Data-X Spring 2019: Homework 05

Linear regression & Logistic regression

Name: Sumaanyu Maheshwari

SID: 3031789255

In this homework, you will do some exercises on prediction using sklearn.

REMEMBER TO DISPLAY ALL OUTPUTS. If the question asks you to do something, make sure to print your results.

In [11]:

```
from IPython.display import display, Latex, Markdown
import seaborn as sns
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import zipfile
from pathlib import Path
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import linear_model
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
import re
from sklearn.model_selection import KFold
#from sklearn.cross_validation import KFold
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import MinMaxScaler
```

Part 1 - Regression

Data:

Data Source: Data file is uploaded to bCourses and is named: **Energy.csv** (Link in the Assignment details page on Bcourses)

The dataset was created by Angeliki Xifara (Civil/Structural Engineer) and was processed by Athanasios Tsanas, Oxford Centre for Industrial and Applied Mathematics, University of Oxford, UK).

Data Description:

The dataset contains eight attributes of a building (or features, denoted by X1...X8) and response being the heating load on the building, y1.

- X1 Relative Compactness
- X2 Surface Area
- X3 Wall Area
- X4 Roof Area
- X5 Overall Height
- X6 Orientation
- X7 Glazing Area
- X8 Glazing Area Distribution
- y1 Heating Load

Q1.1

Read the data file from the csv.

Print the count of NaN values for each attribute in the dataset.

Print the Range (min, max) and percentiles (25th, 50th, and 75th) of each attribute in the dataset

In [12]:

```
df = pd.read_csv('Energy.csv')
display(df.head())
print(df.isnull().sum())
(df.describe()).loc[["min", 'max', '25%', '50%', '75%']]
```

	X1	X2	X3	X4	X5	X6	X7	X8	Y1
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84

```
X1    0
X2    0
X3    0
X4    0
X5    0
X6    0
X7    0
X8    0
Y1    0
dtype: int64
```

Out[12]:

	X1	X2	X3	X4	X5	X6	X7	X8	Y1
min	0.6200	514.500	245.0	110.250	3.50	2.00	0.00	0.00	6.0100
max	0.9800	808.500	416.5	220.500	7.00	5.00	0.40	5.00	43.1000
25%	0.6825	606.375	294.0	140.875	3.50	2.75	0.10	1.75	12.9925
50%	0.7500	673.750	318.5	183.750	5.25	3.50	0.25	3.00	18.9500
75%	0.8300	741.125	343.0	220.500	7.00	4.25	0.40	4.00	31.6675

REGRESSION:

Using the data, we want to predict "Heating load". The output variable is continuous. Hence, we need to use a regression algorithm.

Q 1.2:

Split the dataset randomly into train and test. Train a **Linear Regression** model on 80% of the data (80-20 split). What is the intercept and coefficient values?

In [13]:

```
#spt = np.random.rand(len(df)) < 0.8
#train = df[spt]
#test = df[~spt]

X = df[['X1','X2','X3','X4','X5','X6','X7','X8']].values
y = df[['Y1']].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
state = 5)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
print('intercept:', regressor.intercept_)
print('coefficient values:', regressor.coef_)
```

```
intercept: [84.50671242]
coefficient values: [[-6.42688404e+01 -6.28936957e-02  3.67078157e-0
2 -4.98007557e-02
   4.11430910e+00 -1.25900388e-01  1.95157505e+01  1.97512265e-01]]
```

Q.1.3:

Create a function which takes arrays of prediction and actual values of the output as parameters to calculate **'Root Mean Square error'** (RMSE) metric:

1. Use the function to calculate the training RMSE
2. Use the function to calculate the test RMSE

In [14]:

```
#np.sqrt(metrics.mean_squared_error
def RMSE(pred, tar):
    return np.sqrt(((pred - tar) ** 2).mean())

q1_3_a = RMSE(regressor.predict(X_train), y_train)
print(' training RMSE:', q1_3_a)

q1_3_b = RMSE(regressor.predict(X_test), y_test)
print(' test RMSE:', q1_3_b)
```

```
 training RMSE: 2.879398584299829
 test RMSE: 3.0865243377757547
```

Q1.4:

Let's see the effect of amount of data on the performance of prediction model. Use varying amounts of data (100,200,300,400,500,all) from the training data you used previously to train different regression models. Report training error and test error in each case. Test data is the same as above for all these cases.

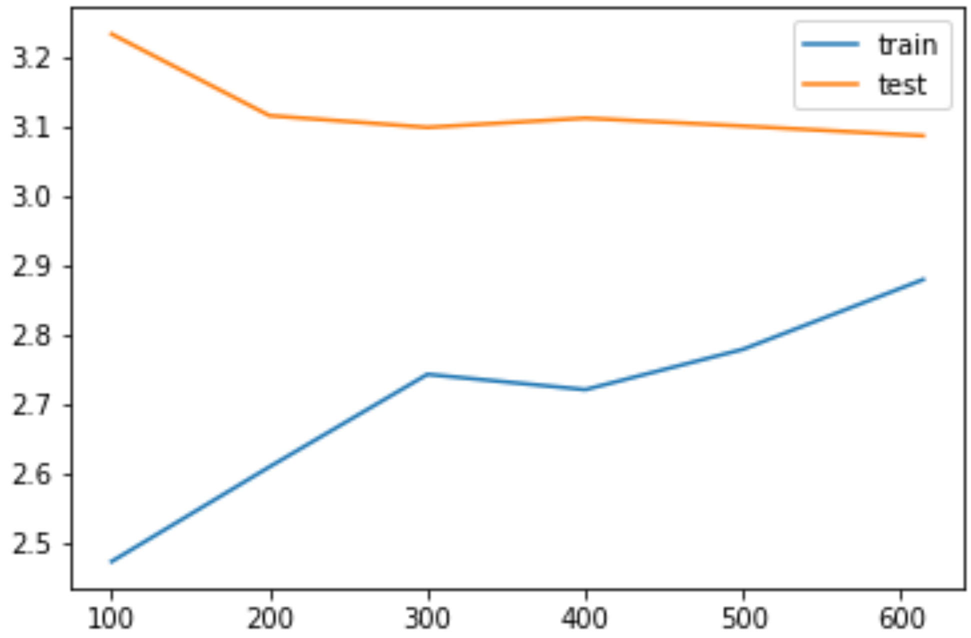
Plot error rates vs number of training examples. Both the training error and the test error should be plotted. Comment on the relationship you observe between the amount of data used to train the model and the test accuracy of the model.

Hint: Use array indexing to choose varying data amounts

In [15]:

```
regressor.fit(X_train[:100], y_train[:100])
xtr1 = RMSE(regressor.predict(X_test), y_test)
x1 = RMSE(regressor.predict(X_train[:100]), y_train[:100])
print("for 100 data points: (Training) ", x1)
print('for 100 data points: (Test)', xtr1)
regressor.fit(X_train[:200], y_train[:200])
x2 = RMSE(regressor.predict(X_train[:200]), y_train[:200])
xtr2 = RMSE(regressor.predict(X_test), y_test)
print("for 200 data points: (Training) ", x2)
print('for 200 data points: (Test)', RMSE(regressor.predict(X_test), y_test))
regressor.fit(X_train[:300], y_train[:300])
x3 = RMSE(regressor.predict(X_train[:300]), y_train[:300])
xtr3 = RMSE(regressor.predict(X_test), y_test)
print("for 300 data points: (Training) ", x3)
print('for 300 data points: (Test)', RMSE(regressor.predict(X_test), y_test))
regressor.fit(X_train[:400], y_train[:400])
x4 = RMSE(regressor.predict(X_train[:400]), y_train[:400])
xtr4 = RMSE(regressor.predict(X_test), y_test)
print("for 400 data points: (Training) ", x4)
print('for 400 data points: (Test)', RMSE(regressor.predict(X_test), y_test))
regressor.fit(X_train[:500], y_train[:500])
x5 = RMSE(regressor.predict(X_train[:500]), y_train[:500])
xtr5 = RMSE(regressor.predict(X_test), y_test)
print("for 500 data points: (Training) ", x5)
print('for 500 data points: (Test)', RMSE(regressor.predict(X_test), y_test))
regressor.fit(X_train, y_train)
xall = RMSE(regressor.predict(X_train), y_train)
xtrall = RMSE(regressor.predict(X_test), y_test)
print("for all data points: (Training) ", xall)
print('for all data points: (Test)', RMSE(regressor.predict(X_test), y_test))
print()
print("the more data I have, my error increases")

value = [x1, x2, x3, x4, x5, xall]
value2 = [xtr1, xtr2, xtr3, xtr4, xtr5, xtrall,]
nums = [100, 200, 300, 400, 500, len(X_train)]



plt.plot(nums, value, label = 'train')
plt.plot(nums, value2, label = 'test')
plt.legend()
plt.show()
```

```
for 100 data points: (Training)  2.4732686300003306
for 100 data points: (Test) 3.232989333096236
for 200 data points: (Training)  2.6096844539978488
for 200 data points: (Test) 3.1154552276926193
for 300 data points: (Training)  2.742879899878274
for 300 data points: (Test) 3.09854855135922
for 400 data points: (Training)  2.7208194053761376
for 400 data points: (Test) 3.1118050044162557
for 500 data points: (Training)  2.77902926307626
for 500 data points: (Test) 3.10044099031993
for all data points: (Training)  2.879398584299829
for all data points: (Test) 3.0865243377757547

the more data I have, my error increases
```



Part 2 - Classification

CLASSIFICATION: LABELS ARE DISCRETE VALUES.

Here the model is trained to classify each instance into a set of predefined discrete classes. On inputting a feature vector into the model, the trained model is able to predict a class of that instance.

Q2.1

Bucket the values of 'y1' i.e 'Heating Load' from the original dataset into 3 classes:

0: 'Low' (< 14),
1: 'Medium' (14-28),
2: 'High' (>28)

HINT: Use pandas.cut

This converts the given dataset into a classification problem. Use this dataset with transformed 'heating load' to create a **logistic regression** classifiction model that predicts heating load type of a building. Split the data randomly into training and test set. Train the model on 80% of the data (80-20 split).

In [16]:

```
df['class'] = pd.cut(df["Y1"], [0,14,28, np.inf], include_lowest = True, labels
= ["Low", 'Medium', 'High'])
df.head()
```

Out[16]:

	X1	X2	X3	X4	X5	X6	X7	X8	Y1	class
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55	Medium
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55	Medium
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55	Medium
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55	Medium
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84	Medium

In [17]:

```
X = df[['X1','X2','X3','X4','X5','X6','X7','X8']]
y = df['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
state = 5)
LogisticRegressionModel = linear_model.LogisticRegression()
LogisticRegressionModel.fit(X_train, y_train)
```

Out[17]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_interce
pt=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jo
bs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0
.0001,
          verbose=0, warm_start=False)
```

Q2.2

- Print the training and test accuracies
- Print the confusion matrix
- Print the precision and recall numbers for all the classes

In [18]:

```
training_accuracy=LogisticRegressionModel.score(X_train,y_train)
print ('Training Accuracy:',training_accuracy)
test_accuracy=LogisticRegressionModel.score(X_test,y_test)
print ('Test Accuracy:',test_accuracy)
print()
y_true = y_test
y_pred = LogisticRegressionModel.predict(X_test)
ConfusionMatrix=pd.DataFrame(confusion_matrix(y_true, y_pred),columns=['Predicte
d Low','Predicted Medium','Predicted High'],index=['Actual Low','Actual Medium',
'Actual High'])
print ('Confusion matrix of test data is: \n',ConfusionMatrix)
print()
print("Average precision for the 3 classes is - ", precision_score(y_true, y_pre
d, average = None))
print()
from sklearn.metrics import recall_score
print("Average recall for the 3 classes is - ", recall_score(y_true, y_pred, ave
rage = None))
```

```
Training Accuracy: 0.8078175895765473
Test Accuracy: 0.7857142857142857

Confusion matrix of test data is:
                Predicted Low  Predicted Medium  Predicted High
Actual Low                56                 0               0
Actual Medium              0                40               1
Actual High               21                11              25

Average precision for the 3 classes is -  [0.72727273 0.78431373 0.9
6153846]

Average recall for the 3 classes is -  [1.         0.97560976 0.4385
9649]
```

Q2.3

K Fold Cross Validation

In k-fold cross-validation, the shuffled training data is partitioned into k disjoint sets and the model is trained on k –1 sets and validated on the kth set. This process is repeated k times with each set chosen as the validation set once. The cross-validation accuracy is reported as the average accuracy of the k iterations

Use 7-fold cross validation on the training data. Print the average accuracy

In [19]:

```
X = df[['X1','X2','X3','X4','X5','X6','X7','X8']]
y = df['class']
F = KFold(n_splits=7, random_state=5, shuffle = True)
LM = linear_model.LogisticRegression()
scores = cross_val_score(LM, X, y, cv = F)
print('Cross-validated scores:', scores)
np.mean(scores)
```

```
Cross-validated scores: [0.80909091 0.77272727 0.7        0.84545455
0.80909091 0.81651376
 0.80733945]
```

Out[19]:

```
0.7943166924818301
```

Q2.4

One of the preprocessing steps in Data science is Feature Scaling i.e getting all our data on the same scale by setting same Min-Max of feature values. This makes training less sensitive to the scale of features . Scaling is important in algorithms that use distance functions as a part of classification. If we Scale features in the range [0,1] it is called unity based normalization.

Perform unity based normalization on the above dataset and train the model again, compare model performance in training and validation with your previous model.

refer:[http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-scaler (http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-scaler)](http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-scaler)
more at: [https://en.wikipedia.org/wiki/Feature_scaling (https://en.wikipedia.org/wiki/Feature_scaling)](https://en.wikipedia.org/wiki/Feature_scaling)

In [20]:

```
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
```

In [21]:

```
y = df['class']
X = df[['X1','X2','X3','X4','X5','X6','X7','X8']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
state = 5)
X_train_minmax = min_max_scaler.fit_transform(X_train)
X_test_minmax = min_max_scaler.fit_transform(X_test)
LogisticRegressionModel = linear_model.LogisticRegression()
LogisticRegressionModel.fit(X_train_minmax, y_train)
training_accuracy=LogisticRegressionModel.score(X_train_minmax,y_train)
print ('Training Accuracy:',training_accuracy)
test_accuracy=LogisticRegressionModel.score(X_test_minmax,y_test)
print ('Test Accuracy:',test_accuracy)
print()
y_true = y_test
y_pred = LogisticRegressionModel.predict(X_test_minmax)
ConfusionMatrix=pd.DataFrame(confusion_matrix(y_true, y_pred),columns=['Predicte
d Low','Predicted Medium','Predicted High'],index=['Actual Low','Actual Medium',
'Actual High'])
print ('Confusion matrix of test data is: \n',ConfusionMatrix)
print()
print("Average precision for the 3 classes is - ", precision_score(y_true, y_pre
d, average = None))
print()
from sklearn.metrics import recall_score
print("Average recall for the 3 classes is - ", recall_score(y_true, y_pred, ave
rage = None))
```

```
Training Accuracy: 0.8061889250814332
Test Accuracy: 0.8311688311688312

Confusion matrix of test data is: 
                Predicted Low  Predicted Medium  Predicted High
Actual Low                56                 0               0
Actual Medium              0                39               2
Actual High               20                 4              33

Average precision for the 3 classes is -  [0.73684211 0.90697674 0.9
4285714]

Average recall for the 3 classes is -  [1.         0.95121951 0.5789
4737]
```

In [26]:

```
print("It works significantly better as compared to q2.2 for the test accuracy b
ut the training accuracy is very slightly worse. It was 0.8078175895765473 for T
raining Accuracy before and 0.7857142857142857 Test Accuracy before." )
"But if I change my random state to say a 100 I cansee my accuracies improve a l
ot. "
```

```
It works significantly better as compared to q2.2 for the test accur
acy but the training accuracy is very slightly worse. It was 0.80781
75895765473 for Training Accuracy before and 0.7857142857142857 Test
Accuracy before.
```

Out[26]:

```
'But if I change my random state to say a 100 I cansee my accuracies
improve a lot. '
```

In [368]:

```
#F = KFold(n_splits=7, random_state=5, shuffle = True)
#LM = linear_model.LogisticRegression()
#scores = cross_val_score(LM, X, y, cv = F)
#print('Cross-validated scores:', scores)
#np.mean(scores)
```

In [ ]:

```

```