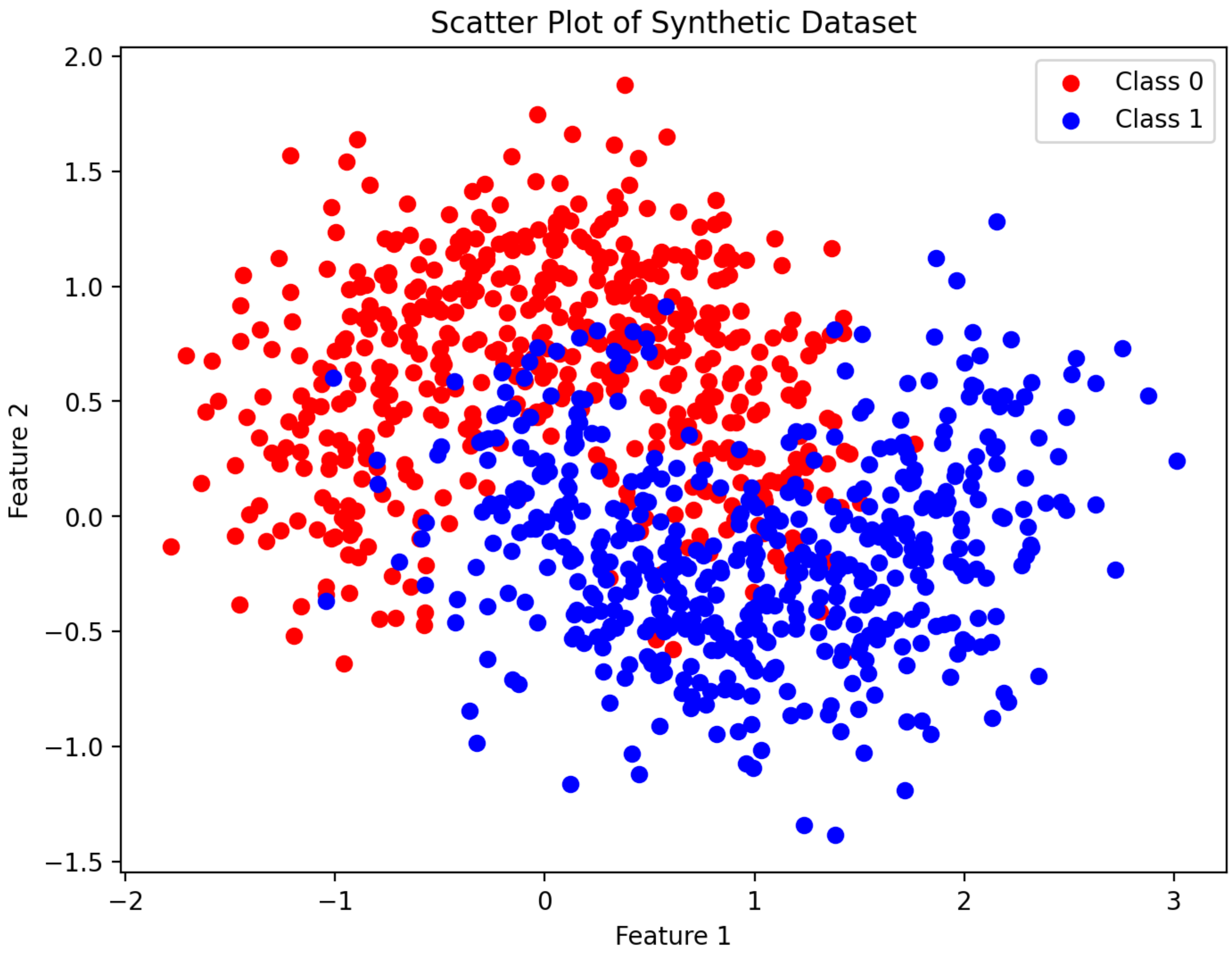
- *Resource for plotting decision boundaries with meshgrid and contour: https://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html*
- If you would like to change the colors of the background, and do not like any of the existing cmap available in matplotlib, you can make your own cmap using the 2 sets of rgb values. Sample code (replace r, g, b with respective rgb values):

In [ ]:
```
# from matplotlib.colors import LinearSegmentedColormap

# newcmp = LinearSegmentedColormap.from_list(
#     "new", [(r / 255, g / 255, b / 255), (r / 255, g / 255, b / 255)], N=2
# )
```

ANSWER

```
In [ ]: # Question (a)

        from sklearn.datasets import make_moons

        X, y = make_moons(n_samples=1000, noise=0.35, random_state=42)
```

```
In [ ]: # Question (b)

        import matplotlib.pyplot as plt

        plt.figure(figsize=(8, 6))
        plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color="red", label="Class 0")
        plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color="blue", label="Class 1")
        plt.title("Scatter Plot of Synthetic Dataset")
        plt.xlabel("Feature 1")
        plt.ylabel("Feature 2")
        plt.legend()
        plt.show()
```



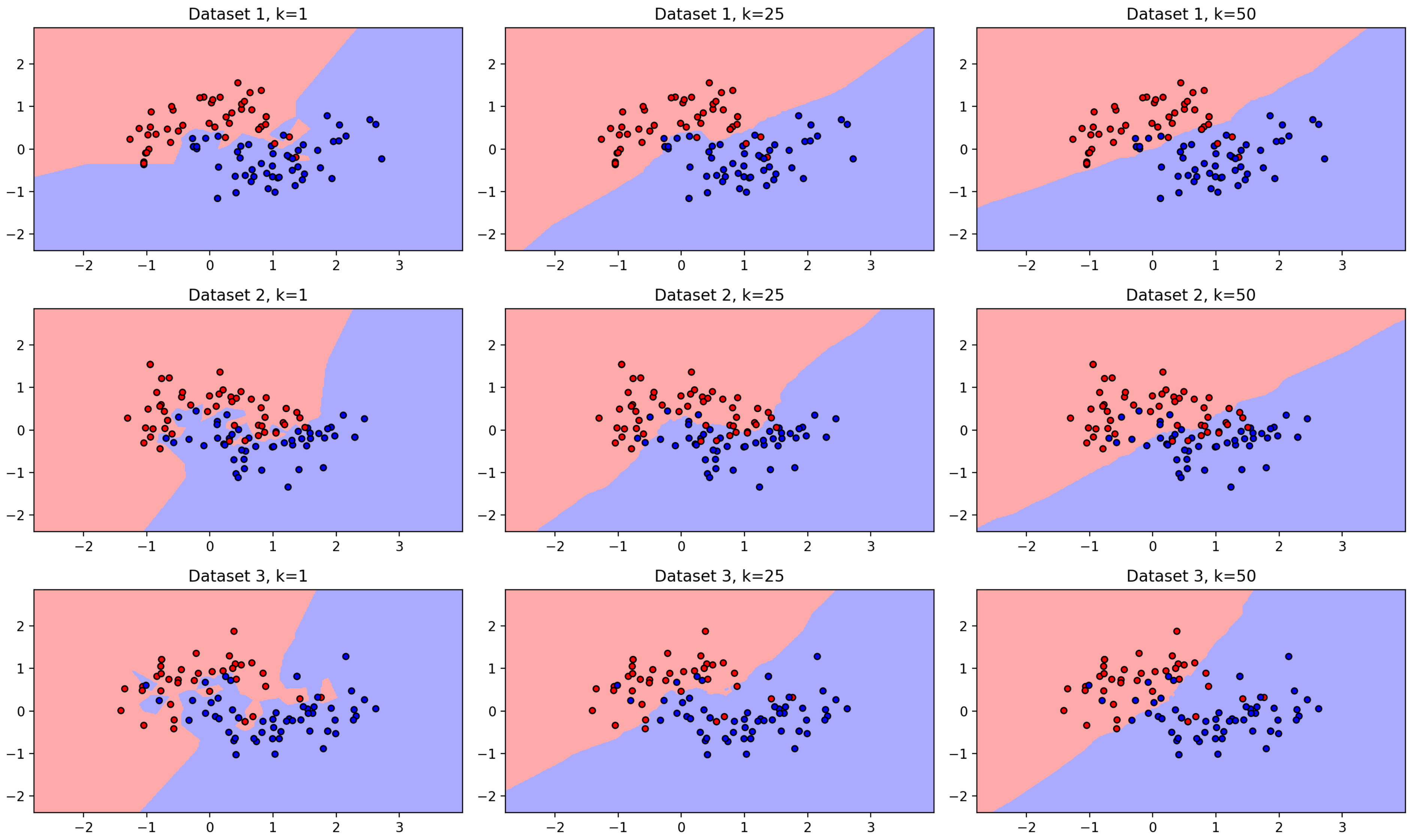
```
In [ ]: # Question (c)

        from sklearn.neighbors import KNeighborsClassifier

        # sample random subsets
        np.random.seed(42)
        indices = np.arange(len(X))
        subsets = [np.random.choice(indices, 100, replace=True) for _ in range(3)]

        # values of k
        k_values = [1, 25, 50]
```

```
# store classifiers
classifiers = []

for subset in subsets:
    X_subset = X[subset]
    y_subset = y[subset]
    classifiers_subset = []
    for k in k_values:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_subset, y_subset)
        classifiers_subset.append(knn)
    classifiers.append(classifiers_subset)
print(classifiers)
```

```
[[KNeighborsClassifier(n_neighbors=1), KNeighborsClassifier(n_neighbors=25), KNeighborsClassifier(n_neighbors=50)], [KN
eighborsClassifier(n_neighbors=1), KNeighborsClassifier(n_neighbors=25), KNeighborsClassifier(n_neighbors=50)], [KNeigh
borsClassifier(n_neighbors=1), KNeighborsClassifier(n_neighbors=25), KNeighborsClassifier(n_neighbors=50)]]
```

In [ ]:

```
# Question (d)

from matplotlib.colors import ListedColormap

# colors and colormap
cmap_light = ListedColormap(["#FFAAAA", "#AAAAFF"])
cmap_bold = ListedColormap(["#FF0000", "#0000FF"])

# mesh grid
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))

plt.figure(figsize=(15, 9))

for i, subset in enumerate(subsets):
    for j, k in enumerate(k_values):
        ax = plt.subplot(3, len(k_values), i * len(k_values) + j + 1)

        Z = classifiers[i][j].predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)
        ax.contourf(xx, yy, Z, cmap=cmap_light)

        ax.scatter(
            X[subset][:, 0],
            X[subset][:, 1],
            c=y[subset],
            cmap=cmap_bold,
            edgecolor="k",
            s=20,
        )
        ax.set_title(f"Dataset {i + 1}, k={k}")

plt.tight_layout()
plt.show()
```



(e) Analyze the decision boundaries:

> - **Rows**: Represent different subsets of the data.

> - **Columns**: Represent different values of k.
> - Observe how the decision boundaries change with different k values and different subsets. Lower values of k typically result in more complex decision boundaries (more prone to overfitting), while higher k values produce smoother, simpler boundaries (may underfit).
> - Decision boundaries that best separate the classes with respect to the training data may not necessarily generalize best to unseen data. Often, a balance is needed to avoid overfitting or underfitting.

(f) Discuss the bias-variance tradeoff:

> - Lower values of k (more flexible models) tend to have low bias but high variance, fitting the training data closely but potentially overfitting and not generalizing well to new data.
> - Higher values of k (less flexible models) tend to have high bias but low variance, underfitting the training data but potentially more stable and generalizable to new data.
> - The key in supervised learning is finding the right balance between bias and variance, which is context-dependent and may require validation techniques like cross-validation to assess model performance on unseen data.

6

[18 points] Bias-variance trade-off II: Quantifying the tradeoff. This exercise explores the impact of the bias-variance tradeoff on classifier performance by looking at the performance on both training and test data.

Here, the value of k determines how flexible our model is.

(a) Using the function created earlier to generate random samples (using the `make_moons` function setting the `noise` parameter to 0.35), create a new set of 1000 random samples, and call this dataset your test set and the previously created dataset your training set.

(b) Train a kNN classifier on your training set for $k = 1, 2, \ldots 500$. Apply each of these trained classifiers to both your training dataset and your test dataset and plot the classification error (fraction of incorrect predictions).

(c) What trend do you see in the results?

(d) What values of k represent high bias and which represent high variance?

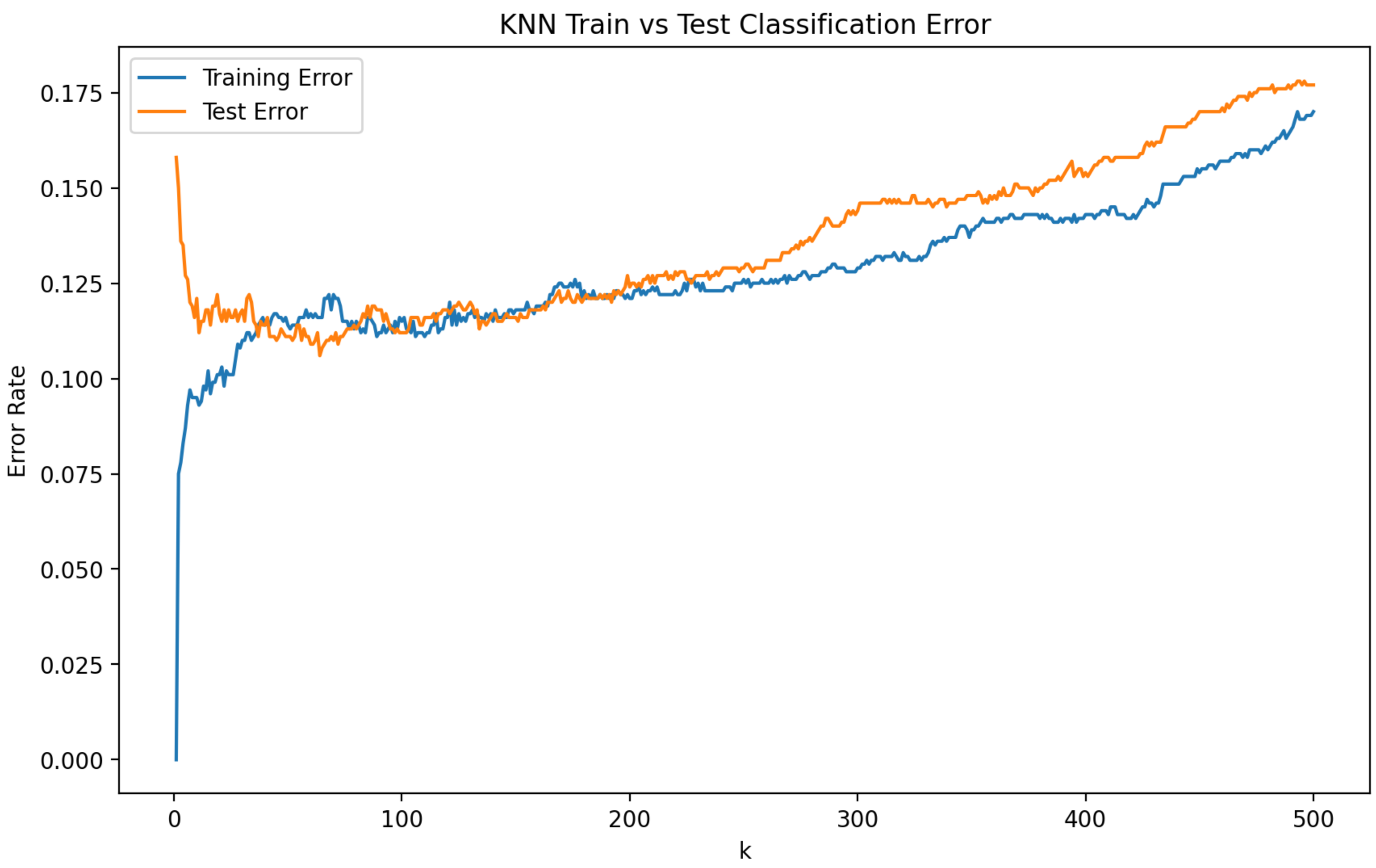
(e) What is the optimal value of k and why?

(f) In KNN classifiers, the value of k controls the flexibility of the model - what controls the flexibility of other models?

ANSWER

In [ ]:
```
# Question (a)

X_test, y_test = make_moons(n_samples=1000, noise=0.35, random_state=52)
```

In [ ]:
```
# Question (b)

train_errors = []
test_errors = []

for k in range(1, 501):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X, y)  # X, y are from the training dataset

    y_train_pred = knn.predict(X)
    y_test_pred = knn.predict(X_test)

    train_errors.append(np.mean(y_train_pred != y))
    test_errors.append(np.mean(y_test_pred != y_test))

plt.figure(figsize=(10, 6))
plt.plot(range(1, 501), train_errors, label="Training Error")
plt.plot(range(1, 501), test_errors, label="Test Error")
plt.title("KNN Train vs Test Classification Error")
plt.xlabel("k")
plt.ylabel("Error Rate")
plt.legend()
plt.show()
```



(c) Expected trend in the results:

- As k increases, the model becomes less flexible, and the training error increases.
- For very low values of k, the model is too flexible and overfits the data, leading to high test error.
- As k increases, the test error initially decreases (up to a certain point) as the model generalizes better.
- After a certain point, as k increases further, the model becomes too simple, starts to underfit, and the test error might increase.

(d) High bias and high variance in terms of k:

- Low values of k (e.g., $k = 1$) represent low bias but high variance (overfitting).

- High values of k (e.g., $k = 500$) represent high bias but low variance (underfitting).

(e) Optimal value of k:

- The optimal value of k is 30 from the plot, and is the one that balances bias and variance, minimizing the test error. It's the value where the test error is at its lowest before it starts to increase due to underfitting.

(f) Flexibility control in other models:

- In other models, flexibility can be controlled by various hyperparameters:
  - **Decision Trees**: The depth of the tree, minimum samples per leaf, etc.
  - **Regularized Regression (Lasso, Ridge)**: The regularization parameter (alpha).
  - **Neural Networks**: The number of layers, the number of neurons in each layer, regularization terms, etc.

Each model has its own set of parameters that control its complexity and flexibility, directly influencing the bias-variance tradeoff.

7

[18 points] Linear regression and nonlinear transformations. Linear regression can be used to model nonlinear relationships when feature variables are properly transformed to represent the nonlinearities in the data. In this exercise, you're given training and test data contained in files "A2_Q7_train.csv" and "A2_Q7_test.csv" in the "data" folder for this assignment. Your goal is to develop a regression algorithm from the training data that performs well on the test data.

Hint: Use the scikit learn LinearRegression module.

(a) Create a scatter plot of your training data.

(b) Estimate a linear regression model ($y = a_0 + a_1 x$) for the training data and calculate both the R^2 value and mean square error for the fit of that model for the training data. Also provide the equation representing the estimated model (e.g. $y = a_0 + a_1 x$, but with the estimated coefficients inserted. Consider this your baseline model against which you will compare other model options. *Evaluating performance on the training data is not a measure of how well this model would generalize to unseen data. We will evaluate performance on the test data once we see our models fit the training data decently well.*

(c) If features can be nonlinearly transformed, a linear model may incorporate those non-linear feature transformation relationships in the training process. From looking at the scatter plot of the training data, choose a transformation of the predictor variable, x that may make sense for these data. This will be a multiple regression model of the form $y = a_0 + a_1 z_1 + a_2 z_2 + \ldots + a_n z_n$. Here z_i could be any transformations of x - perhaps it's $\frac{1}{x}$, $log(x)$, $sin(x)$, x^k (where k is any power of your choosing). Provide the estimated equation for this multiple regression model (e.g. if you chose your predictors to be $z_1 = x$ and $z_2 = log(x)$, your model would be of the form $y = a_0 + a_1 x + a_2 log(x)$. Also provide the R^2 and mean square error of the fit for the training data.

(d) Visualize the model fit to the training data. Using both of the models you created in parts (b) and (c), plot the original data (as a scatter plot) AND the curves representing your models (each as a separate curve) from (b) and (c).

(e) Now its time to compare your models and evaluate the generalization performance on held out test data. Using the models above from (b) an (c), apply them to the test data and estimate the R^2 and mean square error of the test dataset.

(f) Which models perform better on the training data, and which on the test data? Why?

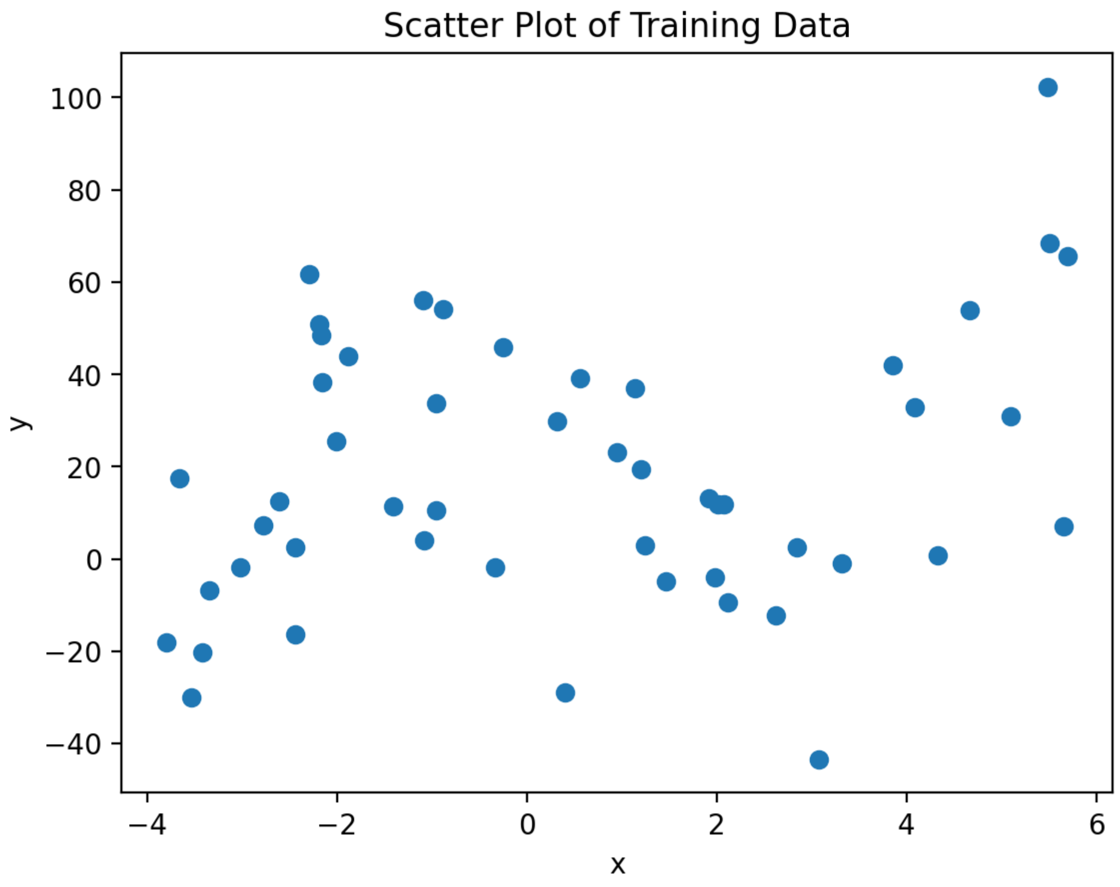
(g) Imagine that the test data were significantly different from the training dataset. How might this affect the predictive capability of your model? How would the accuracy of generalization performance be impacted? Why?

To help get you started - here's some code to help you load in the data for this exercise (you'll just need to update the path):

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

path = "https://github.com/kylebradbury/ids705/raw/main/assignments/data/"
train = pd.read_csv(path + "A2_Q7_train.csv")
test = pd.read_csv(path + "A2_Q7_test.csv")

x_train = train.x.values
y_train = train.y.values

x_test = test.x.values
y_test = test.y.values
```

ANSWER

In [ ]:
```
# Question (a)

# scatter plot for traning data
plt.scatter(x_train, y_train)
plt.title("Scatter Plot of Training Data")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



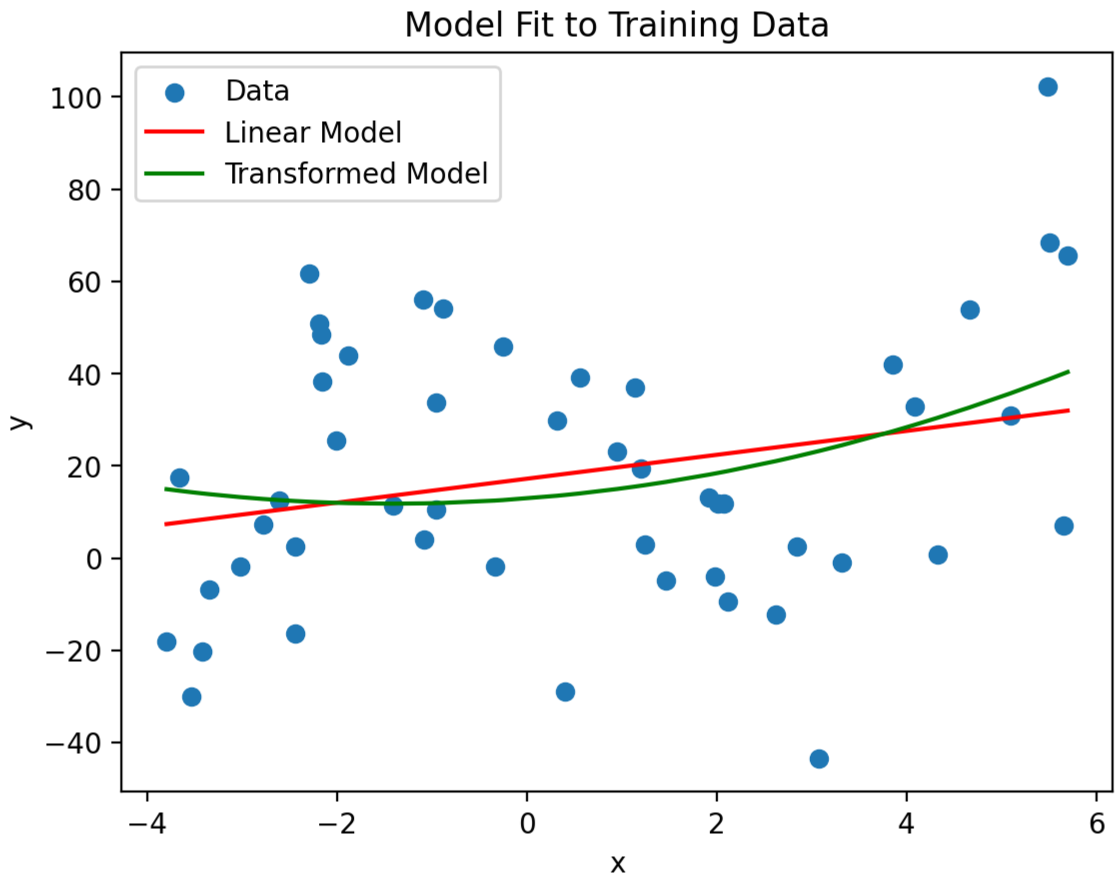
```
In [ ]: # Question (b)

        # estimate the linear model for traning data
        from sklearn.linear_model import LinearRegression
        from sklearn.metrics import mean_squared_error, r2_score

        # train the model
        model = LinearRegression()
        model.fit(x_train.reshape(-1, 1), y_train)

        # predictions
        y_pred_train = model.predict(x_train.reshape(-1, 1))

        # R^2 and MSE
        r2 = r2_score(y_train, y_pred_train)
        mse = mean_squared_error(y_train, y_pred_train)

        print(f"Model_1: y = {model.intercept_:.2f} + {model.coef_[0]:.2f} * x")
print(f"R^2: {r2}")
print(f"MSE: {mse}")
```

```
Model_1: y = 17.20 + 2.59 * x
R^2: 0.06486123304769698
MSE: 791.4167471701106
```

In [ ]:
```
# Question (c)

# transform the predictor and estimate the multiple models
x_train_transformed = np.column_stack((x_train, x_train**2))

# train the model
model_transformed = LinearRegression()
model_transformed.fit(x_train_transformed, y_train)

# predictions
y_pred_train_transformed = model_transformed.predict(x_train_transformed)

# R^2 and MSE
r2_transformed = r2_score(y_train, y_pred_train_transformed)
mse_transformed = mean_squared_error(y_train, y_pred_train_transformed)

print(
    f"Model_2: y = {model_transformed.intercept_:.2f} + {model_transformed.coef_[0]:.2f} * x + {model_transformed.coef_
)
print(f"R^2: {r2_transformed}")
print(f"MSE: {mse_transformed}")
```

```
Model_2: y = 12.94 + 1.61 * x + 0.56 * x^2
R^2: 0.08529040403484756
MSE: 774.1273473276406
```

In [ ]:
```
# Question (d)

# plot for fitting the traning data
plt.scatter(x_train, y_train, label="Data")
plt.plot(
    np.sort(x_train),
    y_pred_train[np.argsort(x_train)],
    label="Linear Model",
    color="red",
)
plt.plot(
    np.sort(x_train),
    y_pred_train_transformed[np.argsort(x_train)],
    label="Transformed Model",
    color="green",
)
plt.title("Model Fit to Training Data")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```



In [ ]:

```
# Question (e)

# compare the models to the test data
y_pred_test = model.predict(x_test.reshape(-1, 1))
x_test_transformed = np.column_stack((x_test, x_test**2))
y_pred_test_transformed = model_transformed.predict(x_test_transformed)

# R^2 and MSE
r2_test = r2_score(y_test, y_pred_test)
```

```
mse_test = mean_squared_error(y_test, y_pred_test)
r2_test_transformed = r2_score(y_test, y_pred_test_transformed)
mse_test_transformed = mean_squared_error(y_test, y_pred_test_transformed)

print(f"Linear Model - Test R^2: {r2_test}, Test MSE: {mse_test}")
print(
    f"Transformed Model - Test R^2: {r2_test_transformed}, Test MSE: {mse_test_transformed}"
)
```

```
Linear Model - Test R^2: -0.13289928472598378, Test MSE: 1116.6632365616085
Transformed Model - Test R^2: -0.12025627926664328, Test MSE: 1104.2014232420706
```

Question (f)

> On the training data, Model 2 (the transformed model) performs marginally better than Model 1 (the linear model). This is indicated by a slightly higher R^2 value (0.085 for Model 2 vs 0.064 for Model 1) and a lower Mean Squared Error (MSE) (774.13 for Model 2 vs 791.42 for Model 1). The higher R^2 value suggests that the transformed model explains a slightly higher proportion of the variance in the dependent variable. The lower MSE indicates that the transformed model's predictions are, on average, closer to the actual values. However, when evaluated on the test data, both models perform poorly, as indicated by the negative R^2 values. Negative R^2 values imply that the models perform worse than a horizontal line drawn at the mean of the dependent variable. This suggests that neither model generalizes well to unseen data. Despite this, the transformed model (Model 2) performs slightly better than the linear model (Model 1), as indicated by a less negative R^2 value (-0.120 for Model 2 vs -0.132 for Model 1) and a slightly lower MSE (1104.20 for Model 2 vs 1116.66 for Model 1).

Question (g)

> If the test data were significantly different from the training dataset, it could severely impact the predictive capability of the model and the accuracy. For example, the model may not be able to correctly understand and predict outcomes for the test data; the model has learned the noise or specific patterns in the training data too perfect (overfitting), it will perform poorly on the test data, and vice versa. The accuracy of generalization performance would decrease. The metrics indicating the model's performance (like R^2, MSE) would show that the model does not predict the outcomes accurately.This is because the model's understanding, derived from the training data, does not apply well to the test data.