Exercise4_LinearRegressionExtension

September 24, 2021

1 Linear Regression

1.0.1 Qualitative Features & Interaction Terms

1-A) Use the credit data set, fit OLS linear regression model to predict credit card balance using all the following features

- Student
- Income
- Limit
- Interaction term: Income*Student
- Interaction term: Limit*Student

Find the p-values of all features. Are they all helpful in predicting the response? Why?

[4]:
```
import statsmodels.formula.api as smf
from pandas import read_csv

credit = read_csv('Credit2.csv')

model = smf.ols('Balance ~ Student+Income+Limit+Student*Income+Limit*Student',␣
↪credit)
fitting_results = model.fit()

print(fitting_results.summary().tables[1])
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
Intercept      -415.3863     12.436    -33.401      0.000    -439.836
-390.936
Student[T.Yes]  235.2261     41.256      5.702      0.000     154.117
316.336
Income           -7.6162      0.252    -30.272      0.000      -8.111
-7.122
Student[T.Yes]:Income    -2.5835      0.702     -3.678      0.000      -3.965
```

```
-1.202
Limit                       0.2613      0.004     69.090      0.000        0.254
0.269
Limit:Student[T.Yes]        0.0667      0.012      5.515      0.000        0.043
0.090
==============================================================================
=========
```

(AP): It seems that all of the features are helpful in predicting the response, since all of the p-values are very low. In addition, none of the 95% confidence intervals contain 0.

1-B) Using sklearn library, find the test R^2 score for estimating the balance from features (Income, Limit, StudentEncode) using linear regression model. The StudentEncode is the binary feature that maps Student status to a numerical value ('yes' to 1 and 'No' to 0).

- Set random state to zero in train test split

[5]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

credit = read_csv('Credit2.csv')
credit['StudentEncode'] = credit.Student.map({'No': 0, 'Yes': 1})

x = credit[['Income', 'Limit', 'StudentEncode']]
y = credit['Balance']

X_train, X_test, Y_train, Y_test = train_test_split(x, y, random_state=0)

model = LinearRegression().fit(X_train, Y_train)
pred = model.predict(X_test)

print('R^2 score: %.2f' % r2_score(pred, Y_test))
```

```
R^2 score: 0.95
```

1-C) Repeat the above question after adding to the model the two interaction terms: (1) (Income x StudentEncode) and (2) (Limit x StudentEncode)

[6]:
```
credit['IncomexStudentEncode'] = credit['Income'] * credit['StudentEncode']
credit['LimitxStudentEncode'] = credit['Limit'] * credit['StudentEncode']

x = credit[['Income', 'Limit', 'StudentEncode', 'IncomexStudentEncode',␣
 ↪'LimitxStudentEncode']]
y = credit['Balance']
```

```
X_train, X_test, Y_train, Y_test = train_test_split(x, y, random_state=0)

model = LinearRegression().fit(X_train, Y_train)
pred = model.predict(X_test)

print('R^2 score: %.2f' % r2_score(pred, Y_test))
```

```
      Income  Limit  StudentEncode  IncomexStudentEncode  LimitxStudentEncode
0     14.891   3606              0                 0.000                    0
1    106.025   6645              1               106.025                 6645
2    104.593   7075              0                 0.000                    0
3    148.924   9504              0                 0.000                    0
4     55.882   4897              0                 0.000                    0
..       …    …              …                 …                    …
395   12.096   4100              0                 0.000                    0
396   13.364   3838              0                 0.000                    0
397   57.872   4171              0                 0.000                    0
398   37.728   2525              0                 0.000                    0
399   18.701   5524              0                 0.000                    0

[400 rows x 5 columns]
R^2 score: 0.95
```

1.0.2 Polynomial Regression

Set random_state= 0 in train_test_split in all the questions below.

2-A) Use the Auto dataset,

- (i) Find the test R^2 metric of a linear regression model that predicts the miles per gallon (mpg) from the horsepower.
- (ii) Use polynomial regression to include both the horsepower feature and $(horsepower)^2$ in the regression model to predict the mpg. Find the test R^2 metric in this case

Hint: You can use numpy.concatenate. For example to add to an array U a column vector W^2, we can use X=np.concatenate((U,W**2),axis=1)

[7]:
```
from pandas import read_csv
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import numpy as np

AutoData=read_csv('Auto_modify.csv') # read the data

x = AutoData['horsepower']
y = AutoData['mpg']

X_train, X_test, Y_train, Y_test = train_test_split(x, y, random_state=0)
```

```
X_train = np.array(X_train).reshape(-1, 1)
X_test = np.array(X_train).reshape(-1, 1)
Y_train = np.array(Y_train).reshape(-1, 1)
Y_test = np.array(Y_train).reshape(-1, 1)

model = LinearRegression().fit(X_train, Y_train)
pred = model.predict(X_test)

print('R^2 score: %.2f' % r2_score(Y_test, pred))
```

```
R^2 score: 0.60
```

[8]:
```
AutoData['horsepower_sq'] = AutoData['horsepower'] ** 2

x = AutoData[['horsepower', 'horsepower_sq']]
y = AutoData['mpg']

X_train, X_test, Y_train, Y_test = train_test_split(x, y, random_state=0)

model = LinearRegression().fit(X_train, Y_train)
pred = model.predict(X_test)

print('R^2 score: %.2f' % r2_score(Y_test, pred))
```

```
R^2 score: 0.73
```

2-B) With the same auto dataset, use KNN regression with K=7, to fit a model that predicts miles per gallon(mpg) in the following cases:

- One feature: Horsepower only
- Two features: horsepower and $(horsepower)^2$

Use MinMax feaures scaling. Find the R^2 metric in each of the above cases. Comparing KNN with linear regression, which model performs better? How does the performance change by adding the quadratic feature?

[11]:
```
from sklearn import neighbors
from sklearn import preprocessing

k = 7
knn = neighbors.KNeighborsRegressor(n_neighbors=k)

x = AutoData['horsepower']
y = AutoData['mpg']

X_train, X_test, Y_train, Y_test = train_test_split(x, y, random_state=0)
X_train = np.array(X_train).reshape(-1, 1)
```

```
X_test = np.array(X_test).reshape(-1, 1)

scaler = preprocessing.MinMaxScaler().fit(X_train)

X_train_t = scaler.transform(X_train)
X_test_t = scaler.transform(X_test)

knn.fit(X_train_t, Y_train)
pred = knn.predict(X_test_t)

print('R^2 score: %.2f' % r2_score(Y_test, pred))
```

```
R^2 score: 0.66
```

[10]:
```
x = AutoData[['horsepower', 'horsepower_sq']]
y = AutoData['mpg']

X_train, X_test, Y_train, Y_test = train_test_split(x, y, random_state=0)

scaler = preprocessing.StandardScaler().fit(X_train)

X_train_t = scaler.transform(X_train)
X_test_t = scaler.transform(X_test)

knn.fit(X_train_t, Y_train)
pred = knn.predict(X_test_t)

print('R^2 score: %.2f' % r2_score(Y_test, pred))
```

```
R^2 score: 0.67
```

(AP): KNN seems to be better for this problem than regular linear regression. Adding the quadratic term does not seem to change the R^2 score.

[ ]:
```

```