Assignment 8: Polynomial Regression

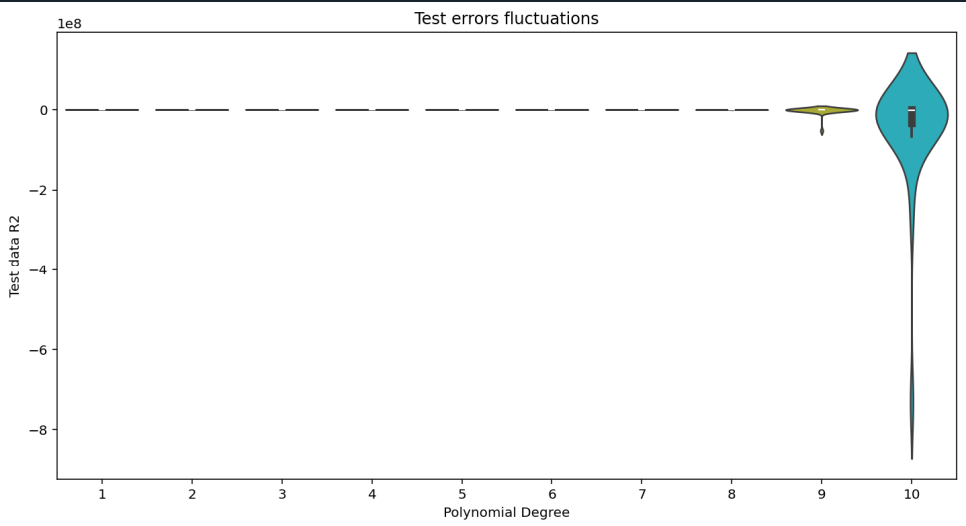
Name: Jasmine Imtiyaj Sayyad

PRN: 22510033

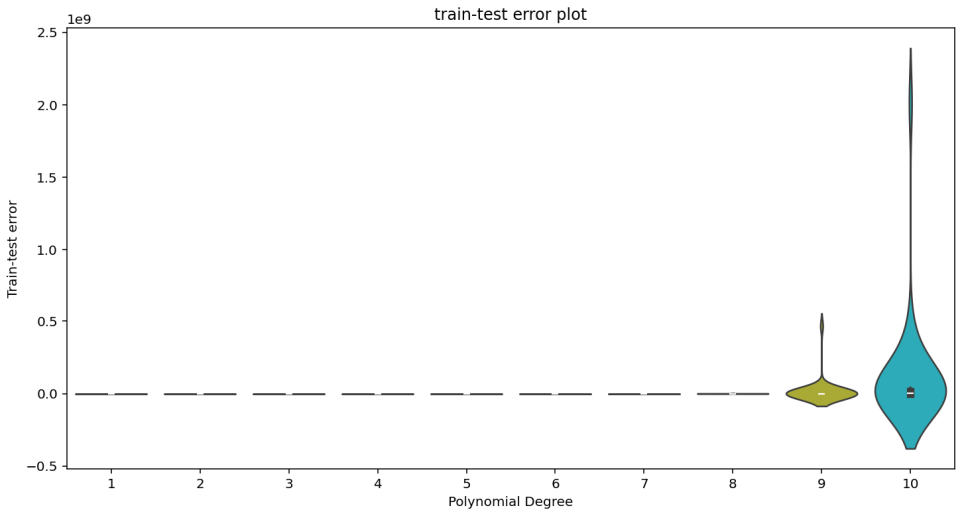
Batch: T2

1. Fit a polynomial regression model (y modelled as linear function of various higher degree terms of x) in following steps
2. 80:20 split into train and test set
3. Create 30 samples of size 20 each.
   1. For each of the sample
      1. Incrementally fit polynomials of degree upto 10.
      2. Measure train accuracy using 20 train data points and test accuracy using the full test set
   2. Use violin plot to observe the fluctuations in test error corresponding to the degree

Observations:



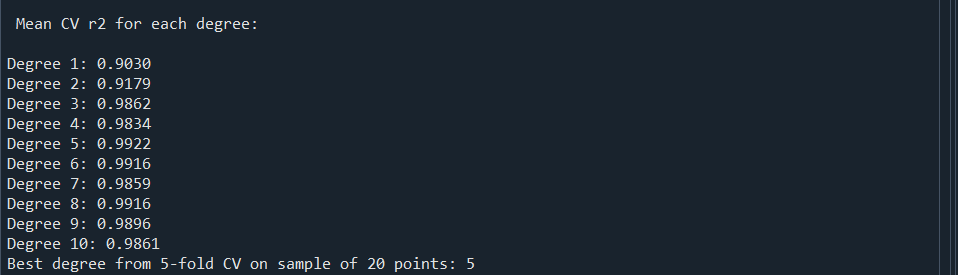
* All degrees from 1 to 8 showflatviolins at or near 0 R².
* This means the test R² scores are:
  + very low indicating poor test performance
  + And not varying much (no matter which 20-sample subset is chosen)
* These polynomial degrees can’t determine data pattern properly leading to underfitting.
* For degree 9, the violin is stretched slightly indicating the start of overfitting and reducing test performance.
* For degree 10, negative R2 values for test data indicates high level of overfitting, and poor generalization.
  1. Create another violin plot of degree vs (train error – test error)



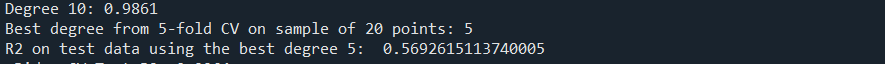
Observations:

* For Degrees 1 to 8, difference between train and test errors is very low indicating the model performs well on both seen and unseen data.
* For degree 9 there is noticible increase in train-test error. Indicates that the model fits training data well but not equally performs on testing data.
* At degree 10, the difference is very high indicating higher overfitting.

1. From the training data sample 20 points.
   1. Use k=5 fold cross validation to determine the highest degree polynomial you should fit on this sample



* 1. Once you finalise the highest degree, train the model of that degree on the 20 sample points
  2. Use the above model to measure accuracy of model on the test set

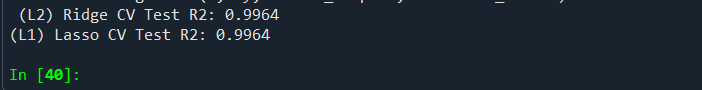


Observation:

As we are training the model on just 20 sample points, It’ll try to fit the training data but will fail to recognize the pattern in the data. When we test that model on the 2000 test data set, the accuracy will be less therefore R2 value is also less.

K-fold (5) Cross validation reduces overfitting risk in the train data i.e. 20 sample points. So generalization is poor.

1. On the full train data, use k (=10) fold cross validation as well as l1 and l2 regularisation to fit a polynomial and measure accuracy on test data.



Observation:

Both L1 (Lasso) and L2 (Ridge) models gave very high accuracy (with R2 = 0.9964 ) on the test data. This means the model is doing correct predictions and is not overfitting. Since both types of regularization gave similar results, it suggests that all the features being used are likely useful, and there’s not much unnecessary information. The polynomial degree chosen works well for this dataset.

**Code:**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split, KFold, cross\_val\_score

import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression, Ridge, Lasso

from sklearn.metrics import r2\_score

import seaborn as sns

filepath = ("polynomial\_regression.csv")

df = pd.read\_csv(filepath)

# %%

np.random.seed(25)

x = df[['x']]

y = df['y']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=29)

no\_of\_samples = 30

sample\_size = 20

degree\_of\_polynomial= np.arange(1,11)

train\_data\_error = []

test\_data\_error = []

# %%

for sample in range(no\_of\_samples):

indices = np.random.choice(len(x\_train), sample\_size, replace=False)

x\_sample = x\_train.iloc[indices]

y\_sample = y\_train.iloc[indices]

train\_sample\_error = []

test\_sample\_error = []

for degree in range(1,11):

polynomial = PolynomialFeatures(degree=degree)

x\_train\_poly = polynomial.fit\_transform(x\_sample)

x\_test\_poly = polynomial.transform(x\_test)

model = LinearRegression()

model.fit(x\_train\_poly, y\_sample)

y\_train\_predict = model.predict(x\_train\_poly)

y\_test\_predict = model.predict(x\_test\_poly)

train\_r2 = r2\_score(y\_sample, y\_train\_predict)

test\_r2 = r2\_score(y\_test, y\_test\_predict)

train\_sample\_error.append(train\_r2)

test\_sample\_error.append(test\_r2)

train\_data\_error.append(train\_sample\_error)

test\_data\_error.append(test\_sample\_error)

train\_data\_error = np.array(train\_data\_error)

test\_data\_error = np.array(test\_data\_error)

average\_train\_r2 = np.mean(train\_data\_error, axis=0)

average\_test\_r2 = np.mean(test\_data\_error, axis=0)

print("\nAverage R² scores over all samples:")

for degree, train\_r2, test\_r2 in zip(degree\_of\_polynomial, average\_train\_r2, average\_test\_r2):

print(f"Degree {degree}: Average Train R² = {train\_r2:.4f}, Average Test R² = {test\_r2:.4f}")

train\_err\_df = pd.DataFrame(train\_data\_error, columns=degree\_of\_polynomial)

test\_err\_df = pd.DataFrame(test\_data\_error, columns=degree\_of\_polynomial)

# violin plot to observe the fluctuations in test error corresponding to the degree

plt.figure(figsize=(12,6))

sns.violinplot(data=test\_err\_df)

plt.xlabel("Polynomial Degree")

plt.ylabel("Test data R2")

plt.title("Test errors fluctuations")

plt.show()

# c. Create another violin plot of degree vs (train error – test error)

difference = train\_err\_df - test\_err\_df

plt.figure(figsize=(12,6))

sns.violinplot(data=difference)

plt.xlabel("Polynomial Degree")

plt.ylabel("Train-test error")

plt.title("train-test error plot")

plt.show()

# Use k=5 fold cross validation to determine the highest degree polynomial you should fit on this sample

new\_sample\_index = np.random.choice(len(x\_train), sample\_size, replace=False)

new\_sample\_x = x\_train.iloc[new\_sample\_index]

new\_sample\_y = y\_train.iloc[new\_sample\_index]

kf = KFold(n\_splits=5, shuffle=True, random\_state=67)

cv\_scores = []

print("\n Mean CV r2 for each degree:\n")

for degree in range(1, 11):

poly = PolynomialFeatures(degree=degree)

x\_poly = poly.fit\_transform(new\_sample\_x)

model = LinearRegression()

scores = cross\_val\_score(model, x\_poly, new\_sample\_y, cv=kf, scoring='r2')

cv\_scores.append(scores.mean())

print(f"Degree {degree}: {scores.mean():.4f}")

best\_degree = np.argmax(cv\_scores) + 1

print("Best degree from 5-fold CV on sample of 20 points:", best\_degree)

# train the model of that degree on the 20 sample points

polynomial\_best = PolynomialFeatures(degree= best\_degree)

x\_sample\_poly = polynomial\_best.fit\_transform(new\_sample\_x)

final\_model = LinearRegression()

final\_model.fit(x\_sample\_poly, new\_sample\_y)

# Used the above model to measure accuracy of model on the test set

x\_test\_final = polynomial\_best.transform(x\_test)

y\_test\_final\_pred = final\_model.predict(x\_test\_final)

final\_test\_r2 = r2\_score(y\_test, y\_test\_final\_pred)

print("R2 on test data using the best degree 5: ", final\_test\_r2)

kf\_10 = KFold(n\_splits=10, shuffle=True, random\_state=42)

for degree in [best\_degree]:

poly\_full = PolynomialFeatures(degree=degree)

X\_full\_poly = poly\_full.fit\_transform(x\_train)

X\_test\_poly\_full = poly\_full.transform(x\_test)

# Ridge (L2)

ridge = Ridge(alpha=1.0)

ridge\_scores = cross\_val\_score(ridge, X\_full\_poly, y\_train, cv=kf\_10, scoring='r2')

ridge.fit(X\_full\_poly, y\_train)

ridge\_test\_r2 = r2\_score(y\_test, ridge.predict(X\_test\_poly\_full))

print(f" (L2) Ridge CV Test R2: {ridge\_test\_r2:.4f}")

# Lasso (L1)

lasso = Lasso(alpha=0.01, max\_iter=10000)

lasso\_scores = cross\_val\_score(lasso, X\_full\_poly, y\_train, cv=kf\_10, scoring='r2')

lasso.fit(X\_full\_poly, y\_train)

lasso\_test\_r2 = r2\_score(y\_test, lasso.predict(X\_test\_poly\_full))

print(f"(L1) Lasso CV Test R2: {lasso\_test\_r2:.4f}")