Predicting higher education student performance based on parents’ levels of education using Linear and Decision Tree Regression

Introduction

There is a lot of discussion and evidence that one of the things affecting the performance of students is the level of education of their parents. There have also been many studies done about this topic. Of Course many other things affect the performances of students but in this project I want to focus on examining the influence of a student's parents’ level of education.

This report first introduces the problem formulated as a machine learning problem explaining the data points, features and labels. Next the dataset and its source are introduced and explained briefly. In the next part, the machine learning methods are introduced with explanations as to why those specific methods were chosen. The methods include Linear Regression and Decision Tree Regression. Finally this report includes the results, where the fitted models are compared and a conclusion part for final discussion.

Problem Formulation

Many things can affect how well students perform in classes and exams. For this project I want to see if it is possible to, on some level, predict students’ performance based on their parents’ levels of education.

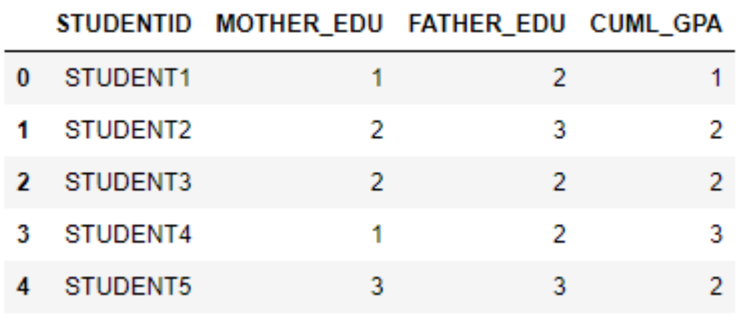
The data points are students. The data consists of 145 students. In the original dataset each student data point has 33 features including for example: student ID, age, graduated high school type, scholarship type, regular artistic or sports activity, mother’s education, father’s education, weekly study hours and cumulative grade point average in the last semester.

For this problem the cumulative GPA of the last semester is going to represent the performance of the student. This means that it is going to work as the label as the label is what is being predicted. The features are going to be the descriptive attributes, in this case the parents’ levels of education.

Label: Expected cumulative GPA in the graduation (/4.00) (1: <2.00, 2: 2.00-2.49, 3: 2.50-2.99, 4: 3.00-3.49, 5: above 3.49)
Features: Mother’s level of education, Father’s level of education (1: primary school, 2: secondary school, 3: high school, 4: university, 5: MSc., 6: Ph.D.)

The Dataset

The dataset used in this project is a dataset consisting of data that was collected from the Faculty of Engineering and Faculty of Educational Sciences students in 2019. The dataset can be found on Kaggle (link). The amount of data points is 145.



	STUDENTID	MOTHER_EDU	FATHER_EDU	CUML_GPA
0	STUDENT1	1	2	1
1	STUDENT2	2	3	2
2	STUDENT3	2	2	2
3	STUDENT4	1	2	3
4	STUDENT5	3	3	2

Here is a snippet of the dataset with the extra columns dropped. The education levels and GPA are simply presented with numbers 1-5 and 1-6. The numbers are explained above in the part where the label and features are explained.

Methods

As previously mentioned, the dataset consists of 145 data points that are students. The data was preprocessed by removing unnecessary columns that contained features which will not be used in this project.

The data originally contained 33 different features for each data point but I decided to use only two of them because I want to try and predict the student performance only based on the two features (mother's education and father's education). I chose these features because the possible correlation between these features and the label is interesting and there is evidence of it.

1. Linear Regression

I decided to use linear regression as one of the machine learning methods for the prediction. Linear regression is one of the most frequently used predictive modelling techniques in machine learning. I chose linear regression for this problem because regression is useful for prediction problems. For my problem I'm using multiple linear regression because I have two independent variables (features) used for prediction.

The hypothesis space of a linear regression model:

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$$

Where y is the dependent or target variable that is being predicted and the x's are the values of the different features, the independent variables.

2. Decision Tree

For the first ML method in this project, I decided to try out using linear regression for prediction. After visualising the data, I noticed that the correlation isn't very linear, meaning multiple linear regression wasn't maybe the best choice for prediction with the type of data that I have.

On top of linear and logistic regression models, decision trees are widely used predictive machine learning models. I decided to use the decision tree regression for prediction in my machine learning problem because decision trees are not affected by the non-linearity of the features and labels, meaning a decision tree regression model could perform better than a linear regression model. The hypothesis space of decision tree models is the set of all hypotheses which is represented by a family of decision trees.

Loss Function

Mean squared loss (MSE) was chosen for the loss function for the linear regression because it is often used with linear regression and it is provided by Scikit-learn. It is also used with decision tree regression which is also why it is the right choice for a loss function.

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

Mean squared error is calculated like above, where the predicted value is subtracted from the true value.

Model Validation

First I tried doing the validation by randomly dividing the data into train and validation sets so that the validation/test set is 33% and the rest of the data is used for training. The splitting is done with a simple train-test split using the train_test_split- function from sklearn.

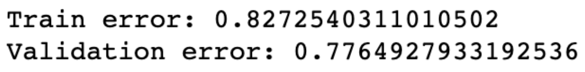
After doing some further investigation and research, I realised that doing a simple train-test split might provide bad results when the data set is small. This is because when using a simple train-test split, the portion of the data that is chosen for testing might consist of mostly abnormal data points. This can make the model perform badly. The data set I am using only has 145 data points meaning it is a small data set. Instead of doing a simple train-test split, I decided to use KFold for the final machine learning model. KFold works better with a small data set because each data point can both be in the model and used for testing. KFold divides the data set into k subsets. The splitting was done using the KFold class from sklearn and 10 folds were used.

The test set was constructed by using a simple train-test split because the test set only needs to be a small part of the data set that is not used in training nor in validating. As the data set is really small, I decided to take only a small portion of 10% for the test set.

Results

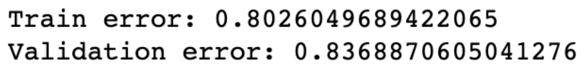
For the first method (linear regression), the validation error was smaller than the training error. This is not usually the case with machine learning models and it indicates that the data might not have been split properly. This could be resulting from not considering the small amount of data when doing the simple train-test split. For the two methods two be comparable, I also created a linear regression model where the data was split using KFold and for that model, the train error was smaller than the validation error as expected.

Training and validation errors of the linear regression model with single train-test split:



```
Train error: 0.8272540311010502
Validation error: 0.7764927933192536
```

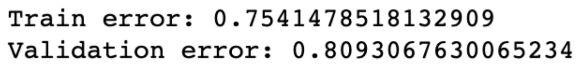
Training and validation errors of the linear regression model using KFold:



```
Train error: 0.8026049689422065
Validation error: 0.8368870605041276
```

For the second method (decision tree), the validation and training errors seem better. The train error is lower than the validation error as expected and both are lower than for the linear regression model. This is one of the reasons for eventually choosing decision tree regression to be the final chosen method.

Training and validation errors of the decision tree regression model using KFold:



```
Train error: 0.7541478518132909
Validation error: 0.8093067630065234
```

Test error of the final decision tree regression model: 0.5891630017642907

Conclusion

Overall I think the test error of the chosen method is surprisingly good but always could be better. The chosen decision tree regression model seems to have potential in solving this type of machine learning problem. There is still almost always room for improvement. Because the data set is so small, the test error can be misleading as some or many of the data points in the test set could be outliers. The training error is not too much smaller than the validation error, which indicates that there is no overfitting. I do think that my machine learning problem of predicting student performance from parents' levels of education, was somewhat solved acceptably but the model could be improved in many ways. The data set was small for a data set used in machine learning. With a larger data set, the model could perform better than it performs now. It could also be interesting to see what other features could be used to better predict student performance and for example use a feature selection method to find those features.

References and Links

Link to the source of the dataset:
https://www.kaggle.com/csafrit2/higher-education-students-performance-evaluation

Scikit-learn linear regression model:
https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html#sphx-glr-auto-examples-linear-model-plot-ols-py

Scikit-learn decision tree regressor:
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

Scikit-learn mean squared error:
https://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-error

Scikit-learn train-test split:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Scikit-learn KFold
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

(Code as appendix)

ML Project

March 31, 2022

[1]:
```
%config Completer.use_jedi = False  # enable code auto-completion
import numpy as np    # library for numerical computations (vectors, matrices,␣
 ↪tensors)
import pandas as pd    # library for data manipulation and analysis
import matplotlib.pyplot as plt    # library providing tools for plotting data
from sklearn.preprocessing import PolynomialFeatures    # function to generate␣
 ↪polynomial and interaction features
from sklearn.linear_model import LinearRegression, HuberRegressor    # classes␣
 ↪providing Linear Regression with ordinary squared error loss and Huber loss,␣
 ↪respectively
from sklearn.metrics import mean_squared_error, accuracy_score
from sklearn.model_selection import train_test_split
```

0.1 The data

[2]:
```
rawdata = pd.read_csv('student_prediction.csv')
rawdata.head(5)
```

[2]:
```
  STUDENTID  AGE  GENDER  HS_TYPE  SCHOLARSHIP  WORK  ACTIVITY  PARTNER  \
0  STUDENT1    2       2        3            3     1         2        2
1  STUDENT2    2       2        3            3     1         2        2
2  STUDENT3    2       2        2            3     2         2        2
3  STUDENT4    1       1        1            3     1         2        1
4  STUDENT5    2       2        1            3     2         2        1

   SALARY  TRANSPORT  …  PREP_STUDY  PREP_EXAM  NOTES  LISTENS  \
0       1          1  …           1          1      3        2
1       1          1  …           1          1      3        2
2       2          4  …           1          1      2        2
3       2          1  …           1          2      3        2
4       3          1  …           2          1      2        2

   LIKES_DISCUSS  CLASSROOM  CUML_GPA  EXP_GPA  COURSE ID  GRADE
0              1          2         1        1          1      1
1              3          2         2        3          1      1
2              1          1         2        2          1      1
3              2          1         3        2          1      1
```

```
4               2          1         2        2           1      1

[5 rows x 33 columns]
```

[3]:
```
rawdata.columns
```

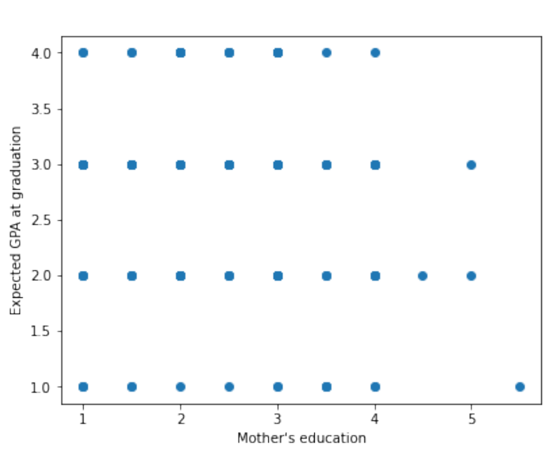
[3]:
```
Index(['STUDENTID', 'AGE', 'GENDER', 'HS_TYPE', 'SCHOLARSHIP', 'WORK',
       'ACTIVITY', 'PARTNER', 'SALARY', 'TRANSPORT', 'LIVING', 'MOTHER_EDU',
       'FATHER_EDU', '#_SIBLINGS', 'KIDS', 'MOTHER_JOB', 'FATHER_JOB',
       'STUDY_HRS', 'READ_FREQ', 'READ_FREQ_SCI', 'ATTEND_DEPT', 'IMPACT',
       'ATTEND', 'PREP_STUDY', 'PREP_EXAM', 'NOTES', 'LISTENS',
       'LIKES_DISCUSS', 'CLASSROOM', 'CUML_GPA', 'EXP_GPA', 'COURSE ID',
       'GRADE'],
      dtype='object')
```

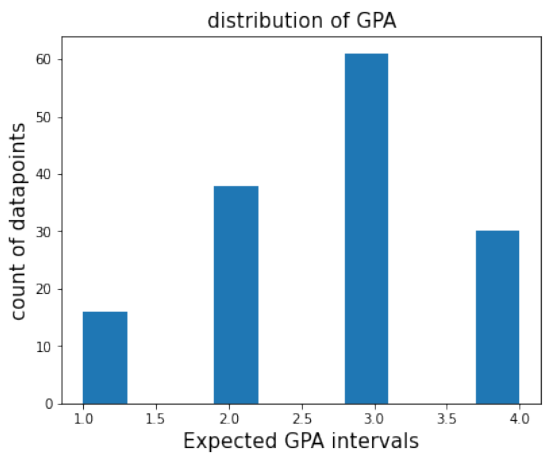
[4]:
```
data = rawdata.drop(['AGE', 'GENDER', 'HS_TYPE', 'SCHOLARSHIP', 'WORK',
       'ACTIVITY', 'PARTNER', 'SALARY', 'TRANSPORT', 'LIVING', '#_SIBLINGS',␣
 ↪'KIDS', 'MOTHER_JOB', 'FATHER_JOB',
       'STUDY_HRS', 'READ_FREQ', 'READ_FREQ_SCI', 'ATTEND_DEPT', 'IMPACT',
       'ATTEND', 'PREP_STUDY', 'PREP_EXAM', 'NOTES', 'LISTENS',
       'LIKES_DISCUSS', 'CLASSROOM', 'CUML_GPA', 'COURSE ID',
       'GRADE'], axis=1)
data['PARENTS_AVG_EDU'] = data[['MOTHER_EDU', 'FATHER_EDU']].mean(axis=1)
data.head(5)
```

[4]:
```
  STUDENTID  MOTHER_EDU  FATHER_EDU  EXP_GPA  PARENTS_AVG_EDU
0  STUDENT1           1           2        1              1.5
1  STUDENT2           2           3        3              2.5
2  STUDENT3           2           2        2              2.0
3  STUDENT4           1           2        2              1.5
4  STUDENT5           3           3        2              3.0
```

[5]:
```
# Visualize data
fig, axes = plt.subplots(1, 2, figsize=(14,5)) # create a figure with two axes␣
 ↪(1 row,2 columns) on it
axes[0].scatter(data['PARENTS_AVG_EDU'],data['EXP_GPA']) # plot a scatter plot␣
 ↪on axes[0] to show the relation between
axes[0].set_xlabel("Mother's education")
axes[0].set_ylabel("Expected GPA at graduation")

axes[1].hist(data['EXP_GPA']) # plot a hist plot to show the distribution of␣
 ↪MaxTemp
axes[1].set_title('distribution of GPA',size=15)
axes[1].set_ylabel("count of datapoints",size=15)
axes[1].set_xlabel("Expected GPA intervals",size=15)
plt.show()
```





0.2 Prediction using multiple linear regression:

First using two separate features and the using their average as the feature.

```
[6]: # Create the feature and label vectors
X = data[['MOTHER_EDU', 'FATHER_EDU']].to_numpy() #.reshape(-1, 1)
y = data['EXP_GPA'].to_numpy()

print(len(X))
print(len(y))
print(X.size)
print(y.size)
```

```
145
145
290
145
```

```
[7]: # Split the data into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.33,␣
 ↪random_state=42)
```

```
[8]: lin_regr = LinearRegression()
lin_regr.fit(X_train, y_train)
```

```
[8]: LinearRegression()
```

```
[20]: y_train_pred = lin_regr.predict(X_train)
y_val_pred = lin_regr.predict(X_val)
print("Prediction for test set: {}".format(y_val_pred))
```

```
Prediction for test set: [2.90084649 2.87332313 3.08623498 2.71545799 2.53006949
2.53006949
 2.10424578 2.87332313 2.9558932  2.90084649 2.28963428 3.08623498
 2.50254614 2.66041127 2.66041127 3.08623498 3.08623498 3.08623498
 3.16880506 2.68793463 2.92836985 2.71545799 2.344681   2.92836985
 2.87332313 2.68793463 2.87332313 2.55759285 2.71545799 2.87332313
 2.71545799 2.53006949 2.71545799 3.08623498 2.90084649 2.50254614
 2.87332313 3.22385178 2.50254614 2.47502278 2.53006949 2.71545799
 3.08623498 2.87332313 2.47502278 2.90084649 2.68793463 3.08623498]
```

[21]:
```
diff = pd.DataFrame({'Actual value': y_val, 'Predicted value': y_val_pred})
diff.head()
```

[21]:
```
   Actual value  Predicted value
0             2         2.900846
1             3         2.873323
2             1         3.086235
3             3         2.715458
4             3         2.530069
```

[22]:
```
#meanSqErr = mean_squared_error(y_val, y_val_pred)
#print('Mean Square Error:', meanSqErr)

train_err1 = mean_squared_error(y_train, y_train_pred)
val_err1 = mean_squared_error(y_val, y_val_pred)
print("Train error: {}".format(train_err1))
print("Validation error: {}".format(val_err1))
```

```
Train error: 0.8272540311010502
Validation error: 0.7764927933192536
```

Parents' education average as feature:

[12]:
```
# Create the feature and label vectors
X2 = data['PARENTS_AVG_EDU'].to_numpy().reshape(-1, 1)
y2 = data['EXP_GPA'].to_numpy()

print(len(X2))
print(len(y2))
print(X2.size)
print(y2.size)
```

```
145
145
145
145
```

[13]:
```
# Split the data into train and validation sets
```

```
X2_train, X2_val, y2_train, y2_val = train_test_split(X2, y2, test_size=0.33,␣
 ↪random_state=42)
```

[14]:
```
lin_regr2 = LinearRegression()
lin_regr2.fit(X2_train, y2_train)
```

[14]:
```
LinearRegression()
```

[15]:
```
y2_pred = lin_regr2.predict(X2_val)
print("Prediction for test set: {}".format(y2_pred))
```

```
Prediction for test set: [2.84749923 2.93433569 3.02117214 2.67382633 2.50015342
2.50015342
 2.32648052 2.93433569 2.67382633 2.84749923 2.50015342 3.02117214
 2.58698987 2.84749923 2.84749923 3.02117214 3.02117214 3.02117214
 2.76066278 2.76066278 2.76066278 2.67382633 2.32648052 2.76066278
 2.93433569 2.76066278 2.93433569 2.41331697 2.67382633 2.93433569
 2.67382633 2.50015342 2.67382633 3.02117214 2.84749923 2.58698987
 2.93433569 2.58698987 2.58698987 2.67382633 2.50015342 2.67382633
 3.02117214 2.93433569 2.67382633 2.84749923 2.76066278 3.02117214]
```

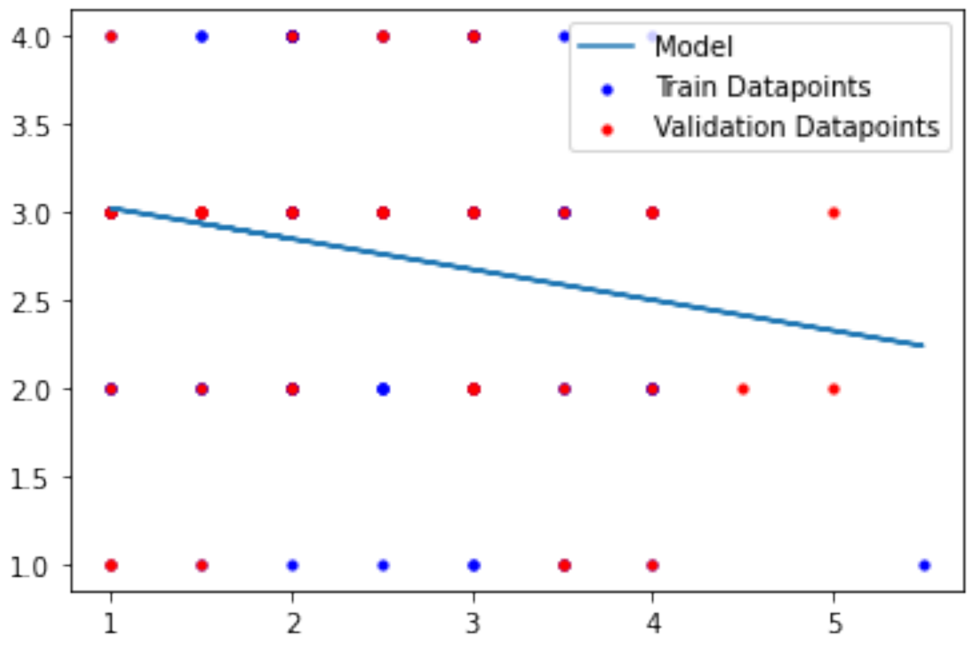
[16]:
```
diff2 = pd.DataFrame({'Actual value': y2_val, 'Predicted value': y2_pred})
diff2.head()
```

[16]:
```
   Actual value  Predicted value
0             2         2.847499
1             3         2.934336
2             1         3.021172
3             3         2.673826
4             3         2.500153
```

[17]:
```
meanSqErr2 = mean_squared_error(y2_val, y2_pred)
print('Mean Square Error:', meanSqErr2)
```

```
Mean Square Error: 0.7527283200583191
```

[18]:
```
plt.tight_layout()
plt.plot(X2, lin_regr2.predict(X2), label="Model")    # plot the polynomial␣
 ↪regression model
plt.scatter(X2_train, y2_train, color="b", s=10, label="Train Datapoints")    #␣
 ↪plot a scatter plot of y(maxtmp) vs. X(mintmp) with color 'blue' and size␣
 ↪'10'
plt.scatter(X2_val, y2_val, color="r", s=10, label="Validation Datapoints")   ␣
 ↪# do the same for validation data with color 'red'
plt.legend(loc="best")    # set the location of the legend
plt.show()
```



0.3 Prediction using decision trees

- No more sigle split because there are so little data points
- Using Kfold

[77]:
```
# Split the data into train and test sets
X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=0.1,␣
↪random_state=42)
```

[84]:
```
from sklearn.model_selection import KFold
from sklearn.tree import DecisionTreeRegressor

train_errors = []
val_errors = []
test_errors = []

# Splitting the data usign KFold
KF = KFold(n_splits=10)
for train_index, test_index in KF.split(X_train3):
    X3_train, X3_val = X_train3[train_index], X_train3[test_index]
    y3_train, y3_val = y_train3[train_index], y_train3[test_index]

    regressor = DecisionTreeRegressor(max_depth=2, random_state=0)
    regressor.fit(X3_train, y3_train)

    y3_train_pred = regressor.predict(X3_train)
    y3_val_pred = regressor.predict(X3_val)
    #meanSqErr2 = mean_squared_error(y3_val, y3_pred)
    #print("MSE: {}".format(meanSqErr2))
    train_err = mean_squared_error(y3_train, y3_train_pred)
    val_err = mean_squared_error(y3_val, y3_val_pred)
    train_errors.append(train_err)
    val_errors.append(val_err)

    y3_test_pred = regressor.predict(X_test3)
    test_err = mean_squared_error(y_test3, y3_test_pred)
    test_errors.append(test_err)

    print("Train error: {}".format(train_err))
    print("Validation error: {}".format(val_err))
    print("Test error: {}".format(test_err))

    print("Prediction for test set: {}".format(y3_val_pred))

    diff = pd.DataFrame({'Actual value': y3_val, 'Predicted value':␣
 ↪y3_val_pred})
    print(diff.head())
    #diff3
```

```
Train error: 0.8200991364344865
Validation error: 0.3691396210772088
Test error: 0.5879608968943066
Prediction for test set: [2.84269663 2.84269663 2.84269663 2.84269663 2.84269663
2.84269663
 2.84269663 2.38461538 2.84269663 2.84269663 2.84269663 2.84269663
 2.38461538]
   Actual value  Predicted value
0             3         2.842697
1             2         2.842697
2             3         2.842697
3             3         2.842697
4             3         2.842697
Train error: 0.7645577852099592
Validation error: 1.0674810055256654
Test error: 0.57027520478891
Prediction for test set: [3.         2.725      3.         2.39130435 3.
2.725
 3.         2.39130435 2.725      3.         2.39130435 2.39130435
 2.39130435]
   Actual value  Predicted value
0             3         3.000000
1             2         2.725000
2             2         3.000000
```

```
3             1         2.391304
4             2         3.000000
Train error: 0.7662867996201329
Validation error: 0.8535422602089269
Test error: 0.5906995884773664
Prediction for test set: [2.87777778 2.87777778 2.87777778 2.4        2.87777778
2.87777778
 2.87777778 2.87777778 2.4        2.4        2.87777778 2.87777778
 2.87777778]
   Actual value  Predicted value
0             3         2.877778
1             1         2.877778
2             1         2.877778
3             2         2.400000
4             4         2.877778
Train error: 0.7817927295023233
Validation error: 0.7255846120862267
Test error: 0.587116874301379
Prediction for test set: [2.42307692 2.82022472 2.82022472 2.82022472 2.82022472
2.82022472
 2.82022472 2.82022472 2.82022472 2.82022472 2.42307692 2.82022472
 2.82022472]
   Actual value  Predicted value
0             2         2.423077
1             3         2.820225
2             1         2.820225
3             4         2.820225
4             3         2.820225
Train error: 0.7339743589743589
Validation error: 1.2260208926875593
Test error: 0.6024691358024691
Prediction for test set: [2.88888889 2.88888889 2.88888889 2.22222222 2.88888889
2.88888889
 2.88888889 2.88888889 2.22222222 2.88888889 2.88888889 2.22222222
 2.88888889]
   Actual value  Predicted value
0             3         2.888889
1             2         2.888889
2             4         2.888889
3             3         2.222222
4             3         2.888889
Train error: 0.7486993682645857
Validation error: 1.0229751345063252
Test error: 0.5936515437933207
Prediction for test set: [2.39130435 2.90217391 2.90217391 2.90217391 2.39130435
2.39130435
 2.90217391 2.90217391 2.39130435 2.90217391 2.39130435 2.90217391
 2.90217391]
```

```
   Actual value  Predicted value
0            1         2.391304
1            1         2.902174
2            2         2.902174
3            4         2.902174
4            2         2.391304
Train error: 0.7856521050965495
Validation error: 0.6795333438851957
Test error: 0.5907093049839963
Prediction for test set: [2.875      2.875      2.875      2.875      2.875
2.875
 2.875      2.875      2.37037037 2.875      2.875      2.875
 2.875     ]
   Actual value  Predicted value
0            2            2.875
1            3            2.875
2            3            2.875
3            4            2.875
4            4            2.875
Train error: 0.7743209876543208
Validation error: 0.80110465337132
Test error: 0.5891292181069958
Prediction for test set: [2.82222222 2.32       2.82222222 2.82222222 2.82222222
2.32
 2.82222222 2.82222222 2.82222222 2.32       2.82222222 2.82222222
 2.82222222]
   Actual value  Predicted value
0            4         2.822222
1            3         2.320000
2            4         2.822222
3            3         2.822222
4            4         2.822222
Train error: 0.8033043015756929
Validation error: 0.5218807285439389
Test error: 0.5907359459674144
Prediction for test set: [2.42307692 2.87640449 2.87640449 2.42307692 2.87640449
2.87640449
 2.87640449 2.87640449 2.87640449 2.87640449 2.87640449 2.87640449
 2.87640449]
   Actual value  Predicted value
0            1         2.423077
1            3         2.876404
2            2         2.876404
3            3         2.423077
4            3         2.876404
Train error: 0.7352896486229821
Validation error: 1.164717188983856
Test error: 0.588882304526749
```

```
Prediction for test set: [2.81111111 2.48       2.81111111 2.48       2.81111111
2.81111111
 2.81111111 2.81111111 2.81111111 2.81111111 2.81111111 2.48
 2.81111111]
   Actual value  Predicted value
0             4         2.811111
1             2         2.480000
2             3         2.811111
3             1         2.480000
4             3         2.811111
```

[85]:
```
from numpy import mean

print("Train error: {}".format(mean(train_errors)))
print("Validation error: {}".format(mean(val_errors)))
print("Test error: {}".format(mean(test_errors)))
```

```
Train error: 0.7713977220955391
Validation error: 0.8431979440876223
```

Liner regression model and KFold

[71]:
```
train_errors2 = []
val_errors2 = []

# Splitting the data usign KFold
KF2 = KFold(n_splits=10)
for train_index, test_index in KF.split(X):
    X4_train, X4_val = X[train_index], X[test_index]
    y4_train, y4_val = y[train_index], y[test_index]

    lin_regr3 = LinearRegression()
    lin_regr3.fit(X4_train, y4_train)

    y4_train_pred = lin_regr3.predict(X4_train)
    y4_val_pred = lin_regr3.predict(X4_val)
    #meanSqErr2 = mean_squared_error(y3_val, y3_pred)
    #print("MSE: {}".format(meanSqErr2))
    train_err = mean_squared_error(y4_train, y4_train_pred)
    val_err = mean_squared_error(y4_val, y4_val_pred)
    train_errors2.append(train_err)
    val_errors2.append(val_err)
    print("Train error: {}".format(train_err))
    print("Validation error: {}".format(val_err))

    print("Prediction for test set: {}".format(y4_val_pred))
    diff = pd.DataFrame({'Actual value': y4_val, 'Predicted value':␣
↪y4_val_pred})
    print(diff.head())
```

```
Train error: 0.7652944985192811
Validation error: 1.1538445727662965
Prediction for test set: [2.85389028 2.69855901 2.85515583 2.85389028 2.69982456
2.69982456
 2.69729346 2.70109011 2.54196219 2.85389028 2.54322774 2.38916202
 2.38663092 2.85515583 3.01301819]
   Actual value  Predicted value
0             1         2.853890
1             3         2.698559
2             2         2.855156
3             2         2.853890
4             2         2.699825
Train error: 0.803591613336393
Validation error: 0.8126124286287075
Prediction for test set: [2.52756601 2.81615291 2.81615291 2.81615291 2.67185946
2.67185946
 2.81615291 2.96044636 2.52889184 2.95956248 2.52889184 2.52756601
 2.96044636 2.81615291 2.67185946]
   Actual value  Predicted value
0             2         2.527566
1             3         2.816153
2             2         2.816153
3             3         2.816153
4             3         2.671859
Train error: 0.7967474063323695
Validation error: 0.8738694316408484
Prediction for test set: [2.98054909 2.83412495 2.82675116 2.83412495 2.67295323
2.83412495
 2.98054909 2.67295323 2.67295323 2.68770081 2.68032702 2.54127667
 2.51915529 2.68032702 2.68770081]
   Actual value  Predicted value
0             4         2.980549
1             3         2.834125
2             3         2.826751
3             3         2.834125
4             1         2.672953
Train error: 0.822196439376733
Validation error: 0.6556069653661097
Prediction for test set: [2.63894166 2.7915381  2.6595742  2.64925793 2.63894166
2.95445082
 2.7915381  2.62862539 2.95445082 2.46571267 2.95445082 2.62862539
 2.82248691 2.62862539 2.49666148]
   Actual value  Predicted value
```

```
0            3        2.638942
1            2        2.791538
2            3        2.659574
3            2        2.649258
4            2        2.638942
Train error: 0.7941448321364124
Validation error: 0.9480140677567468
Prediction for test set: [3.28930842 2.53839694 2.60497076 2.53839694 2.81399195
2.39594957
 2.88056577 2.46252339 2.60497076 2.81399195 2.7381184  2.53839694
 2.95643932 2.53839694 2.81399195]
   Actual value  Predicted value
0            1        3.289308
1            4        2.538397
2            4        2.604971
3            3        2.538397
4            3        2.813992
Train error: 0.7756928637159026
Validation error: 1.1138590814487705
Prediction for test set: [2.47706433 2.63093067 2.54064959 2.68306059 2.68306059
2.88905684
 2.78605871 2.91449094 2.61821362 2.82420987 2.88905684 2.70849469
 2.60549656 2.61821362]
   Actual value  Predicted value
0            1        2.477064
1            2        2.630931
2            1        2.540650
3            2        2.683061
4            4        2.683061
Train error: 0.8278064008440154
Validation error: 0.6093509351222366
Prediction for test set: [2.40092152 2.63961073 2.85608446 2.52026612 2.80338631
2.75895534
 2.52026612 2.78117082 2.63961073 2.54248161 2.52026612 2.85608446
 2.66182621 2.87829995]
   Actual value  Predicted value
0            2        2.400922
1            2        2.639611
2            4        2.856084
3            2        2.520266
4            2        2.803386
Train error: 0.7777436256794171
Validation error: 1.0706116196038367
Prediction for test set: [2.66747394 2.48444068 2.85234336 2.85234336 2.66747394
2.85188432
 2.48444068 2.48444068 2.66839202 3.03629469 2.48444068 3.03629469
 3.03629469 3.03629469]
   Actual value  Predicted value
```

```
0              4         2.667474
1              1         2.484441
2              3         2.852343
3              4         2.852343
4              2         2.667474
Train error: 0.8176252756891425
Validation error: 0.6992252062954146
Prediction for test set: [2.53775493 2.84816098 2.70204928 2.70204928 3.03063797
2.70204928
 2.88452627 2.68386664 3.03063797 2.17280095 2.88452627 2.51957229
 3.03063797 2.84816098]
   Actual value  Predicted value
0              3         2.537755
1              3         2.848161
2              3         2.702049
3              2         2.702049
4              1         3.030638
Train error: 0.8452067337923989
Validation error: 0.43187629641230857
Prediction for test set: [2.66368899 2.77826107 2.91549171 2.91549171 2.65613614
2.65613614
 2.78581392 2.93059741 2.65613614 2.77826107 2.91549171 2.5189055
 2.52645835 2.93059741]
   Actual value  Predicted value
0              3         2.663689
1              4         2.778261
2              3         2.915492
3              3         2.915492
4              2         2.656136
```

[72]:
```python
print("Train error: {}".format(mean(train_errors2)))
print("Validation error: {}".format(mean(val_errors2)))
```

```
Train error: 0.8026049689422065
Validation error: 0.8368870605041276
```

[ ]: