Assignment 2

April 9, 2021

---

*You are currently looking at **version 1.5** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

---

1   Assignment 2

In this assignment you'll explore the relationship between model complexity and generalization performance, by adjusting key parameters of various supervised learning models. Part 1 of this assignment will look at regression and Part 2 will look at classification.

1.1   Part 1 - Regression

First, run the following block to set up the variables needed for later sections.

```
In [110]: import numpy as np
          import pandas as pd
          from sklearn.model_selection import train_test_split


          np.random.seed(0)
          n = 15
          x = np.linspace(0,10,n) + np.random.randn(n)/5
          y = np.sin(x)+x/6 + np.random.randn(n)/10

          #print(x)
          #print(y)
          X_train, X_test, y_train, y_test = train_test_split(x, y, random_state=0)

          # You can use this function to help you visualize the dataset by
          # plotting a scatterplot of the data points
          # in the training and test sets.
          #def part1_scatter():
              #import matplotlib.pyplot as plt
              #%matplotlib notebook
```

```
#plt.figure()
#plt.scatter(X_train, y_train, label='training data')
#plt.scatter(X_test, y_test, label='test data')
#plt.legend(loc=4);


# NOTE: Uncomment the function below to visualize the data, but be sure
# to **re-comment it before submitting this assignment to the autograder
#part1_scatter()
#print(x.shape)
#print(X_train.shape)
```

1.1.1 Question 1

Write a function that fits a polynomial LinearRegression model on the *training data* X_train for degrees 1, 3, 6, and 9. (Use PolynomialFeatures in sklearn.preprocessing to create the polynomial features and then fit a linear regression model) For each model, find 100 predicted values over the interval x = 0 to 10 (e.g. np.linspace(0,10,100)) and store this in a numpy array. The first row of this array should correspond to the output from the model trained on degree 1, the second row degree 3, the third row degree 6, and the fourth row degree 9.

The figure above shows the fitted models plotted on top of the original data (using plot_one()).

This function should return a numpy array with shape (4, 100)

```
In [111]: from sklearn.linear_model import LinearRegression
          from sklearn.preprocessing import PolynomialFeatures
          X_train, X_test, y_train, y_test = train_test_split(x, y,random_state = (

          np.random.seed(0)
          n = 15
          x = np.linspace(0,10,n) + np.random.randn(n)/5
          y = np.sin(x)+x/6 + np.random.randn(n)/10

          def answer_one():
              from sklearn.linear_model import LinearRegression
              from sklearn.preprocessing import PolynomialFeatures


              # Your code here
              X_train, X_test, y_train, y_test = train_test_split(x, y,random_state
              poly_1 = PolynomialFeatures(degree=1)
              poly_3 = PolynomialFeatures(degree=3)
              poly_6 = PolynomialFeatures(degree=6)
              poly_9 = PolynomialFeatures(degree=9)
              X_train = X_train.reshape(-1,1)
              x_poly_1 = poly_1.fit_transform(X_train)
              x_poly_3 = poly_3.fit_transform(X_train)
              x_poly_6 = poly_6.fit_transform(X_train)
```

```
            x_poly_9 = poly_9.fit_transform(X_train)

            x_predict = np.linspace(0,10,100).reshape(-1,1)
            x_predict_1 = poly_1.fit_transform(x_predict)
            x_predict_3 = poly_3.fit_transform(x_predict)
            x_predict_6 = poly_6.fit_transform(x_predict)
            x_predict_9 = poly_9.fit_transform(x_predict)
            #x_polys = [x_poly_1,x_poly_3,x_poly_6,x_poly_9]
            model1 = LinearRegression()
            model3 = LinearRegression()
            model6 = LinearRegression()
            model9 = LinearRegression()
            model1.fit(x_poly_1, y_train)
            model3.fit(x_poly_3, y_train)
            model6.fit(x_poly_6, y_train)
            model9.fit(x_poly_9, y_train)
            prediction1 = None
            prediction3 = None
            prediction6 = None
            prediction9 = None
            predictions = [prediction1,prediction3,prediction6,prediction9]
            degrees = [1,3,6,9]

            X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(x_poly_1,
            X_train_3, X_test_3, y_train_3, y_test_3 = train_test_split(x_poly_3,
            X_train_6, X_test_6, y_train_6, y_test_6 = train_test_split(x_poly_6,
            X_train_9, X_test_9, y_train_9, y_test_9 = train_test_split(x_poly_9,

            #x_predict_poly_1 = model1.transform(x_predict)
            #x_predict_poly_3 = model3.transform(x_predict)
            #x_predict_poly_6 = model6.transform(x_predict)
            #x_predict_poly_9 = model9.transform(x_predict)
            #linreg1 = LinearRegression().fit(X_train_1, y_train_1)
            #linreg3 = LinearRegression().fit(X_train_3, y_train_3)
            #linreg6 = LinearRegression().fit(X_train_6, y_train_6)
            #linreg9 = LinearRegression().fit(X_train_9, y_train_9)
            prediction1 = model1.predict(x_predict_1)
            prediction3 = model3.predict(x_predict_3)
            prediction6 = model6.predict(x_predict_6)
            prediction9 = model9.predict(x_predict_9)
            predictions = np.array([prediction1, prediction3, prediction6, predic
            return predictions

        #answer_one()

In [112]:  # feel free to use the function plot_one() to replicate the figure
           # from the prompt once you have completed question one
           #def plot_one(degree_predictions):
```

```
#import matplotlib.pyplot as plt
#%matplotlib notebook
#plt.figure(figsize=(10,5))
#plt.plot(X_train, y_train, 'o', label='training data', markersize=1(
#plt.plot(X_test, y_test, 'o', label='test data', markersize=10)
#for i,degree in enumerate([1,3,6,9]):
    #plt.plot(np.linspace(0,10,100), degree_predictions[i], alpha=0.8
#plt.ylim(-1,2.5)
#plt.legend(loc=4)

#plot_one(answer_one())
```

1.1.2 Question 2

Write a function that fits a polynomial LinearRegression model on the training data `X_train` for degrees 0 through 9. For each model compute the R^2 (coefficient of determination) regression score on the training data as well as the the test data, and return both of these arrays in a tuple.

This function should return one tuple of numpy arrays (r2_train, r2_test). Both arrays should have shape (10,)

```
In [113]: from sklearn.linear_model import LinearRegression
          from sklearn.preprocessing import PolynomialFeatures
          X_train, X_test, y_train, y_test = train_test_split(x, y,random_state = 0

          np.random.seed(0)
          n = 15
          x = np.linspace(0,10,n) + np.random.randn(n)/5
          y = np.sin(x)+x/6 + np.random.randn(n)/10
          def answer_two():

              from sklearn.linear_model import LinearRegression
              from sklearn.preprocessing import PolynomialFeatures
              from sklearn.metrics.regression import r2_score
              X_train, X_test, y_train, y_test = train_test_split(x, y,random_state
              poly_0 = PolynomialFeatures(degree=0)
              poly_1 = PolynomialFeatures(degree=1)
              poly_2 = PolynomialFeatures(degree=2)
              poly_3 = PolynomialFeatures(degree=3)
              poly_4 = PolynomialFeatures(degree=4)
              poly_5 = PolynomialFeatures(degree=5)
              poly_6 = PolynomialFeatures(degree=6)
              poly_7 = PolynomialFeatures(degree=7)
              poly_8 = PolynomialFeatures(degree=8)
              poly_9 = PolynomialFeatures(degree=9)
              X_train = X_train.reshape(-1,1)
              x_poly_0 = poly_0.fit_transform(X_train)
              x_poly_1 = poly_1.fit_transform(X_train)
              x_poly_2 = poly_2.fit_transform(X_train)
```

4

```python
x_poly_3 = poly_3.fit_transform(X_train)
x_poly_4 = poly_4.fit_transform(X_train)
x_poly_5 = poly_5.fit_transform(X_train)
x_poly_6 = poly_6.fit_transform(X_train)
x_poly_7 = poly_7.fit_transform(X_train)
x_poly_8 = poly_8.fit_transform(X_train)
x_poly_9 = poly_9.fit_transform(X_train)

X_test = X_test.reshape(-1,1)
x_test_0 = poly_0.fit_transform(X_test)
x_test_1 = poly_1.fit_transform(X_test)
x_test_2 = poly_2.fit_transform(X_test)
x_test_3 = poly_3.fit_transform(X_test)
x_test_4 = poly_4.fit_transform(X_test)
x_test_5 = poly_5.fit_transform(X_test)
x_test_6 = poly_6.fit_transform(X_test)
x_test_7 = poly_7.fit_transform(X_test)
x_test_8 = poly_8.fit_transform(X_test)
x_test_9 = poly_9.fit_transform(X_test)

model0 = LinearRegression()
model1 = LinearRegression()
model2 = LinearRegression()
model3 = LinearRegression()
model4 = LinearRegression()
model5 = LinearRegression()
model6 = LinearRegression()
model7 = LinearRegression()
model8 = LinearRegression()
model9 = LinearRegression()

degrees = [0,1,2,3,4,5,6,7,8,9]


model0.fit(x_poly_0, y_train)
model1.fit(x_poly_1, y_train)
model2.fit(x_poly_2, y_train)
model3.fit(x_poly_3, y_train)
model4.fit(x_poly_4, y_train)
model5.fit(x_poly_5, y_train)
model6.fit(x_poly_6, y_train)
model7.fit(x_poly_7, y_train)
model8.fit(x_poly_8, y_train)
model9.fit(x_poly_9, y_train)

#print(X_train_0.shape)
#print(y_train_0.shape)
```

```python
prediction_train0 = model0.predict(x_poly_0)
prediction_train1 = model1.predict(x_poly_1)
prediction_train2 = model2.predict(x_poly_2)
prediction_train3 = model3.predict(x_poly_3)
prediction_train4 = model4.predict(x_poly_4)
prediction_train5 = model5.predict(x_poly_5)
prediction_train6 = model6.predict(x_poly_6)
prediction_train7 = model7.predict(x_poly_7)
prediction_train8 = model8.predict(x_poly_8)
prediction_train9 = model9.predict(x_poly_9)




prediction_test0 = model0.predict(x_test_0)
prediction_test1 = model1.predict(x_test_1)
prediction_test2 = model2.predict(x_test_2)
prediction_test3 = model3.predict(x_test_3)
prediction_test4 = model4.predict(x_test_4)
prediction_test5 = model5.predict(x_test_5)
prediction_test6 = model6.predict(x_test_6)
prediction_test7 = model7.predict(x_test_7)
prediction_test8 = model8.predict(x_test_8)
prediction_test9 = model9.predict(x_test_9)
#print(X_train_0, prediction_train0)

r2_train_0 = r2_score(y_train, prediction_train0)
r2_train_1 = r2_score(y_train, prediction_train1)
r2_train_2 = r2_score(y_train, prediction_train2)
r2_train_3 = r2_score(y_train, prediction_train3)
r2_train_4 = r2_score(y_train, prediction_train4)
r2_train_5 = r2_score(y_train, prediction_train5)
r2_train_6 = r2_score(y_train, prediction_train6)
r2_train_7 = r2_score(y_train, prediction_train7)
r2_train_8 = r2_score(y_train, prediction_train8)
r2_train_9 = r2_score(y_train, prediction_train9)
#print(X_test_0, prediction_test0)

r2_test_0 = r2_score(y_test, prediction_test0)
r2_test_1 = r2_score(y_test, prediction_test1)
r2_test_2 = r2_score(y_test, prediction_test2)
r2_test_3 = r2_score(y_test, prediction_test3)
r2_test_4 = r2_score(y_test, prediction_test4)
r2_test_5 = r2_score(y_test, prediction_test5)
r2_test_6 = r2_score(y_test, prediction_test6)
r2_test_7 = r2_score(y_test, prediction_test7)
r2_test_8 = r2_score(y_test, prediction_test8)
r2_test_9 = r2_score(y_test, prediction_test9)
```

```
            #r2_train = [r2_train_0,r2_train_1,r2_train_2,r2_train_3,r2_train_4,
                    #r2_train_7,r2_train_8,r2_train_9]
        #r2_test = [r2_test_0,r2_test_1,r2_test_2,r2_test_3,r2_test_4,r2_test
        r2_train = np.array([r2_train_0, r2_train_1, r2_train_2, r2_train_3,
                        r2_train_6, r2_train_7, r2_train_8, r2_train_9])
        r2_test = np.array([r2_test_0, r2_test_1, r2_test_2, r2_test_3, r2_te
                        r2_test_6, r2_test_7, r2_test_8, r2_test_9])
        return (r2_train, r2_test)


    #answer_two()

    #def plot(degree_predictions):
        #import matplotlib.pyplot as plt
        #%matplotlib notebook
        #plt.figure(figsize=(10,5))
        #degrees = np.arange(0,10,1)
        #plt.plot(degrees, r2_train, '-o',label='training data', markersize=
        #plt.plot(degrees, r2_test, '-o', label='test data', markersize=10)
        #degrees = np.arange(0,11,1)
        #for i,degree in enumerate([0,1,2,3,4,5,6,7,8,9]):
            #plt.plot(np.arange(0,11,1), degree_predictions[i], alpha=0.8, lw
        #plt.xlim(0,10)
        #plt.ylim(-0.9,1.2)
        #plt.legend(loc=4)

    #plot(answer_two())
    #print(answer_two())
```

1.1.3 Question 3

Based on the R^2 scores from question 2 (degree levels 0 through 9), what degree level corresponds to a model that is underfitting? What degree level corresponds to a model that is overfitting? What choice of degree level would provide a model with good generalization performance on this dataset?

Hint: Try plotting the R^2 scores from question 2 to visualize the relationship between degree level and R^2. Remember to comment out the import matplotlib line before submission.

This function should return one tuple with the degree values in this order: (Underfitting, Overfitting, Good_Generalization). There might be multiple correct solutions, however, you only need to return one possible solution, for example, (1,2,3).

```
In [114]: def answer_three():

            # Your code here
            degree_range = [0,1,2,3,4,5,6,7,8,9]
            r2_train, r2_test = answer_two()

            r2_train_sorted = np.sort(r2_train)
```

7

```python
        r2_test_sorted = np.sort(r2_test)

        # Initialize the values
        Underfitting = 0
        Overfitting = 0
        Good_Generalization = 0
        min_r2_train = np.min(r2_train)
        max_r2_train = np.max(r2_train)
        mean_r2_train = np.mean(r2_train)
        min_r2_test = np.min(r2_test)
        max_r2_test = np.max(r2_test)
        mean_r2_test = np.mean(r2_test)

        for degree, data in zip(degree_range, zip(r2_train, r2_test)):
            if data[0] == min_r2_train and data[1] == min_r2_test:
                Underfitting = degree
            if data[0] > r2_train_sorted[1] and data[1] <= r2_test_sorted[1]:
                Overfitting = degree
            if data[0] < max_r2_train and data[1] == max_r2_test:
                Good_Generalization = degree

        return Underfitting, Overfitting, Good_Generalization


#return #(0,9,7) # Return your answer
#r2_train, r2_test = answer_two()

# Sort the scores
#r2_train_sorted = np.sort(r2_train)
#r2_test_sorted = np.sort(r2_test)

# Initialize the values
#Underfitting = 0
#Overfitting = 0
#Good_Generalization = 0
#min_r2_train = np.min(r2_train)
#max_r2_train = np.max(r2_train)
#mean_r2_train = np.mean(r2_train)
#min_r2_test = np.min(r2_test)
#max_r2_test = np.max(r2_test)
#mean_r2_test = np.mean(r2_test)
#print(min_r2_train, min_r2_test)
#print(max_r2_train, max_r2_test)
#print(mean_r2_train, mean_r2_test)

#answer_three()
#answer_two()
```

1.1.4 Question 4

Training models on high degree polynomial features can result in overly complex models that overfit, so we often use regularized versions of the model to constrain model complexity, as we saw with Ridge and Lasso linear regression.

For this question, train two models: a non-regularized LinearRegression model (default parameters) and a regularized Lasso Regression model (with parameters `alpha=0.01`, `max_iter=10000`) both on polynomial features of degree 12. Return the R^2 score for both the LinearRegression and Lasso model's test sets.

This function should return one tuple `(LinearRegression_R2_test_score, Lasso_R2_test_score)`

```
In [115]: def answer_four():
              from sklearn.preprocessing import PolynomialFeatures
              from sklearn.linear_model import Lasso, LinearRegression
              from sklearn.metrics.regression import r2_score
              from sklearn.preprocessing import MinMaxScaler
              scaler = MinMaxScaler()

              # Your code here
              X_train, X_test, y_train, y_test = train_test_split(x, y,random_state
              X_train = X_train.reshape(-1,1)
              X_test = X_test.reshape(-1,1)
              #X_train_scaled = scaler.fit_transform(X_train)
              #X_test_scaled = scaler.transform(X_test)
              poly = PolynomialFeatures(degree=12)
              model1 = LinearRegression()
              model2 = Lasso(alpha=0.01, max_iter = 10000)
              x_poly_train = poly.fit_transform(X_train)
              x_poly_test = poly.fit_transform(X_test)
              model1.fit(x_poly_train, y_train)
              model2.fit(x_poly_train, y_train)
              linear_prediction_test = model1.predict(x_poly_test)
              lasso_prediction_test = model2.predict(x_poly_test)
              r2_test_linear = r2_score(y_test, linear_prediction_test)
              r2_test_lasso = r2_score(y_test, lasso_prediction_test)


              return (r2_test_linear, r2_test_lasso)# Your answer here
          #answer_four()
```

1.2 Part 2 - Classification

Here's an application of machine learning that could save your life! For this section of the assignment we will be working with the UCI Mushroom Data Set stored in `readonly/mushrooms.csv`. The data will be used to train a model to predict whether or not a mushroom is poisonous. The following attributes are provided:

Attribute Information:

1. cap-shape: bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s
2. cap-surface: fibrous=f, grooves=g, scaly=y, smooth=s
3. cap-color: brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y
4. bruises?: bruises=t, no=f
5. odor: almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s
6. gill-attachment: attached=a, descending=d, free=f, notched=n
7. gill-spacing: close=c, crowded=w, distant=d
8. gill-size: broad=b, narrow=n
9. gill-color: black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y
10. stalk-shape: enlarging=e, tapering=t
11. stalk-root: bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?
12. stalk-surface-above-ring: fibrous=f, scaly=y, silky=k, smooth=s
13. stalk-surface-below-ring: fibrous=f, scaly=y, silky=k, smooth=s
14. stalk-color-above-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
15. stalk-color-below-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
16. veil-type: partial=p, universal=u
17. veil-color: brown=n, orange=o, white=w, yellow=y
18. ring-number: none=n, one=o, two=t
19. ring-type: cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z
20. spore-print-color: black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y
21. population: abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y
22. habitat: grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d

The data in the mushrooms dataset is currently encoded with strings. These values will need to be encoded to numeric to work with sklearn. We'll use pd.get_dummies to convert the categorical variables into indicator variables.

```
In [116]: import pandas as pd
          import numpy as np
          from sklearn.model_selection import train_test_split


          mush_df = pd.read_csv('mushrooms.csv')
          mush_df2 = pd.get_dummies(mush_df)

          X_mush = mush_df2.iloc[:,2:]
          y_mush = mush_df2.iloc[:,1]

          # use the variables X_train2, y_train2 for Question 5
          X_train2, X_test2, y_train2, y_test2 = train_test_split(X_mush, y_mush, r
```

```
        # For performance reasons in Questions 6 and 7, we will create a smaller
        # entire mushroom dataset for use in those questions.  For simplicity we
        # the 25% test split created above as the representative subset.
        #
        # Use the variables X_subset, y_subset for Questions 6 and 7.
        X_subset = X_test2
        y_subset = y_test2
```

1.2.1 Question 5

Using `X_train2` and `y_train2` from the preceeding cell, train a DecisionTreeClassifier with default parameters and random_state=0. What are the 5 most important features found by the decision tree?

As a reminder, the feature names are available in the `X_train2.columns` property, and the order of the features in `X_train2.columns` matches the order of the feature importance values in the classifier's `feature_importances_` property.

This function should return a list of length 5 containing the feature names in descending order of importance.

Note: remember that you also need to set random_state in the DecisionTreeClassifier.

```
In [117]: def answer_five():
              from sklearn.tree import DecisionTreeClassifier

              # Your code here
              clf = DecisionTreeClassifier(random_state=0).fit(X_train2, y_train2)
              top_five = clf.feature_importances_.argsort()[::-1][:5]
              result = list(X_train2.columns[top_five])

              return result# Your answer here
```

1.2.2 Question 6

For this question, we're going to use the `validation_curve` function in `sklearn.model_selection` to determine training and test scores for a Support Vector Classifier (`SVC`) with varying parameter values. Recall that the validation_curve function, in addition to taking an initialized unfitted classifier object, takes a dataset as input and does its own internal train-test splits to compute results.

Because creating a validation curve requires fitting multiple models, for performance reasons this question will use just a subset of the original mushroom dataset: please use the variables X_subset and y_subset as input to the validation curve function (instead of X_mush and y_mush) to reduce computation time.

The initialized unfitted classifier object we'll be using is a Support Vector Classifier with radial basis kernel. So your first step is to create an `SVC` object with default parameters (i.e. `kernel='rbf', C=1`) and `random_state=0`. Recall that the kernel width of the RBF kernel is controlled using the `gamma` parameter.

With this classifier, and the dataset in X_subset, y_subset, explore the effect of `gamma` on classifier accuracy by using the `validation_curve` function to find the training and test scores for 6 values of `gamma` from `0.0001` to `10` (i.e. `np.logspace(-4,1,6)`). Recall that you can specify

what scoring metric you want validation_curve to use by setting the "scoring" parameter. In this case, we want to use "accuracy" as the scoring metric.

For each level of `gamma`, `validation_curve` will fit 3 models on different subsets of the data, returning two 6x3 (6 levels of gamma x 3 fits per level) arrays of the scores for the training and test sets.

Find the mean score across the three models for each level of `gamma` for both arrays, creating two arrays of length 6, and return a tuple with the two arrays.

e.g.

if one of your array of scores is

```
array([[ 0.5,   0.4,   0.6],
       [ 0.7,   0.8,   0.7],
       [ 0.9,   0.8,   0.8],
       [ 0.8,   0.7,   0.8],
       [ 0.7,   0.6,   0.6],
       [ 0.4,   0.6,   0.5]])
```

it should then become

```
array([ 0.5,   0.73333333,   0.83333333,   0.76666667,   0.63333333, 0.5])
```

This function should return one tuple of numpy arrays `(training_scores, test_scores)` *where each array in the tuple has shape* `(6,)`.

```
In [121]: def answer_six():
              from sklearn.svm import SVC
              from sklearn.model_selection import validation_curve
              # Your code here
              param_range = np.logspace(-4,1,6)
              clf = SVC(kernel='rbf', C=1, random_state=0)
              train_scores, test_scores = validation_curve(SVC(), X_subset, y_subse
                                                           param_name='gamma',
                                                           param_range=param_range, cv=3
              train_scores_mean = np.mean(train_scores, axis=1)
              test_scores_mean = np.mean(test_scores, axis=1)
              return (train_scores_mean, test_scores_mean)
          #answer_six()

Out[121]: (array([ 0.56647847,   0.93155951,   0.99039881,   1.        ,   1.        ,
           array([ 0.56768547,   0.92959558,   0.98965952,   1.        ,   0.99507994,
                 0.52240279]))
```

1.2.3 Question 7

Based on the scores from question 6, what gamma value corresponds to a model that is underfitting (and has the worst test set accuracy)? What gamma value corresponds to a model that is overfitting (and has the worst test set accuracy)? What choice of gamma would be the best choice

for a model with good generalization performance on this dataset (high accuracy on both training and test set)?

Hint: Try plotting the scores from question 6 to visualize the relationship between gamma and accuracy. Remember to comment out the import matplotlib line before submission.

This function should return one tuple with the degree values in this order: `(Underfitting, Overfitting, Good_Generalization)` *Please note there is only one correct solution.*

```
In [124]: def answer_seven():

              training_scores, test_scores = answer_six()
              param_range = np.logspace(-4, 1, 6)


              train_scores_sorted = np.sort(training_scores)
              test_scores_sorted = np.sort(test_scores)

              Underfitting = 0
              Overfitting = 0
              Good_Generalization = 0
              min_train_scores = np.min(training_scores)
              max_train_scores = np.max(training_scores)
              min_test_scores = np.max(test_scores)
              max_test_scores = np.max(test_scores)

              for gamma, data in zip(param_range, zip(training_scores, test_scores)
                  if data[0] <= train_scores_sorted[1] and data[1] <= test_scores_s
                      Underfitting = gamma
                  if data[0] > train_scores_sorted[1] and data[1] <= test_scores_sc
                      Overfitting = gamma
                  if data[0] == max_train_scores and data[1] == max_test_scores:
                      Good_Generalization = gamma

              return (Underfitting, Overfitting, Good_Generalization)
          #answer_seven()

Out[124]: (0.0001, 10.0, 0.10000000000000001)

In [ ]:
```