MM20B007 Tut 5

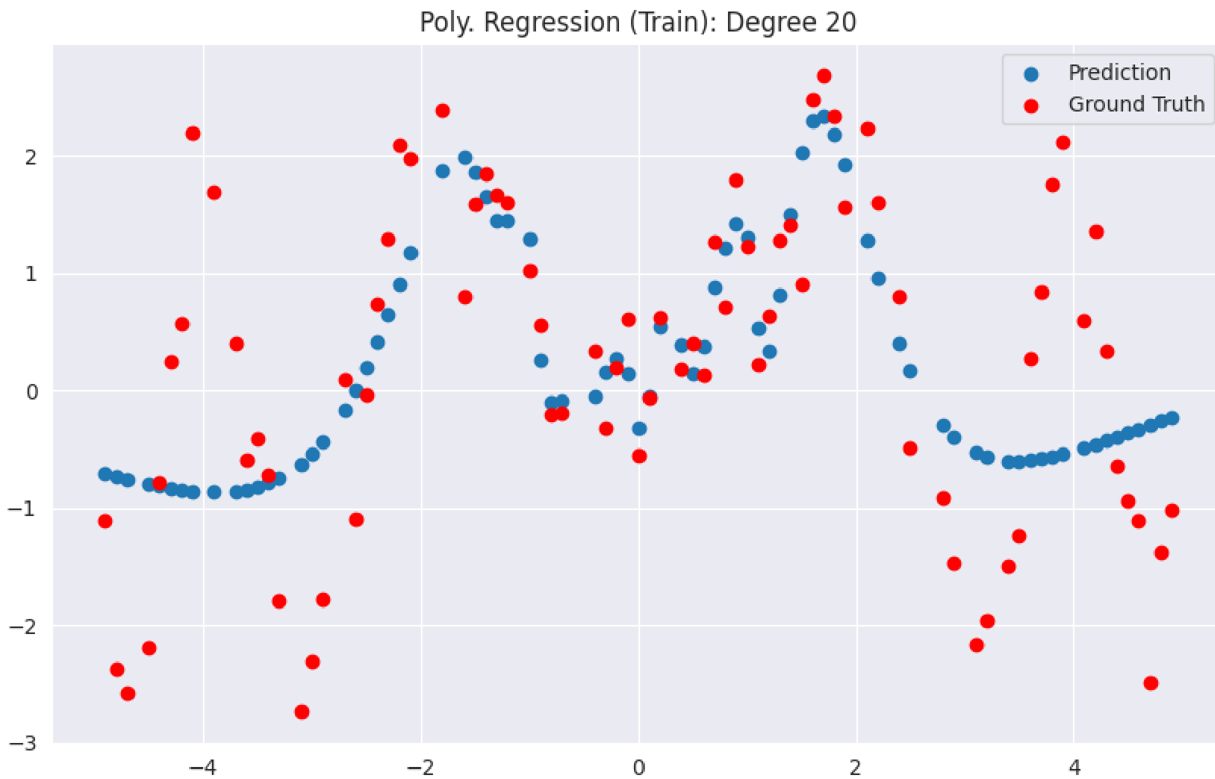
Problem 1

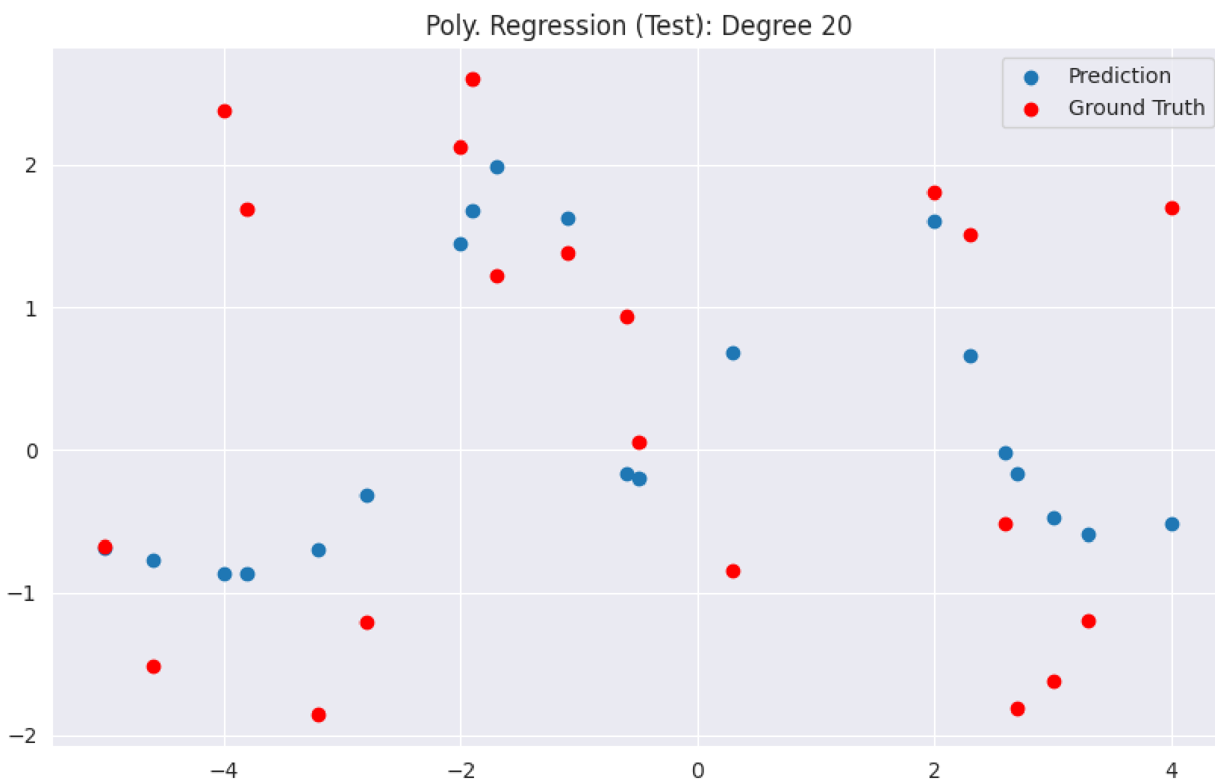
[Bias-Variance Tradeoff] In this problem, the goal is to use Dataset 2 described in the tutorial notebook and plot training and testing error curves against model complexity. Use polynomial regression, discussed in class to fit polynomial of degree k to the data. Search space for the degree of the polynomial can be taken to be k ∈ [1, 30]. Plot following 2 curves: Train/Test MSE vs Degree of Polynomial Regression Report optimal choice of k based on the testing loss.

Importing necessary packages

```
import math
import random
import sklearn
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures

num= 100
random.seed = 42
np.random.seed = 42
sns.set_style("darkgrid")

dataset_path = '/content/drive/MyDrive/sem 7/ID5055/Tutorial
5/poly_reg2.csv'

# Function to load data and get train and test data
def load_data(path):
  data = pd.read_csv(path)
  arr = data.to_numpy().T

  return train_test_split(arr[0], arr[1], test_size = 0.2,
random_state = 42, shuffle = True)

# Funtion for poly regression
def poly_regression(path, k = 2, plot = True):
  x_train, x_test, y_train, y_test = load_data(path)
```

```
  poly_train = PolynomialFeatures(degree = k, include_bias = False)
  poly_x_train = poly_train.fit_transform(x_train.reshape(-1, 1))
  poly_x_train = sklearn.preprocessing.normalize(poly_x_train)

  poly_test = PolynomialFeatures(degree = k, include_bias = False)
  poly_x_test = poly_test.fit_transform(x_test.reshape(-1, 1))
  poly_x_test = sklearn.preprocessing.normalize(poly_x_test)

  poly_model = LinearRegression()
  poly_model.fit(poly_x_train, y_train)
  y_pred_train = poly_model.predict(poly_x_train)
  y_pred_test = poly_model.predict(poly_x_test)

  mse_train = mean_squared_error(y_train, y_pred_train)
  mse_test = mean_squared_error(y_test, y_pred_test)

  # print('Degree', k)
  # print('Mean Squared Error (TRAIN):', mse_train)
  # print('Mean Squared Error (TEST):', mse_test)

  if plot:
    # Train data visualization
    plt.figure(figsize = (10, 6))
    plt.scatter(x_train, y_pred_train)
    plt.scatter(x_train, y_train, c = 'r')
    plt.legend(['Prediction', 'Ground Truth'])
    plt.title(r'Poly. Regression (Train): Degree {}'.format(k))
    plt.show()

    # Test data visualization
    plt.figure(figsize = (10, 6))
    plt.scatter(x_test, y_pred_test)
    plt.scatter(x_test, y_test, c = 'r')
    plt.legend(['Prediction', 'Ground Truth'])
    plt.title(r'Poly. Regression (Test): Degree {}'.format(k))
    plt.show()

  return mse_train, mse_test

# For dataset 2

a, b = poly_regression(dataset_path, k = 20, plot = True)
print('Degree', 20)
print('Mean Squared Error (TRAIN):', a)
print('Mean Squared Error (TEST):', b)
```


Poly. Regression (Train): Degree 20
Prediction
Ground Truth
2
1
0
−1
−2
−3
−4
−2
0
2
4

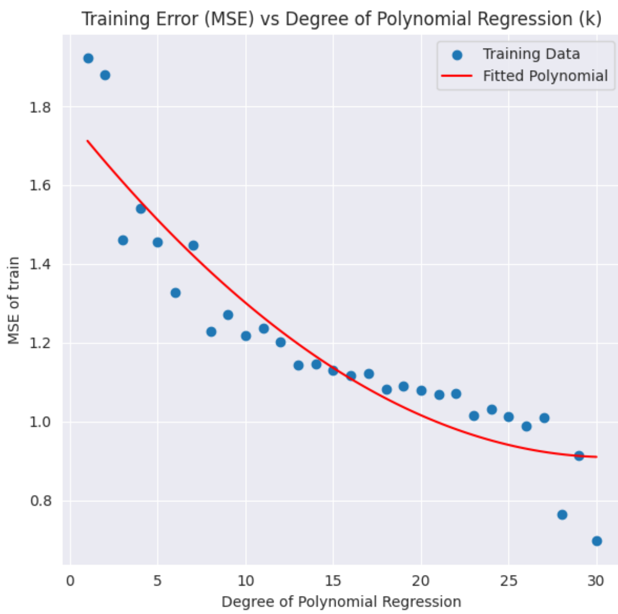

Poly. Regression (Test): Degree 20
Prediction
Ground Truth
2
1
0
−1
−2
−4
−2
0
2
4

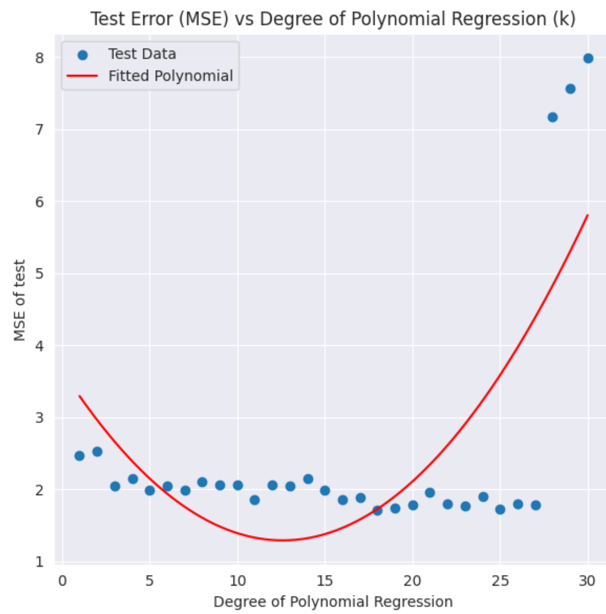
```
Degree 20
Mean Squared Error (TRAIN): 1.0783998135488606
Mean Squared Error (TEST): 1.7840628969537604
```

```
# Relation between degree of polynomial regression and MSE

def k_vs_mse(k, degree_of_polyfit):
    mse_corresponding_to_k = {}

    for i in range(1, k):
        train_mse, test_mse = poly_regression(dataset_path, k=i, 
plot=False)
        mse_corresponding_to_k[i] = [train_mse, test_mse]

    [print(f'for k: {k} goodness values are 
{mse_corresponding_to_k[k]}') for k in mse_corresponding_to_k.keys()]

    train_mse_values = [mse[0] for mse in 
mse_corresponding_to_k.values()]
    test_mse_values = [mse[1] for mse in 
mse_corresponding_to_k.values()]
    degrees = list(mse_corresponding_to_k.keys())

    # Fit polynomial curves to the scatter points
    train_coefficients = np.polyfit(degrees, train_mse_values, 
degree_of_polyfit)
    test_coefficients = np.polyfit(degrees, test_mse_values, 
degree_of_polyfit)

    # Create polynomial functions
    train_poly = np.poly1d(train_coefficients)
    test_poly = np.poly1d(test_coefficients)

    x_values = np.linspace(min(degrees), max(degrees), 100)

    fig, ax = plt.subplots(1, 2, figsize=(12, 6))

    # Plot training error (MSE) vs Degree of Polynomial Regression (k) 
with the fitted curve
    ax[0].scatter(degrees, train_mse_values, label='Training Data')
    ax[0].plot(x_values, train_poly(x_values), label='Fitted 
Polynomial', color='red')
    ax[0].set_ylabel('MSE of train')
    ax[0].set_xlabel('Degree of Polynomial Regression')
    ax[0].set_title('Training Error (MSE) vs Degree of Polynomial 
Regression (k)')
    ax[0].legend()

    # Plot test error (MSE) vs Degree of Polynomial Regression (k) 
with the fitted curve
```

```
    ax[1].scatter(degrees, test_mse_values, label='Test Data')
    ax[1].plot(x_values, test_poly(x_values), label='Fitted
Polynomial', color='red')
    ax[1].set_ylabel('MSE of test')
    ax[1].set_xlabel('Degree of Polynomial Regression')
    ax[1].set_title('Test Error (MSE) vs Degree of Polynomial
Regression (k)')
    ax[1].legend()

    plt.tight_layout()
    plt.show()

k_vs_mse(31, 2)
```

```
for k: 1 goodness values are [1.922344044282697, 2.472440789929736]
for k: 2 goodness values are [1.8803850247470137, 2.528384419991515]
for k: 3 goodness values are [1.4620617871706039, 2.0479225453996497]
for k: 4 goodness values are [1.5416281443086395, 2.1460324189503117]
for k: 5 goodness values are [1.4564690063263845, 1.9959867579135373]
for k: 6 goodness values are [1.3270551293251713, 2.050579893096593]
for k: 7 goodness values are [1.4488152523761275, 1.9911302504294752]
for k: 8 goodness values are [1.2300870260618988, 2.1086299148178536]
for k: 9 goodness values are [1.2727103940988558, 2.0602288832805344]
for k: 10 goodness values are [1.2179282875955129, 2.0569774301257566]
for k: 11 goodness values are [1.2380848616381255, 1.8568492208285445]
for k: 12 goodness values are [1.2023398227894166, 2.0567793378852266]
for k: 13 goodness values are [1.1442877929155122, 2.0473147698417256]
for k: 14 goodness values are [1.1470533793992557, 2.153307808606411]
for k: 15 goodness values are [1.1308749336937194, 1.9933067318772575]
for k: 16 goodness values are [1.11598138099663, 1.8632171134452615]
for k: 17 goodness values are [1.1227812447074352, 1.8930783652379621]
for k: 18 goodness values are [1.0809907774378966, 1.7177410773096344]
for k: 19 goodness values are [1.0910379812114213, 1.736333537200751]
for k: 20 goodness values are [1.0783998135488606, 1.7840628969537604]
for k: 21 goodness values are [1.069520386386658, 1.961110413343603]
for k: 22 goodness values are [1.07254265361323, 1.8015447556556317]
for k: 23 goodness values are [1.014742946851216, 1.772323051486029]
for k: 24 goodness values are [1.0303277408733742, 1.9015641003735617]
for k: 25 goodness values are [1.01258502651729, 1.7349299296303637]
for k: 26 goodness values are [0.9877751739949623, 1.8014077799179087]
for k: 27 goodness values are [1.0089568667296025, 1.792258707208886]
for k: 28 goodness values are [0.7633244981137262, 7.169016071958339]
for k: 29 goodness values are [0.9128770444295684, 7.558650513480917]
for k: 30 goodness values are [0.6980473212217384, 7.98531878837407]
```





Observation

1. From the test loss it is clear that the most optimum degree of polynomial regression (k) value is 18 because for k: 18 the train-test goodness values are [1.0809907774378966, 1.7177410773096344], which the least we can find.

2. However, when we fitted the test losses in a polynomial curve the dip comes between k values 10 and 15, so we can consider them as optimal values.

3. As we increase the degree of polynomial regression we see that train mse reduces drastically but the test mse increases implying that the model overfitted.