Assignment_2

February 14, 2021

1 Assignment 2 - Machine Learning Basics

1.1 *Maobin Guo*

Netid: *mg471*

Instructions for all assignments can be found here, which is also linked to from the course syllabus.

1.2 Learning Objectives:

By successfully completing this assignment you will be able 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` - 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 - Explain the bias-variance tradeoff and the impact of model flexibility on algorithm performance - Construct simple feature transformations for improving model fit in linear models - Fit a `scikit-learn` supervised learning technique to data and make predictions using it

[268]:
```
# 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

ANSWER 1 In this situation, the flexible model will be better because the sample size is large and the predictor is small. It means the model is not easy to overfit.

ANSWER 2 The flexible model will be worse than the inflexible model because the sample size is small and predictors p is large. In this situation, a complex model is easy to overfit.

ANSWER 3 The flexible model will be better since it is suitable to fit a non-linear relationship.

ANSWER 4 The flexible model is worse than inflexible because it will learn too much noise in this situation.

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) This is a regression problem, the response variable is profit. - (ii) We are interested in inference. - (iii) n = 500 (500 firms), p = 3 (number of employees, industry, CEO salary)

(b) - (i) This is a classification problem, the response variable is whether it will be a success or a failure. - (ii) We are interested in prediction. - (iii) n = 20 (20 similar products), p = 13 (price charged for the product, marketing budget, competition price, and ten other variables)

(c) - (i) This is a regression problem, the response variable is percentage change in the US dollar. - (ii) We are interested in prediction. - (iii) n = 52 (52 weeks), p = 3 (percentagle change in the US, British and 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
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. *Note: coding is only required for part (a), for (b)-(d) please provide your reasoning.*

(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?

[269]:
```python
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)

[270]:
```python
from sklearn.metrics.pairwise import euclidean_distances

test =  [[0, 0, 0]]

euc_dist = euclidean_distances(X, test)
print(euc_dist)
```

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

Table 2. Euclidean Distances from test point $x_1 = x_2 = x_3 = 0$ to each observation

Obs.	Euclidean Distances
1	3.00
2	2.00
3	3.16
4	2.24
5	1.41
6	1.73

(b)

K=1 means we need only check the nearest point, which is point No.5. Hence, the test point's label should be blue when K=1.

(c)

Then K=3, the test point's label should be red because two of its three nearest points are red.

(d)

If the Bayes decision boundary in this problem is highly nonlinear, a small K is better. Because KNN with a small K is flexible and a flexible model is more suitable for nonlinear problems.

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 KNN class

[271]:
```
import time
import pandas as pd
import scipy.stats as ss

from sklearn.metrics.pairwise import euclidean_distances

# (a) Write your own kNN classifier
class Knn:
    # k-Nearest Neighbor class object for classification training and testing
    def __init__(self):
        self.x = []
        self.y = []

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

    def predict(self, x, k):
        # Calculate the distance from each vector in x to the training data
        dist_matrix = euclidean_distances(self.x, x, squared=True)

        top_points = np.apply_along_axis(lambda z: list(map(lambda p: 1 if␣
 ↪p[1]<k else 0,
                                                         enumerate(ss.
 ↪rankdata(z) - 1))),
                                           1, dist_matrix)

        y_hat = [1 if x else 0 for x in (np.dot(top_points, self.y) > k/2)]

        # 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
```

(b) Load data

```
[272]: X_train_high = pd.read_csv("./data/A2_X_train_high.csv", header = None)
X_train_low = pd.read_csv("./data/A2_X_train_low.csv", header = None)
X_test_high = pd.read_csv("./data/A2_X_test_high.csv", header = None)
X_test_low = pd.read_csv("./data/A2_X_test_low.csv", header = None)

y_train_high = pd.read_csv("./data/A2_y_train_high.csv", header = None)
y_train_low = pd.read_csv("./data/A2_y_train_low.csv", header = None)
y_test_high = pd.read_csv("./data/A2_y_test_high.csv", header = None)
y_test_low = pd.read_csv("./data/A2_y_test_low.csv", header = None)
```

(c)

```
[273]: K = 5
knn = Knn()

### Low dimension
time_begin = time.time()
knn.fit(X_train_low, y_train_low)
y_hat     = knn.predict(X_test_low, K)
accu = accuracy(y_test_low.iloc[:,0], y_hat)
duration = time.time() - time_begin

print("[Low dimension] Accuracy [{:0.4f}] \t Time [{:0.4f}] Sec".format(accu,␣
 ↪duration))

### High dimension
time_begin = time.time()
knn.fit(X_train_high, y_train_high)
y_hat     = knn.predict(X_test_high, K)
accu = accuracy(y_test_high.iloc[:,0], y_hat)
duration = time.time() - time_begin

print("[High dimension] Accuracy [{:0.4f}] \t Time [{:0.4f} Sec]".format(accu,␣
 ↪duration))
```

```
[Low dimension] Accuracy [0.9170]        Time [0.6034] Sec
[High dimension] Accuracy [0.9900]       Time [0.5964 Sec]
```

(d)

```
[274]: from sklearn.neighbors import KNeighborsClassifier

nkc = KNeighborsClassifier(n_neighbors = 5)

# Low
time_begin = time.time()
nkc.fit(X_train_low.values, y_train_low.values.T[0])
y_hat     = nkc.predict(X_test_low.values)
```

```
accu = accuracy(y_test_low.iloc[:,0], y_hat)
duration = time.time() - time_begin
print("[Low dimension] Accuracy [{:0.4f}] \t Time [{:0.4f} Sec]".format(accu,␣
↪duration))

# High
time_begin = time.time()
nkc.fit(X_train_high.values, y_train_low.values.T[0])
y_hat    = nkc.predict(X_test_high.values)
accu = accuracy(y_test_high.iloc[:,0], y_hat)
duration = time.time() - time_begin
print("[High dimension] Accuracy [{:0.4f}] \t Time [{:0.4f} Sec]".format(accu,␣
↪duration))
```

```
[Low dimension] Accuracy [0.9250]        Time [0.0359 Sec]
[High dimension] Accuracy [0.9930]       Time [0.2334 Sec]
```

There is a minor difference about accuray between my result and KNeighborsClassifier's result. Also, KNeighborsClassifier costs far less time than my method.

(e)

The slow prediction will bring problems for some situations where there require real-time decisions. Typically, prediction speed will limit the applied range of an algorithm. For example, fraudulent transaction detection often requires fast decisions because a slow prediction will harm the user experience.

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

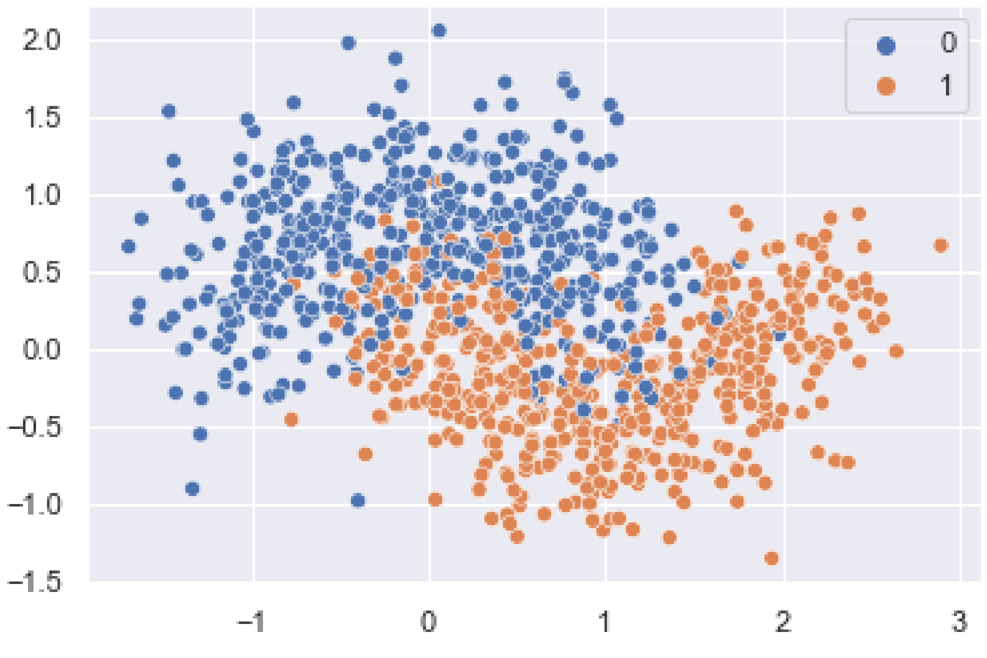
(a)

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

       # Make sure reproducible
       np.random.seed(213)

       X_train, y_train = make_moons(n_samples=1000, noise=0.35)
```

(b)

```
[276]: import matplotlib.pyplot as plt
       import seaborn as sns
       sns.set_theme(style="darkgrid")


       sns.scatterplot(data=X_train, x=X_train[:,0], y=X_train[:,1], hue=y_train,␣
        ↪palette="deep")
```

```
[276]: <AxesSubplot:>
```

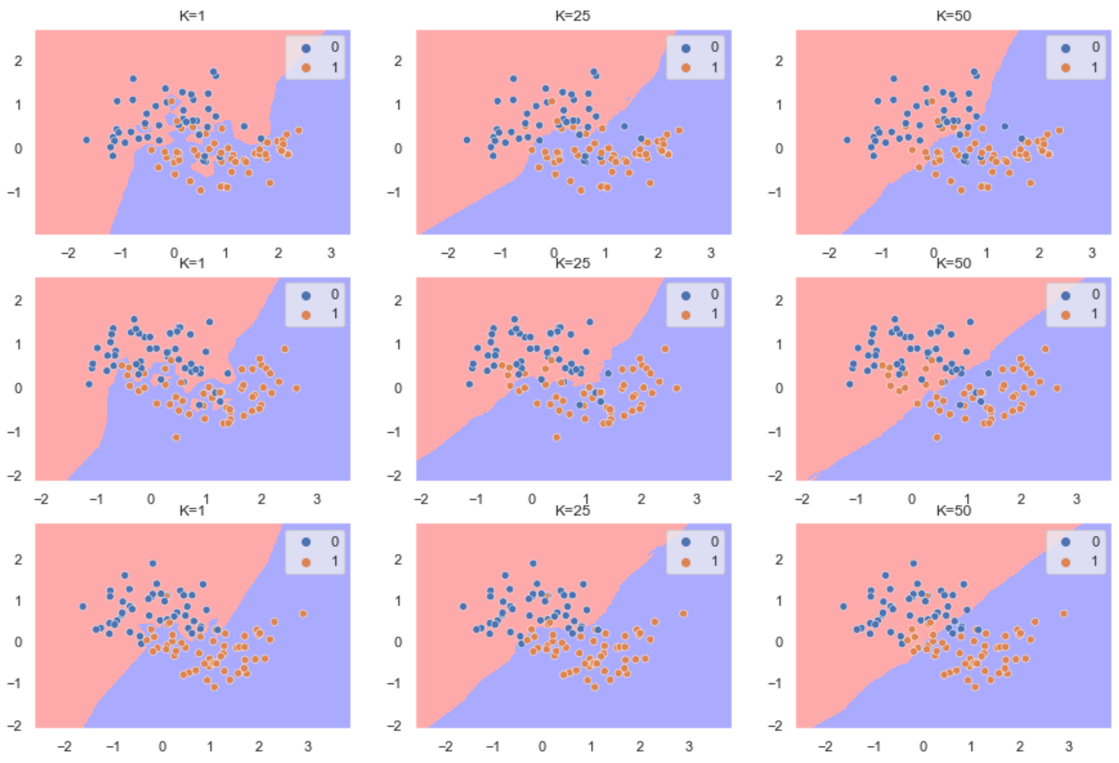


(c)

[277]:
```python
from sklearn.neighbors import KNeighborsClassifier

K_value = [1, 25, 50]
Sample_size = 100

# Set random seed
random.seed(123)

# get random index
sample_index_arr = [random.sample(range(0, 999), 100) for i in range(0, 3)]

# generate KNN model
knn_models_arr = [KNeighborsClassifier(n_neighbors = k) for k in K_value]
```

(d)

[278]:
```python
from matplotlib.colors import ListedColormap

step_size = .02

col_index = -1
row_index = -1

fig = plt.figure(figsize=(15,10))

for sample_index in sample_index_arr:
    col_index += 1
    row_index = -1

    for model in knn_models_arr:
        row_index += 1

        X = X_train[sample_index, :]
        y = y_train[sample_index]

        model.fit(X, y)

        # Plot the decision boundary.
        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, step_size),
                             np.arange(y_min, y_max, step_size))
        Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

        # Put the result into a color plot
        Z = Z.reshape(xx.shape)

        plt.subplot(3, 3, (3*col_index) + row_index + 1)
        plt.pcolormesh(xx, yy, Z, cmap=cmap_light, shading='auto')
        # Plot also the training points
        #plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold)
        sns.scatterplot(data=X, x=X[:, 0], y=X[:, 1], hue=y, palette="deep")
        plt.xlim(xx.min(), xx.max())
        plt.ylim(yy.min(), yy.max())
        plt.title("K={}".format(K_value[row_index]))
```



(e)

The decision boundaries become more and more smooth from the right column to the left column. At the same time, there is no apparent difference between each row. The right image boundary appears to best separate the two data classes because it looks smooth, which means there is no overfitting problem. The decision boundaries in the left columns (K=1) vary the most.

(f)

The bias-variance tradeoff is the property of a model. The variance of the parameter estimates across samples can be reduced by increasing the bias in the estimated parameters. Take this task as an example; when K=1, the decision boundary is very rugged and can fig out almost every point, which means the model's bias error is low. However, it also means the model is overfitting, and it will suffer significant variance. For kNN, the best K value would neither make it overfit nor underfit.

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)

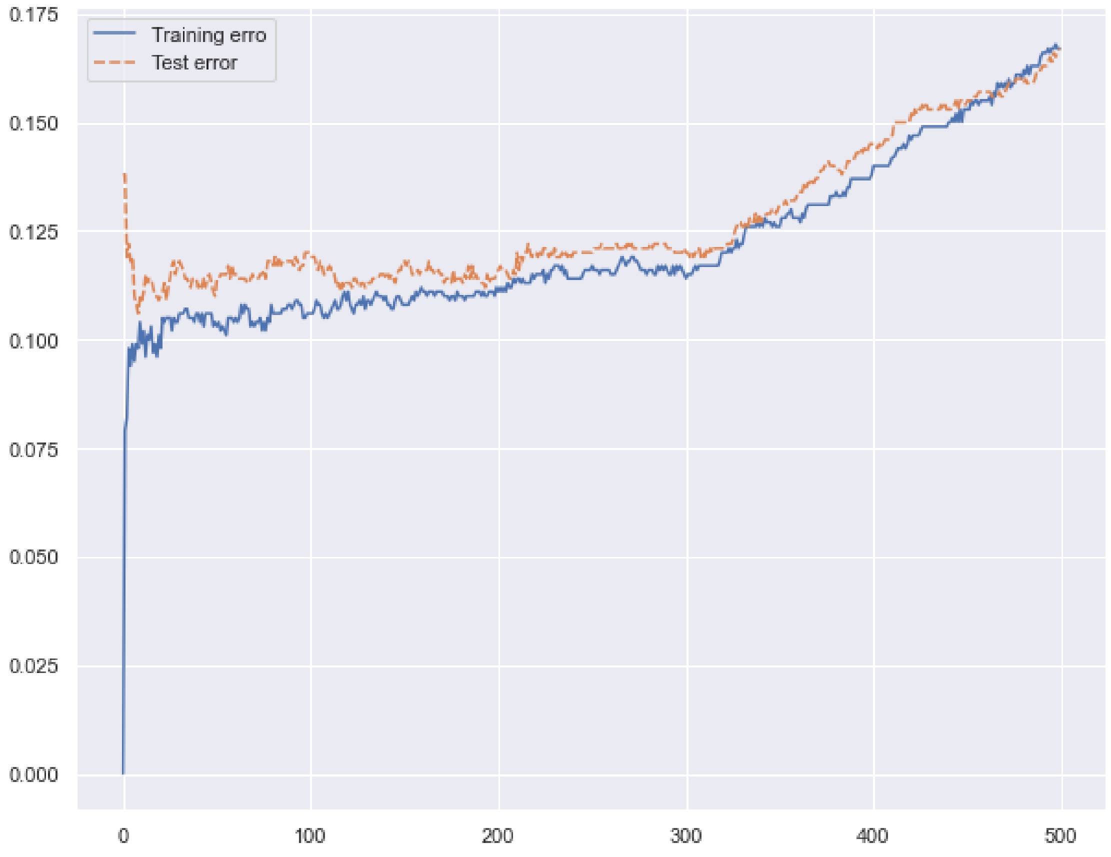
[279]:
```python
import numpy as np
import random
from sklearn.datasets import make_moons

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

(b)

[282]:
```python
K_values = range(1, 501)
train_err = []
test_err =  []

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

for k in K_values:

    # Train the classifier
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train,y_train)

    y_hat_train = knn.predict(X_train)
    y_hat_test  = knn.predict(X_test)

    train_err.append(1-accuracy(y_train, y_hat_train))
    test_err.append(1-accuracy(y_test, y_hat_test))
```

```
[283]: plt.figure(figsize=(10,8))
       sns.lineplot(data=[train_err, test_err])
       plt.legend(labels=['Training erro', 'Test error'])
```

```
[283]: <matplotlib.legend.Legend at 0x2434ddbf780>
```



(c)

In the beginning, as the value of K gradually increased from 1, training error jump up from 0 to 0.1 while test error decrease from 0.13 to 0.1. The test error hit its lowest rate when the K value is about 10. After that, test error oscillates in a narrow range, while training error rises gradually. The two kinds of error upsurge when K bigger than

300 and hit their highest value at K =500.

(d)

Small K value shows high variance, while high K value shows high bias.

(e)

The test error stays at a low-level steady when k between 110 and 150; hence K = 110 is the best choice.

(f)

For other parameter models, their flexibility depends on parameter size. The linear regression model typically has few parameters. However, neural networks, especially deep neural networks, can have billions of parameters. The more parameter a model owns, the more training data it needs. So billions of training data are not surprising for modern deep learning models.

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. Consider this your baseline model against which you will compare other model options.

(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?

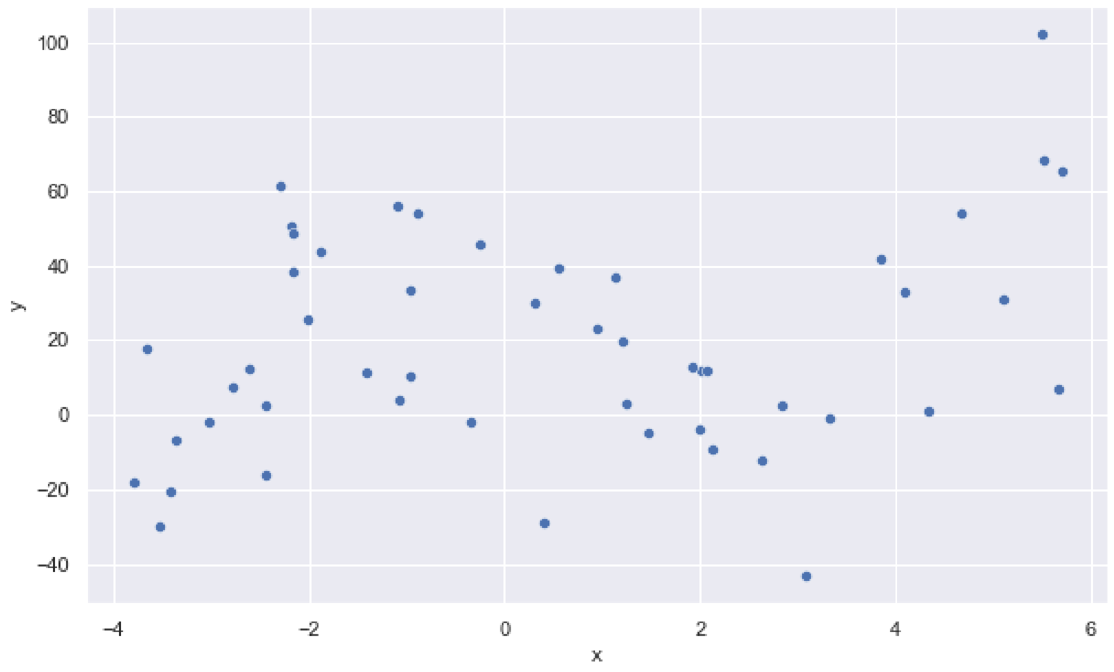
ANSWER

(a)

```
[284]: import numpy as np
       import pandas as pd
       import matplotlib.pyplot as plt
       import seaborn as sns

       train = pd.read_csv('./data/A2_Q7_train.csv')
       test = pd.read_csv('./data/A2_Q7_test.csv')

       X_train, y_train = train.x, train.y
       X_test, y_test = test.x, test.y
```

```
[285]: plt.figure(figsize=(10,6))
       sns.scatterplot(x=X_train, y=y_train)
```

```
[285]: <AxesSubplot:xlabel='x', ylabel='y'>
```



(b)

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

```
X_train_np = np.array(X_train).reshape(-1, 1)
y_train_np = np.array(y_train).reshape(-1, 1)

model = LinearRegression(fit_intercept=True)
model.fit(X_train_np, y_train_np)

print('Equation: y = {:0.2f} + {:0.2f} * x'.format(model.intercept_[0], model.
 ↪coef_[0][0]))

prediction = model.predict(X_train_np)

r2 = r2_score(y_train_np, prediction)
mse = mean_squared_error(y_train_np, prediction)

print('R-square: {:0.4f}'.format(r2))
print('MSE {:0.4f}'.format(mse))
```

```
Equation: y = 17.20 + 2.59 * x
R-square: 0.0649
MSE 791.4167
```

(c)

[287]:
```
def transform_x(x):
    return np.concatenate((x, x ** 2, x ** 3), axis=1)

X_train_enhanced = transform_x(X_train_np)

model_enhanced = LinearRegression(fit_intercept=True)
model_enhanced.fit(X_train_enhanced, y_train_np)


prediction_enhanced = model_enhanced.predict(X_train_enhanced)

r2 = r2_score(y_train_np, prediction_enhanced)
mse = mean_squared_error(y_train_np, prediction_enhanced)

print('Equation: y = {:0.2f} + {:0.2f} * x + {:0.2f} * x^2 + {:0.2f} * x^3'.\
              format(model_enhanced.intercept_[0],
                     model_enhanced.coef_[0][0],
                     model_enhanced.coef_[0][1],
                     model_enhanced.coef_[0][2]))

print('R-square: {:0.4f}'.format(r2))
print('MSE {:0.4f}'.format(mse))
```

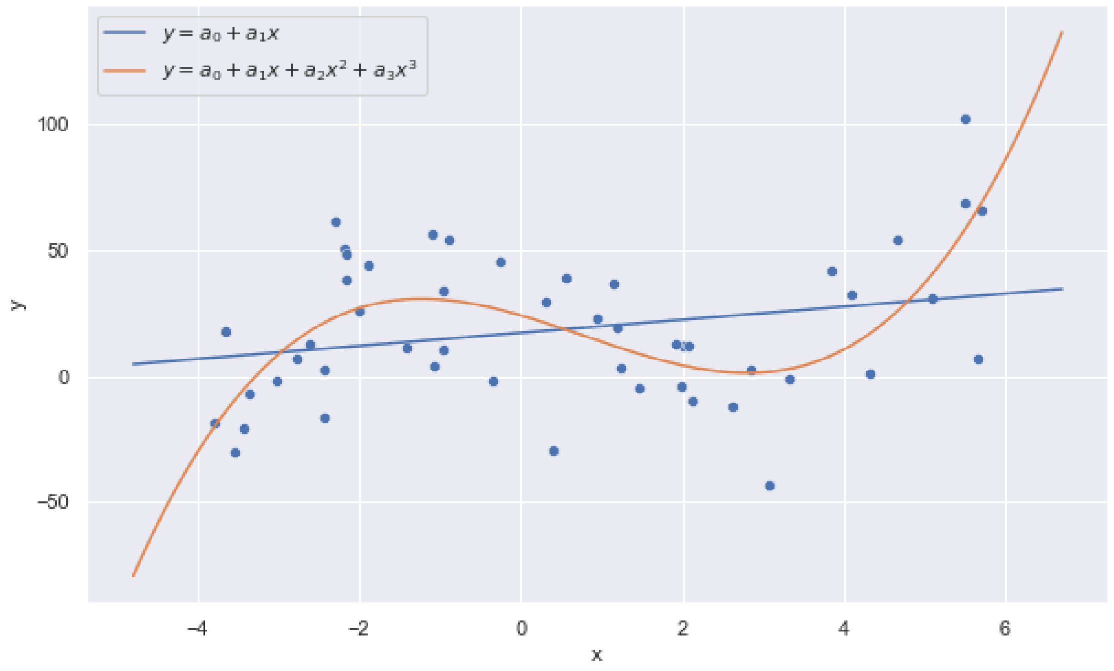
```
Equation: y = 24.16 + -9.25 * x + -2.13 * x^2 + 0.90 * x^3
```

```
R-square: 0.3963
MSE 510.8850
```

(d)

[289]:
```python
#plt.figure(figsize=(10,6))

x_min = X_train.min()
x_max = X_train.max()
x = np.linspace(x_min - 1, x_max + 1, 200).reshape(-1, 1)

fig, ax = plt.subplots(figsize=(10, 6))
y = model.predict(x)
y_enhanced = model_enhanced.predict(transform_x(x))

sns.scatterplot(x=X_train, y=y_train, ax=ax)
plt.plot(x, y, label='$y = a_0 + a_1 x$')
plt.plot(x, y_enhanced, label='$ y=a_0 + a_1x + a_2x^2 + a_3x^3 $')

plt.legend()
plt.xlabel('x')
plt.ylabel('y')
```

[289]:
```
Text(0, 0.5, 'y')
```



(e)

[290]:
```
X_test_np = np.array(X_test).reshape(-1, 1)
y_test_np = np.array(y_test).reshape(-1, 1)

x_test_enhanced = transform_x(X_test_np)

pred          = model.predict(X_test_np)
pred_enhanced = model_enhanced.predict(x_test_enhanced)

r2 = r2_score(y_test_np, pred)
mse = mean_squared_error(y_test_np, pred)

r2_enhanced = r2_score(y_test_np, pred_enhanced)
mse_enhanced = mean_squared_error(y_test_np, pred_enhanced)

print('R2 for test data: {:0.3f}'.format(r2))
print('R2 for test data (with transformed): {:0.3f}'.format(r2_enhanced))
print('MSE for test data: {:0.1f}'.format(mse))
print('MSE for test data (with transformed): {:0.1f}'.format(mse_enhanced))
```

```
R2 for test data: -0.133
R2 for test data (with transformed): 0.229
MSE for test data: 1116.7
MSE for test data (with transformed): 759.5
```

(f)

The transformed model performs better than the baseline model on both training data and test data. According to the scatter plot chart, there is an apparent non-linear relationship between x and y. The baseline model does not contain non-linear items, so it can't learn how to process this non-linear relationship. While the transformed model includes x square and x cube items; hence it can capture the relationship.

(g)

It will be a disaster for the trained model. All machine learning algorithms are based on the assumption that training data is representative of real data. The more the training data resembles the overall data, the algorithm may learn the data's rules. If test data were significantly different from the training dataset, the algorithm could not learn enough information. It is foreseeable that the model will perform very poorly when predicting unknown data.

[ ]:
```

```