Assignment_2_Solutions

February 10, 2020

1 Assignment 2 - Machine Learning Basics

1.1 SOLUTIONS

1.2 Learning Objectives:

This assignment will provide structured practice to help enable you to... - Implement a k-nearest neighbors machine learning algorithm from scratch in a style similar to that of popular machine learning tools like `scikit-learn` - Apply basic regression and classification supervised learning techniques to data and evaluate the performance of those methods - Understand the bias-variance tradeoff and the impact of model flexibility algorithm performance and model selections

[2]:
```
# MAC USERS TAKE NOTE:
# For clearer plots in Jupyter notebooks on macs, run the following line of␣
 ↪code:
# %config InlineBackend.figure_format = 'retina'
```

2 Conceptual Questions

2.1 1

[5 points] For each part (a) through (d), 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. Since there is a large sample, a more flexible approach will have ample examples to help prevent overfit.

2. Worse. Here a flexible method needs to represent data in a very high dimensional space, but has few observations to do so. A flexible method is likely to overfit in this case.

3. Better. We will need a more flexible model to be able to fit a complex relationship.

4. Worse. If the variance of the error terms (i.e. the noise in the model) is high, then more flexible methods are likely to fit to the noise rather than the underlying patterns in the data. In this situation, a less flexible model is desirable.

2.2 2

[5 points] For each of the following, (i) explain if each scenario is a classification or regression problem, (ii) indicate whether we are most interested in inference or prediction for that problem, 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 interesting 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)

i Regression. The target variable, CEO salary, is continuous.

ii Inference. In this case, we want to understand the factors that impact salary, not predict the salary.

iii $n = 500$ firms in the US; $p = 3$, the features are: profit, number of employees, and industry.

(b)

i Classification. The target variable, success/failure of a product, is categorial.

ii Prediction. We want to predict whether or not the product will succeed.

iii $n = 20$ similar products that were previously launched; $p = 13$, the features are: price charged for the product, the marketing budget, competition price, and ten other variables.

(c)

i Regression. The target variable, the percentage change in the U.S. dollar, is continuous.

ii Prediction. We're tring to predict the percentage change.

iii $n = 52$ weeks of 2012; $p = 3$: percent change in the U.S. market, percent change in the British market, and the percent change in German market.

3 Practical Questions

3.1 3

[10 points] Classification II. The table below provides a training dataset containing six observations ($n = 6$), three predictors ($p = 3$), and one qualitative response variable.

Table 1. 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

Obs.	x_1	x_2	x_3	y
6	1	1	1	Red

We want to use this dataset to make a prediction for y when $x_1 = x_2 = x_3 = 0$ using K-nearest neighbors. You are given some code below to get you started.

(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 with $K = 1$? Why?

(c) What is our prediction with $K = 3$? 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?

[3]:
```
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) Computing the Euclidean distance of each training data point to the test point $x = [0, 0, 0]^T$.

[4]:
```
x          = np.array([0,0,0])
difference = (X - x)**2
distance   = np.sqrt(np.sum(difference,axis=1))

print(distance)
```

```
[3.         2.         3.16227766 2.23606798 1.41421356 1.73205081]
```

Table 2. Distance from the origin to each training point to the test point $\mathbf{x}_{test} = [0, 0, 0]^T$.

Obs.	Distance to $\mathbf{x}_{test}$
1	3.00
2	2.00
3	3.16
4	2.24
5	1.41
6	1.73

(b) Blue. Our prediction for $k = 1$ is the label corresponding to observation 5 since it's the closest point, and the label of that point is blue.

(c) Red. Our prediction for $k = 3$ is the majority vote of the labels of the three closest observation, which are observations 2, 5, and 6, which are red, blue, and red, respectively. Therefore, our

prediction is red in this case.

(d) If the Bayes decision boundary is highly nonlinear, then we will require a model with greater flexibility in order to enable the model to fit adequately (assuming we have enough data to accommodate this). A lower value of k will result in this greater flexibility.

3.2 4

[20 points] Classification I: Creating a classification algorithm.

(a) Build a working version of a binary kNN classifier using the skeleton code below.

(b) Load the datasets to be evaluated here. Each includes training features (**X**), and test features (**y**) for both a low dimensional ($p = 2$ features/predictors) and a high dimensional ($p = 100$ features/predictors). For each of these datasets there are $n = 100$ 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. Calculate the time it takes to make the predictions in each case and the overall accuracy of each set of test data predictions.

(d) Compare your implementation's accuracy and computation time to the scikit learn KNeighborsClassifier class. How do the results and speed compare?

(e) Some supervised learning algorithms are more computationally intensive during training than testing. What are the drawbacks of the prediction process being slow?

ANSWER:

(a) A working version of a binary kNN classifier is shown below:

[5]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

class Knn:

    def __init__(self):
        # Save the training data to properties of this class
        self.x_train = []
        self.y_train = []

    def fit(self, x, y):
        # Save the training data to properties of this class
        self.x_train = x
        self.y_train = y
        self.n_values = len(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
        for i,v in enumerate(x):
            diff = self.x_train - v
```

```
            distance = np.sum(diff**2,axis=1)
            distance_sorted = np.sort(distance)
            indices = np.where(distance <= distance_sorted[k-1])
            vote = sum(self.y_train[indices])
            y_hat.append(vote > k/2)

        # Return the estimated targets
        return np.array(y_hat)

def metrics(y,y_hat):
    nvalues = len(y)
    accuracy = sum(y == y_hat) / nvalues
    return accuracy
```

(b) I load the datasets to be evaluated here:

[6]:
```
# Set the data directories
dir_data = './data/'

# Load the training data
X_train_high = pd.read_csv(dir_data + 'A2_X_train_high.csv',header=None).values
X_train_low  = pd.read_csv(dir_data + 'A2_X_train_low.csv', header=None).values
X_test_high  = pd.read_csv(dir_data + 'A2_X_test_high.csv', header=None).values
X_test_low   = pd.read_csv(dir_data + 'A2_X_test_low.csv',  header=None).values

# Load the test data
y_train_low  = pd.read_csv(dir_data + 'A2_y_train_low.csv', header=None).values
y_train_high = pd.read_csv(dir_data + 'A2_y_train_high.csv',header=None).values
y_test_low   = pd.read_csv(dir_data + 'A2_y_test_low.csv',  header=None).values
y_test_high  = pd.read_csv(dir_data + 'A2_y_test_high.csv', header=None).values
```

(c) I train the classifier on first the low dimensional dataset and then the high dimensional dataset with $k = 5$.

[7]:
```
import time

k = 5
knn = Knn()

# Run the high dimensional case
t0 = time.time()
knn.fit(X_train_high,y_train_high)

y_hat_high    = knn.predict(X_test_high, k)
accuracy_high = metrics(y_test_high,y_hat_high)

t1 = time.time()
time_high = t1 - t0
print('Accuracy (High Dim) = {:0.4f}. Time = {:0.4f} seconds'.
 ↪format(accuracy_high[0], time_high))
```

```
t0 = time.time()
knn.fit(X_train_low,y_train_high)

y_hat_low    = knn.predict(X_test_low, k)
accuracy_low = metrics(y_test_low,y_hat_low)

t1 = time.time()
time_low = t1 - t0
print('Accuracy (Low  Dim) = {:0.4f}. Time = {:0.4f} seconds'.
 ↪format(accuracy_low[0], time_low))
```

```
Accuracy (High Dim) = 0.9930. Time = 0.2997 seconds
Accuracy (Low  Dim) = 0.9250. Time = 0.0819 seconds
```

(d) I compare my implementation's accuracy and computation time to the scikit learn `KNeighborsClassifier` class.

[8]:
```
from sklearn.neighbors import KNeighborsClassifier
skl_knn = KNeighborsClassifier(n_neighbors=k)

# Run the high dimensional case
t0 = time.time()
skl_knn.fit(X_train_high,y_train_high.T[0]) # y_train_high needs to be␣
 ↪converted into a 1-d array to make this work

y_hat_high    = skl_knn.predict(X_test_high)
accuracy_high = metrics(y_test_high,y_hat_high[np.newaxis].T) # y_hat_high␣
 ↪needs to be a column vector for our simple metrics
                                                              # function to␣
 ↪work properly (good reason to use scikit learn instead)

t1 = time.time()
time_high = t1 - t0
print('Accuracy (High Dim) = {:0.4f}. Time = {:0.4f} seconds'.
 ↪format(accuracy_high[0], time_high))

t0 = time.time()
skl_knn.fit(X_train_low,y_train_low.T[0]) # y_train_high needs to be converted␣
 ↪into a 1-d array to make this work

y_hat_low    = skl_knn.predict(X_test_low)
accuracy_low = metrics(y_test_low,y_hat_low[np.newaxis].T) # y_hat_low needs to␣
 ↪be a column vector for our simple metrics
                                                           # function to work␣
 ↪properly (good reason to use scikit learn instead)

t1 = time.time()
```

```
time_low = t1 - t0
print('Accuracy (Low  Dim) = {:0.4f}. Time = {:0.4f} seconds'.
 ↪format(accuracy_low[0], time_low))
```

```
Accuracy (High Dim) = 0.9930. Time = 0.2098 seconds
Accuracy (Low  Dim) = 0.9250. Time = 0.0350 seconds
```

Although the results are identical, the speed is considerably faster with the scikit-learn implentation for the low dimensional case. The high dimensional case is about equivalent, so no performance improvement was see for more advanced programming implementations.

(e) Often times, predictions need to be made in near-real-time, and slow predictions could be problematic. Additionally, if there were considerably more data points, this process will become much, much worse since the number of comparisons that need to be calculate grow as the number of training points increase. This algorithm, in short, will not scale well into high dimensions, or to high numbers of training data points.

3.3 5

[20 points] Bias-variance tradeoff I: Understanding the tradeoff. This exercise will illustrate the impact of the bias-variance tradeoff on classifier performance by looking at classifier decision boundaries.

(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) 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. For each of these 100-sample datasets, fit three k-Nearest Neighbor classifiers with: $k = \{1, 25, 50\}$. This will result in 9 combinations (3 datasets, with 3 trained classifiers).

(d) For each combination of dataset trained classifier, in a 3-by-3 grid, plot the decision boundary (similar in style to Figure 2.15 from *Introduction to Statistical Learning*). 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 rows and the columns. Which decision boundaries appear to best separate the two classes of data? Which decision boundaries vary the most as the data change?

(f) Explain the bias-variance tradeoff using the example of the plots you made in this exercise.

ANSWER

(a) Generate the synthetic data

[9]:
```
import numpy as np
from sklearn.datasets import make_moons

# Ensures future runs of the code will provide the same result
np.random.seed(1231)

# Generate a training dataset of 1000 samples
noise = 0.35
nsamples = 1000
X_train, y_train = make_moons(n_samples=nsamples, noise=noise)
```

(b) Scatterplot the data

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.colors as colors

color0 = '#121619' # Dark grey
color1 = '#00B050' # Green

samples0 = X_train[y_train==0]
samples1 = X_train[y_train==1]

plt.figure(figsize=(10,10))
plt.plot(samples0[:,0],samples0[:,1],'s',
         color=color0,
         markersize=5,
         markeredgecolor='w',
         markeredgewidth=0.5,
         label = 'Class 0')
plt.plot(samples1[:,0],samples1[:,1],'o',
         color=color1,
         markersize=5,
         markeredgecolor='w',
         markeredgewidth=0.5,
         label = 'Class 1')
plt.grid(True)
plt.axis([-2,3,-1,2])
plt.xlabel('$x_0$')
plt.ylabel('$x_1$')
plt.legend()
plt.show()
```

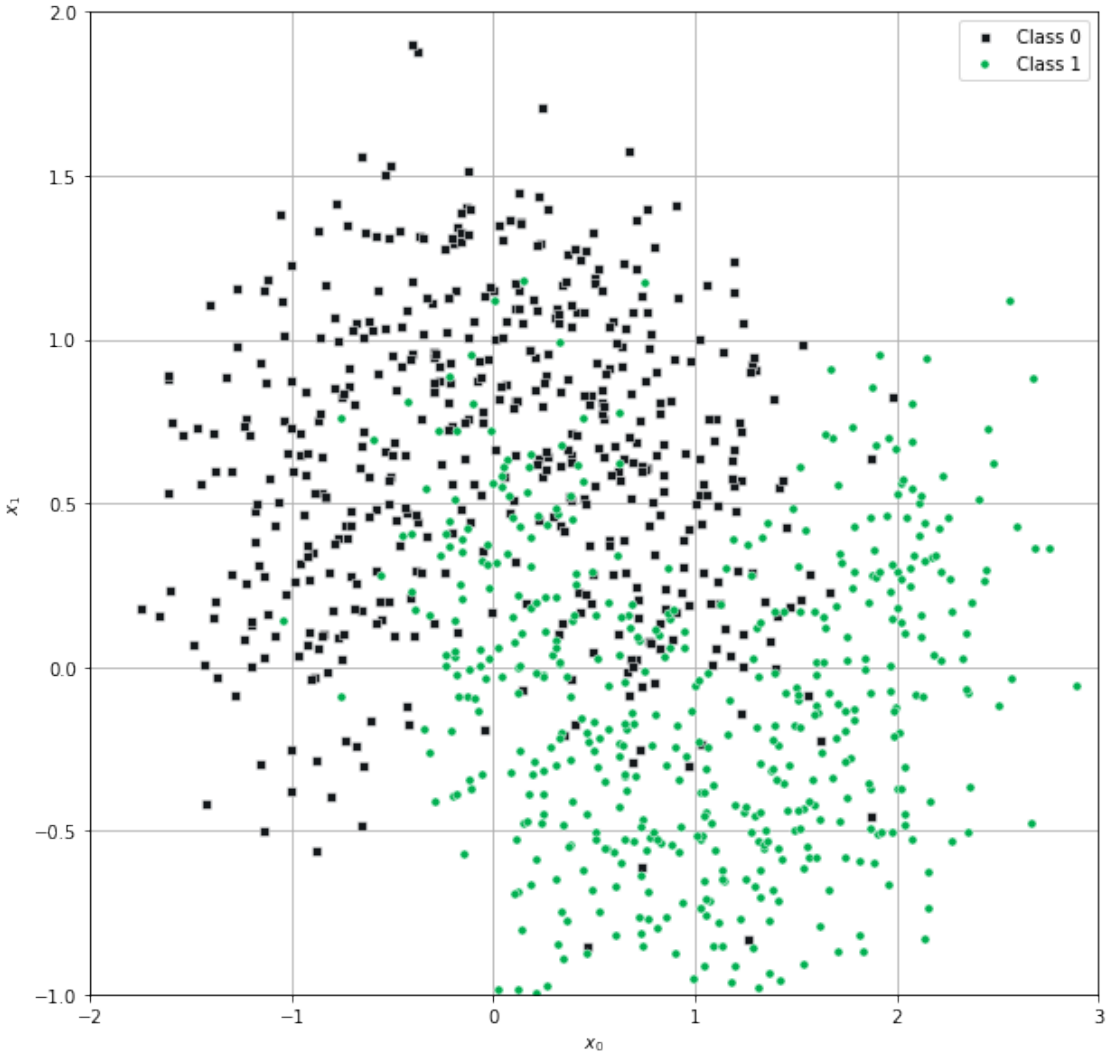

Figure 1: Scatterplot showing the two classes of synthetic data for question 5(b)

(c) and (d) Create the 3-by-3 grid of knn boundaries and datasets

[11]:
```
from sklearn import neighbors

plt.close()

# Custom color code (this is not necessary)
# Function to convert from hex to RGB (removing the hashtag)
def hex2rgb(hex):
    hex = hex.replace('#', '')
    return np.array([int(hex[i:i+2], 16) for i in (0, 2, 4)])

color0_background = '#dcdcdc' # Light grey
color1_background = '#c2ead4' # Light green
```

```
# Create a custom colormap for these plots
initial_color = hex2rgb(color0_background)
final_color   = hex2rgb(color1_background)

nsteps = 2
custom_cmap = np.array([np.linspace(i,j,nsteps) for i,j in␣
 ↪zip(initial_color,final_color)]).T
cm = colors.ListedColormap(custom_cmap/255.0)

# Function to plot each boundary and set of samples
def plot_samples(data,target,matrix,limits,x,y, title=np.empty(0)):
    # Determine positive/negative samples
    samples0 = data[target==0]
    samples1 = data[target==1]

    plt.imshow(np.flipud(matrix),extent=limits,cmap=cm)
    plt.contour(x,y,matrix, 1,
                colors='black',
                linewidths=1,
                linestyles='solid')
    plt.plot(samples0[:,0],samples0[:,1],'s',
             color=color0,
             markersize=5,
             markeredgecolor='w',
             markeredgewidth=0.5)
    plt.plot(samples1[:,0],samples1[:,1],'o',
             color=color1,
             markersize=5,
             markeredgecolor='w',
             markeredgewidth=0.5)
    plt.axis('square')
    plt.axis(limits)
    plt.xlabel('$x_0$')
    plt.ylabel('$x_1$')
    if len(title):
        plt.title(title, fontsize=10)

rows = 3
cols = 3
nsamples = 100

k = [1,25,50]

plt.figure(figsize=(12,8))
for i in range(3):
    # Generate sample data by randomly selection 100 of the points from the␣
 ↪previously created data
```

```
    n_subsamples = 100
    indices      = np.random.choice(np.arange(0,nsamples-1),size=n_subsamples)
    features     = X_train[indices]
    target       = y_train[indices]

    # Plot the decision boundary
    limits = [-1.5,2.5,-1,1.5]
    N = 500
    x   = np.linspace(limits[0],limits[1],N)
    y   = np.linspace(limits[2],limits[3],N)
    X,Y = np.meshgrid(x,y)
    shape = X.shape
    Xv = X.reshape(-1)
    Yv = Y.reshape(-1)
    Pv = np.array([Xv,Yv]).T

    for j in range(3):
        # Estimate a kNN decision boundary with low k
        clf_knn = neighbors.KNeighborsClassifier(n_neighbors=k[j])
        clf_knn.fit(features,target)
        predictions = clf_knn.predict(Pv)
        predictions_knn = predictions.reshape(shape)

        # Plot the kNN decision boundary for low k
        plt.subplot(rows,cols,i*cols + j+1)
        plot_samples(features,target,predictions_knn,limits,x,y, title='k={}'.
 ↪format(k[j]))

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

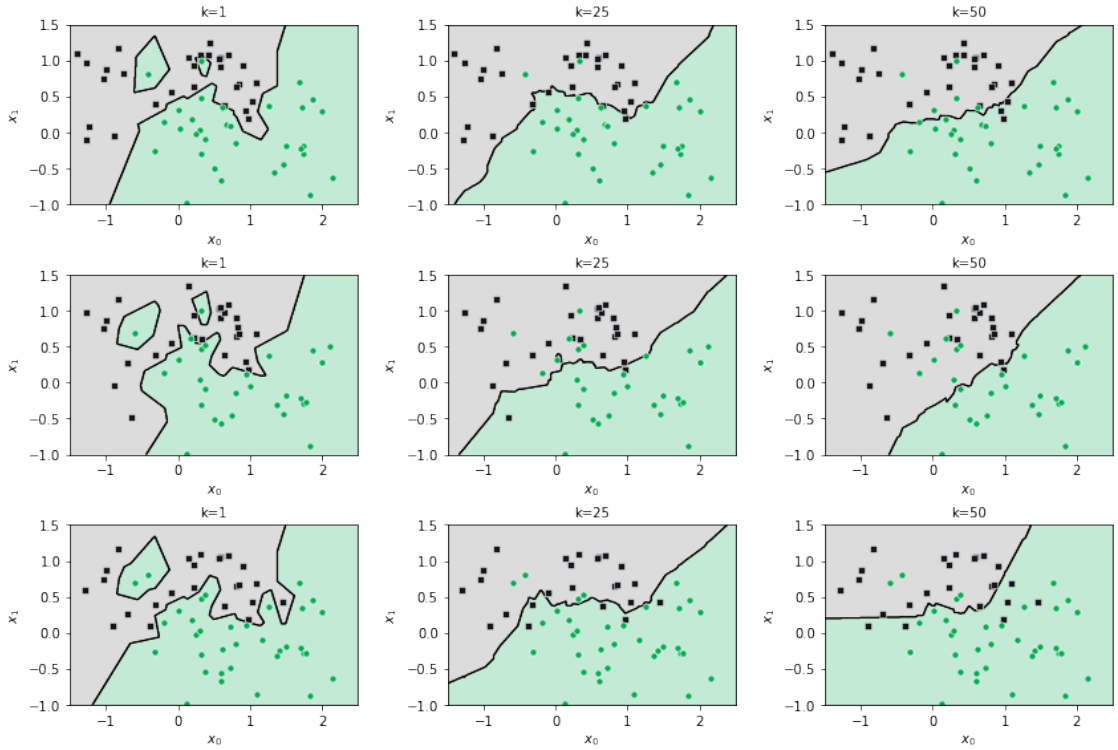

Figure 2: Scatterplots of training data and KNN decision boundaries for question 5 (c) and (d). Each column represents a value of k = 1, 25, or 50 (left to right) and the rows each represent a different sample of the training data. The two classes are presented in black (Class 0) and green (Class 1)

OPTIONAL FOR INTERSTED READERS: Alternative plotting approach: In case you're interested in how to make this plot "prettier", the code below replicates the last piece of code, but using the "axes_grid1" module.

[12]:
```
# This replicates the answer above, but with a more compact plotting approach
from sklearn import neighbors

from mpl_toolkits.axes_grid1 import Grid

# Custom color code (this is not necessary)
# Function to convert from hex to RGB (removing the hashtag)
def hex2rgb(hex):
    hex = hex.replace('#', '')
    return np.array([int(hex[i:i+2], 16) for i in (0, 2, 4)])

color0_background = '#dcdcdc' # Light grey
color1_background = '#c2ead4' # Light green

# Create a custom colormap for these plots
initial_color = hex2rgb(color0_background)
final_color   = hex2rgb(color1_background)
```

```
nsteps = 2
custom_cmap = np.array([np.linspace(i,j,nsteps) for i,j in␣
 ↪zip(initial_color,final_color)]).T
cm = colors.ListedColormap(custom_cmap/255.0)

# Function to plot each boundary and set of samples
def plot_samples(ax,data,target,matrix,limits,x,y, title=np.empty(0)):
    # Determine positive/negative samples
    samples0 = data[target==0]
    samples1 = data[target==1]

    ax.imshow(np.flipud(matrix),extent=limits,cmap=cm)
    ax.contour(x,y,matrix, 1,
                colors='black',
                linewidths=1,
                linestyles='solid')
    ax.plot(samples0[:,0],samples0[:,1],'s',
             color=color0,
             markersize=5,
             markeredgecolor='w',
             markeredgewidth=0.5)
    ax.plot(samples1[:,0],samples1[:,1],'o',
             color=color1,
             markersize=5,
             markeredgecolor='w',
             markeredgewidth=0.5)
    ax.axis('square')
    ax.axis(limits)
    ax.set_xlabel('$x_0$')
    ax.set_ylabel('$x_1$')
    if len(title):
        ax.set_title(title, fontsize=10)

rows = 3
cols = 3
nsamples = 100

k_values = [1,25,50]



fig = plt.figure(figsize=(12, 8))
grid = Grid(fig, rect=111, nrows_ncols=(3,3), axes_pad=0.25, label_mode='L')

axis_index = 0
for row in range(3):
```

```
    # Generate sample data by randomly selection 100 of the points from the␣
 ↪previously created data
    n_subsamples = 100
    indices      = np.random.choice(np.arange(0,nsamples-1),size=n_subsamples)
    features     = X_train[indices]
    target       = y_train[indices]

    # Plot the decision boundary
    limits = [-1.5,2.5,-1,1.5]
    N = 500
    x   = np.linspace(limits[0],limits[1],N)
    y   = np.linspace(limits[2],limits[3],N)
    X,Y = np.meshgrid(x,y)
    shape = X.shape
    Xv = X.reshape(-1)
    Yv = Y.reshape(-1)
    Pv = np.array([Xv,Yv]).T

    for k in k_values:

        # Estimate a kNN decision boundary with low k
        clf_knn = neighbors.KNeighborsClassifier(n_neighbors=k)
        clf_knn.fit(features,target)
        predictions = clf_knn.predict(Pv)
        predictions_knn = predictions.reshape(shape)

        # Plot the kNN decision boundary for low k
        ax = grid[axis_index]
        title = []
        if row == 0:
            title='k={}'.format(k)
        plot_samples(ax, features,target,predictions_knn,limits,x,y, title)
        axis_index += 1

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

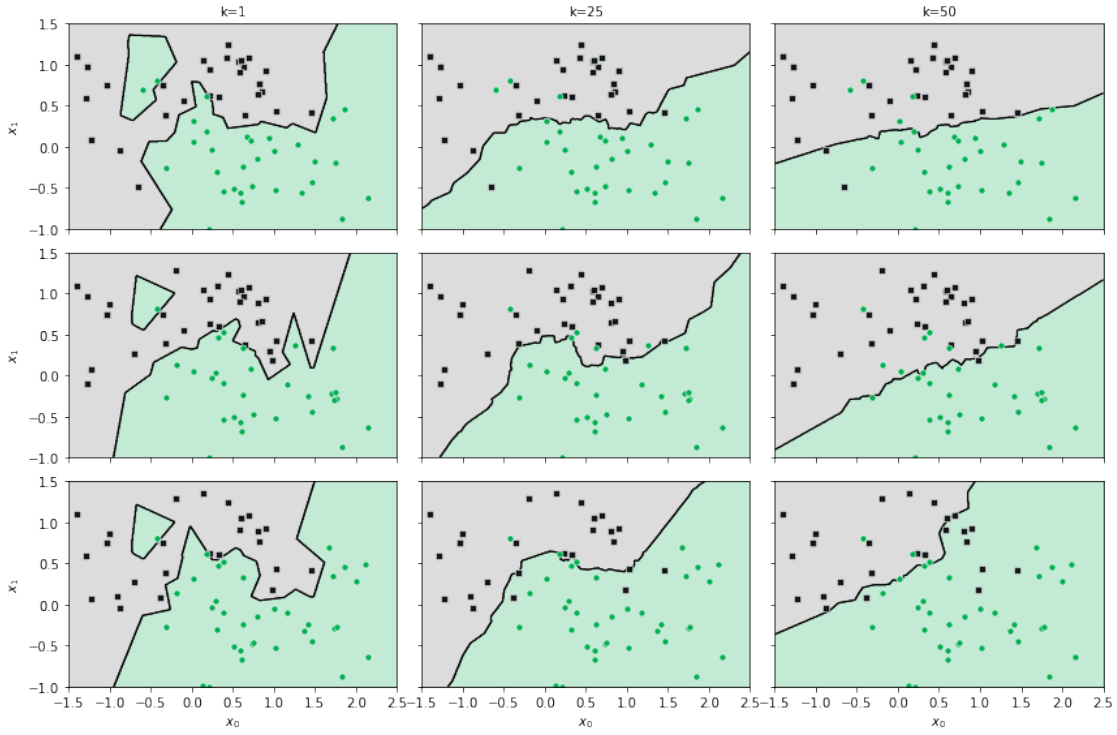

Figure 3: "Prettier" version of Figure 2. Scatterplots of training data and KNN decision boundaries for question 5 (c) and (d). Each column represents a value of k = 1, 25, or 50 (left to right) and the rows each represent a different sample of the training data. The two classes are presented in black (Class 0) and green (Class 1)

(e) The rows demonstrate the difference in flexibility between the models. On the left, the highly flexible models overfit to the data, so the decision boundaries vary wildly across the rows (the different datasets). On the right, the less flexible models see less variation in the decision boundaries across the rows. The value here of $k = 25$ appears to do the best at separating the two classes that may generalize well.

(f) The bias-variance tradeoff reflects the challenge of choosing how much flexibility the model should have before it's fit to the training data. Allowing too much results in overfitting (left column), the model begins to become less sensitive to changes in the samples as the flexibility decreases, but then begins to underfit the data as the value of k gets too high, which can be seen in the consistently high predictions errors of the right columns. The ideal is to carefully choose a level of flexibility that is not too high, but also is not too low so as to underfit the data.

Note: there are not many data points in this exercise, and we're only looking at the performance on the training (in-sample) data not the test (out-of-sample) data. The test data is where we legitimately evaluate generalization performance.

3.4 6

[20 points] Bias-variance trade-off II: Quantifying the tradeoff. This exercise will explore the impact of the bias-variance tradeoff on classifier performance by looking at classifier decision boundaries.

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), 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 $k = 1, 2, ...500$. Apply each of these trained classifiers to both your training dataset and your test dataset and plot the classification error (fraction of mislabeled datapoints).

(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

(a) Create a test dataset

[13]:
```
# Generate a training dataset of 1000 samples
noise = 0.35
nsamples = 1000
X_test, y_test = make_moons(n_samples=nsamples, noise=noise)
```

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

[14]:
```
def error(predictions,truth):
    ncorrect = sum(predictions == truth)
    nvalues  = len(truth)
    return 1 - ncorrect / nvalues

# Set the parameters of the experiment
maxk = 500
kvalues = np.round(np.logspace(0,np.log10(maxk),50)).astype(int)

error_training      = []
error_testing       = []

for k in kvalues:
    # Train the classifier
    clf_knn = neighbors.KNeighborsClassifier(n_neighbors=k)
    clf_knn.fit(X_train,y_train)

    predictions_training = clf_knn.predict(X_train)
    predictions_testing  = clf_knn.predict(X_test)

    # Calculate training error
    error_training.append(error(predictions_training,y_train))

    # Calculate test error
    error_testing.append(error(predictions_testing,y_test))
```

```
# Plot the figure showing the training and testing error
plt.figure(figsize=(7,5))
plt.semilogx(kvalues,error_training,color=color0,label='Training (in-sample)')
plt.semilogx(kvalues,error_testing,color=color1,label='Test (out-of-sample)')
plt.legend()
plt.grid(True)
plt.xlabel('k nearest neighbors')
plt.ylabel('Binary Classification Error Rate')
plt.axis([1,maxk,0,0.5])
plt.tight_layout()
plt.show()
```

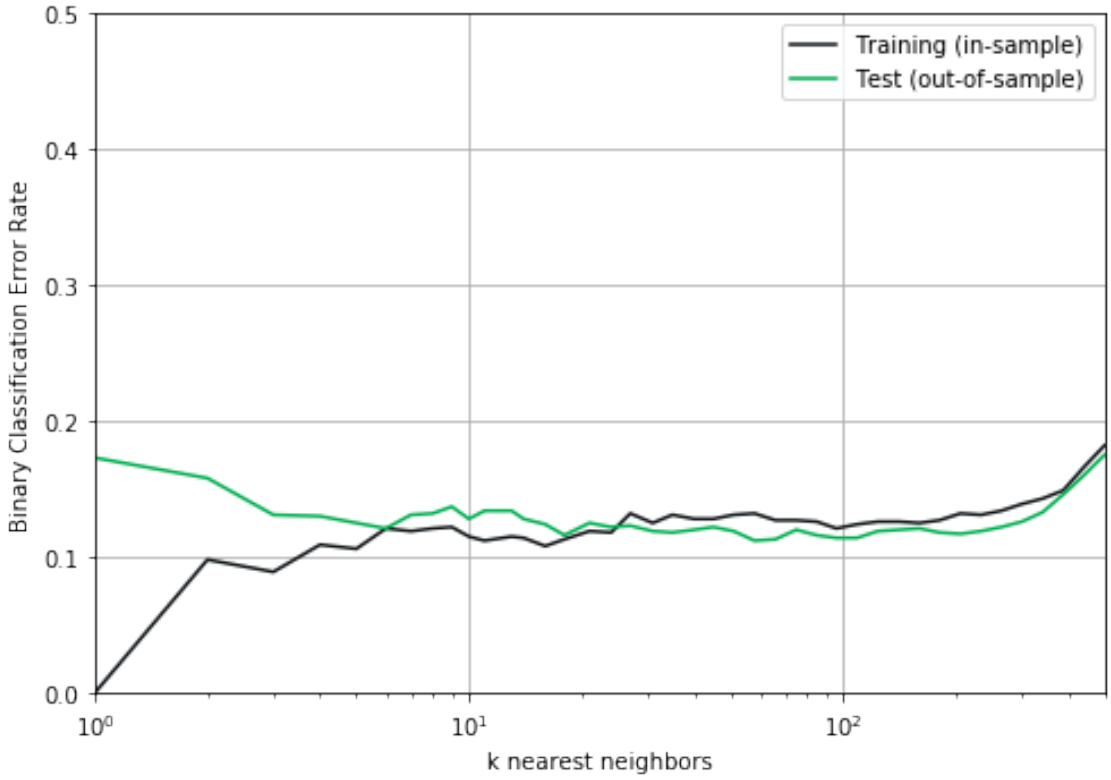

Figure 4: Training and testing error curves for Question 6(b), showing how training and test error varies with changes in the choice of k for the KNN classifier

(c) The training error starts with zero error at $k = 0$ and continues to rise as k is inreased. This makes sense, since this effectively increases the bias of the model. On the other hand, the test error starts high, gradually decreasing until around $k = 90$, then begins increasing again. This indicates there is an optimal point that results in a model with the most generalizable performance of the options tested here.

Note: it's possible that in your dataset, due to random chance, that the curves look a bit different from this. That's that nature of randomly simulated data. However, you should be able to explain the meaning

of the trends that you see accurately, and include a proper interpretation of the relationship to bias and variance

(d) Lower values of k represent high variance, higher values of k indicate higher bias.

(e) The optimum value of k is that for which k is lowest. In the example above, this would be about $k = 90$.

(f) In othe models, the number of parameters that can be adjusted during the training process represents the flexibility of the model. The more options there are to adjust, the more complexity can be represented with the model. Linear models can have few or many parameters, depending on how they are designed. Neural networks may have hundreds, thousands, or even millions of paramters, and thus require a significant amount of data to trian without overfitting. While each of these models can be alternatively defined in a way that limits the number of adjustable parameters.

3.5 7

[20 points] Linear regression and nonlinear transformations. You're given training and testing 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.

(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 $y = a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_n x_n$. Here x_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 $x_1 = x$ and $x_2 = log(x)$, 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) Using both of the models you created here in (b) and (c), plot the original data (as a scatter plot) and the two curves representing your models (each as a separate line).

(e) Using the models above, 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? Why?

[15]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

path = '../ids705/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

(a) Scatter plot of the training data:

[16]:
```
plt.figure(figsize=(7,5))
plt.plot(x_train,y_train,'k.')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.show()
```

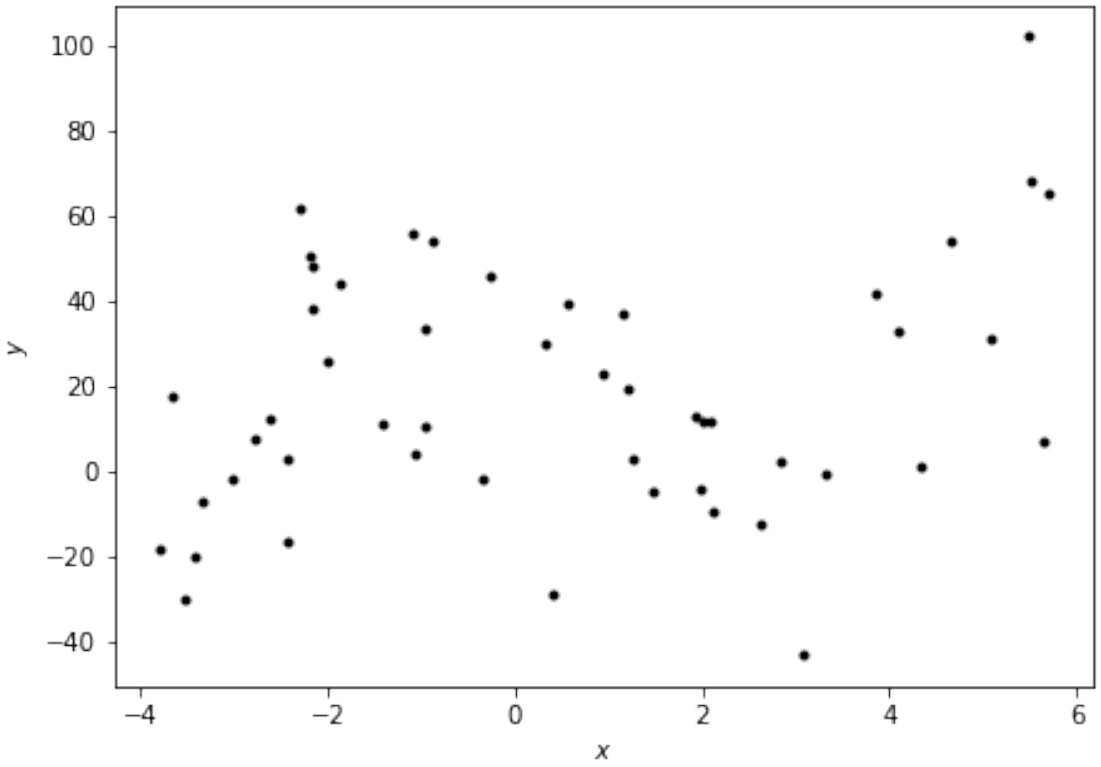

Figure 5: Scatterplot of the training data for question 7(a)

(b) Estimate a linear regression model and provide the R^2, mean square error, and equation of the trained model.

[17]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Convert the data into numpy format and reshape the data into the shape that
#  LinearRegression expects
```

```
x_train = np.array(x_train).reshape((-1,1))
y_train = np.array(y_train).reshape((-1,1))
x_test = np.array(x_test).reshape((-1,1))
y_test = np.array(y_test).reshape((-1,1))


linreg = LinearRegression(fit_intercept=True)
linreg.fit(x_train,y_train)

print('Model equation: y = {:0.4f} + {:0.4f} * x'.format(linreg.
 ↪intercept_[0],linreg.coef_[0][0]))
rsquare = linreg.score(x_train,y_train)
y_pred  = linreg.predict(x_train)
mse     = mean_squared_error(y_train,y_pred)
print('R-square for training data: {:0.3f}'.format(rsquare))
print('MSE for training data:      {:0.1f}'.format(mse))
```

```
Model equation: y = 17.2049 + 2.5907 * x
R-square for training data: 0.065
MSE for training data:      791.4
```

[18]:
```
np.sum((y_train - y_pred)**2) / len(y_train)
```

[18]:
```
791.4167471701105
```

(c) Design a multiple regression model the transforms the predictors into a form that may increase the flexibility beyond a purely linear relationship in the predictor. Here, we try the model: $y = a_0 + a_1 x + a_2 x^2 + a^3 x^3$

[19]:
```
# Generate the new features to add to the regression:
# Function to transform the features:
def gen_new_features(x):
    x2 = x**2
    x3 = x**3
    return np.concatenate((x,x2,x3),axis=1)

x_train_transformed = gen_new_features(x_train)
linreg_transformed = LinearRegression(fit_intercept=True)
linreg_transformed.fit(x_train_transformed,y_train)

print('Model equation: y = {:0.4f} + {:0.4f} * x + {:0.4f} * x^2 + {:0.4f} *␣
 ↪x^3'.format(
    linreg_transformed.intercept_[0],linreg_transformed.coef_[0][0],
    linreg_transformed.coef_[0][1],  linreg_transformed.coef_[0][2]))
rsquare = linreg_transformed.score(x_train_transformed,y_train)
y_pred  = linreg_transformed.predict(x_train_transformed)
mse     = mean_squared_error(y_train,y_pred)
print('R-square for training data: {:0.3f}'.format(rsquare))
print('MSE for training data:      {:0.1f}'.format(mse))
```

```
Model equation: y = 24.1554 + -9.2519 * x + -2.1257 * x^2 + 0.8970 * x^3
R-square for training data: 0.396
MSE for training data:      510.9
```

(d) Plot the original data (as a scatter plot) and the two curves representing your models:

[20]:
```
# To plot the curve, we need x values over the range of the data
x_min = x_train.min()
x_max = x_train.max()
x     = np.linspace(x_min,x_max,250)[np.newaxis].T

# Predict labels for the test data from the original regression
y_plot_pred = linreg.predict(x)

# Predict labels for the test data from the transformed features in multiple␣
 ↪regression
x_plot_transformed      = gen_new_features(x)
y_plot_transformed_pred = linreg_transformed.predict(x_plot_transformed)

plt.figure(figsize=(7,5))
plt.plot(x_train,y_train,'k.')
plt.plot(x,y_plot_pred,            label='$y = a_0 + a_1 x$')
plt.plot(x,y_plot_transformed_pred,label='$y = a_0 + a_1 x + a_2 x^2 + a^3␣
 ↪x^3$')
plt.legend()
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.show()
```

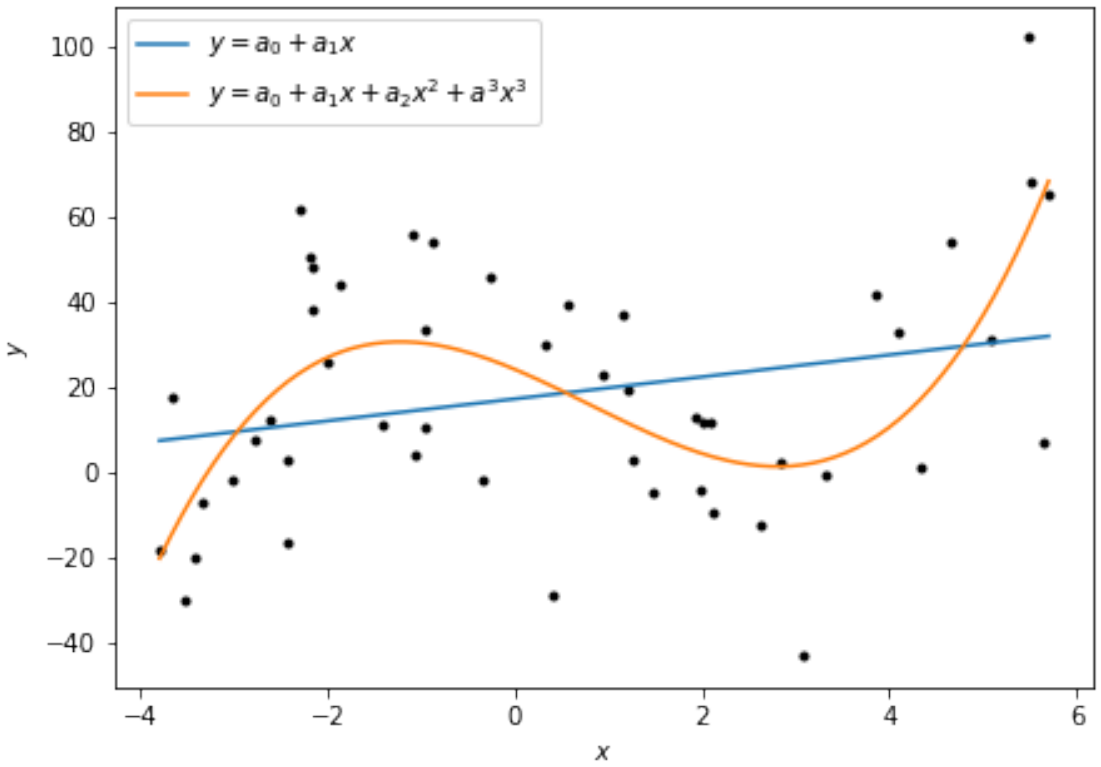

Figure 6: Comparison of model fit for question 7(d) as compared to the training data

(e) Using the models created above, apply them to the test data and estimate the R^2 and mean square error of the test dataset:

[21]:
```
x_test_transformed = gen_new_features(x_test)
rsquare       = linreg.score(x_test,y_test)
rsquare_trans = linreg_transformed.score(x_test_transformed,y_test)
y_pred        = linreg.predict(x_test)
y_pred_trans  = linreg_transformed.predict(x_test_transformed)
mse           = mean_squared_error(y_test,y_pred)
mse_trans     = mean_squared_error(y_test,y_pred_trans)
print('R-square for test data:                {:0.3f}'.format(rsquare))
print('R-square for test data (transformed): {:0.3f}'.format(rsquare_trans))
print('MSE for test data:                     {:0.1f}'.format(mse))
print('MSE for test data (transformed):       {:0.1f}'.format(mse_trans))
```

```
R-square for test data:                -0.133
R-square for test data (transformed): 0.229
MSE for test data:                     1116.7
MSE for test data (transformed):       759.5
```

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

The transformed model (third order polynomial) functioned better than the simple linear regression approach because there was a nonlinear relationship between the target/response and

feature/predictor variables. This relationship appeared visibly third order, and the coefficients a_0, a_1, a_2, and a_3 were large contributors to the model. In both the training and test data, the transformed features performed better than the single feature in both R^2 and MSE, where the test performance increased the most. However, the test dataset is small, so we should exercise caution in overstating the generalizability of the model.

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

It's important that the training data be representative of the model. Without representative examples, the predictions are likely to be inaccurate. Therefore, if the test data were significantly different from the training dataset, we would expect the predictive capability of the model to be lower.

Now, one challenge here is what kind of difference is present. Perhaps there were more samples for lower x values in the training dataset than in the test dataset. Another challenge could be that the data were fundamentally different. Perhaps x values of 1 in the training data correspond to y values of 0.5 in the training data, but of something significantly higher, for example 37, in the test data. In this case, the model will learn from the examples seen. In either of these two scenarios, the model will struggle to make predictions on the test data that is not representative of the training data.