Data-X Spring 2019: Homework 05

Linear regression & Logistic regression

Name: McClain Thiel

SID: 3034003600

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.

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 [35]:
```
# your code
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp

df = pd.read_csv('Energy.csv')
df.head()

df.describe()

import warnings
warnings.filterwarnings('ignore')
#spacer
```

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 [36]:
```
# your code
from sklearn.model_selection import train_test_split

#simplifying naming
x1, x2, x3, x4, x5, x6, x7, x8, y = df['X1'],df['X2'],df['X3'],df['X4'],
df['X5'],df['X6'],df['X7'],df['X8'],df['Y1'],

X = np.array([x1,x2,x3,x4,x5,x6,x7,x8])
Y = np.array(y)

X = X.transpose()

x_train, x_test, y_train, y_test= train_test_split(X, y, test_size=0.2,
random_state=100)
```

In [37]:
```
from sklearn import linear_model
model = linear_model.LinearRegression()
print(model.fit(x_train, y_train))
print('The score was ' +  str(model.score(x_test, y_test)))
print('The intercept was ' + str(model.intercept_))
print('The coefficient values are ' + str(model.coef_))
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
The score was 0.9094940669491967
The intercept was 79.13116174147359
The coefficient values are [-6.33926290e+01 -5.86380428e-02  3.46024306
e-02 -4.66202367e-02
  4.36194652e+00  1.81224259e-02  1.98760201e+01  2.19167208e-01]
```

In [ ]:
```
```

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 [38]:
```
# your code

pred1 = model.predict(x_train)
pred2 = model.predict(x_test)

def RMSE(pred, actual):
    s = (sum(pred - actual)**2)/pred.size
    return s**.5

print('RMSE for training data is ', RMSE(pred1, y_train))
print('RMSE for testing data is ', RMSE(pred2, y_test))
```

```
RMSE for training data is  2.523415058415105e-13
RMSE for testing data is  1.2259224747687392
```

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 [39]:
```
# your code
def models(num_examples):
    if num_examples == 'all':
        num_examples = y_train.size

    x_train1 = x_train[0:num_examples][0:]
    y_train1 = y_train[0:num_examples]
    model = linear_model.LinearRegression()
    model.fit(x_train1, y_train1)
    pred1 = model.predict(x_train)
    pred2 = model.predict(x_test)
    print('RMSE for training data is ', RMSE(pred1, y_train), ' for  training on ', num_examples, ' values.')
    print('RMSE for testing data is ', RMSE(pred2, y_test), ' for  training on ', num_examples, ' values.')

models(100)
models(200)
models(300)
models(400)
models(500)
models('all')
```

```
RMSE for training data is  9.109895958244898  for  training on  100  values.
RMSE for testing data is  4.2456857853810845  for  training on  100  values.
RMSE for training data is  0.02942263973160135  for  training on  200  values.
RMSE for testing data is  1.1003920704322423  for  training on  200  values.
RMSE for training data is  2.1921138608752777  for  training on  300  values.
RMSE for testing data is  0.19674948307672882  for  training on  300  values.
RMSE for training data is  2.3951569468145455  for  training on  400  values.
RMSE for testing data is  0.32443641266109935  for  training on  400  values.
RMSE for training data is  0.28545134544544354  for  training on  500  values.
RMSE for testing data is  1.0921559743977671  for  training on  500  values.
RMSE for training data is  2.523415058415105e-13  for  training on  614  values.
RMSE for testing data is  1.2259224747687392  for  training on  614  values.
```

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 [40]:
```
# your code

bins = [0,14,28,100]  #= pd.IntervalIndex.from_tuples([(0,14), (14, 28),
 (28, 100)])
yn = pd.cut(y, bins, labels=["low", "medium", "high"])
Yn = np.array(yn)

x_trainC, x_testC, y_trainC, y_testC= train_test_split(X, Yn, test_size=
0.2, random_state=100)

log = linear_model.LogisticRegression()
log.fit(x_trainC, y_trainC)
```

Out[40]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=
True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          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 [41]:
```
# your code
from sklearn.metrics import confusion_matrix, classification_report

train_ac = log.score(x_trainC, y_trainC)
test_ac = log.score(x_testC, y_testC)

print('Training accuracy is ', train_ac)
print('Testing accuracy is ', test_ac)
predict = log.predict(x_trainC)

print('confustion matrix:')
print(confusion_matrix(y_trainC, predict))
print(classification_report(y_trainC, predict))
```

```
Training accuracy is  0.8061889250814332
Testing accuracy is  0.7727272727272727
confustion matrix:
[[228   0   0]
 [  0 170   5]
 [ 71  43  97]]
              precision    recall  f1-score   support

        high       0.76      1.00      0.87       228
         low       0.80      0.97      0.88       175
      medium       0.95      0.46      0.62       211

   micro avg       0.81      0.81      0.81       614
   macro avg       0.84      0.81      0.79       614
weighted avg       0.84      0.81      0.78       614
```

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 [59]:

```
# your code
from sklearn.model_selection import KFold

kf = KFold(n_splits = 7, shuffle=True)

log = linear_model.LogisticRegression()
train_acc = []
test_acc = []

xTrainM = []
xTestM = []
yTrainM =[]
yTestM =[]

for train_index, test_index in kf.split(X):
    x_trainC2, x_testC2 = X[train_index], X[test_index]
    y_trainC2, y_testC2 = Yn[train_index], Yn[test_index]
    xTrainM.extend(x_trainC2)
    xTestM.extend(x_testC2)
    yTrainM.extend(y_trainC2)
    yTestM.extend(y_testC2)

log.fit(xTrainM, yTrainM)
train_ac = log.score(xTrainM, yTrainM)
test_ac = log.score(xTestM, yTestM)


'''
for train_index, test_index in kf.split(X):
    x_trainC2, x_testC2 = X[train_index], X[test_index]
    y_trainC2, y_testC2 = Yn[train_index], Yn[test_index]
    log.fit(x_trainC2, y_trainC2)
    train_ac = log.score(x_trainC2, y_trainC2)
    test_ac = log.score(x_testC2, y_testC2)
    train_acc.append(train_ac)
    test_acc.append(test_ac)

'''

print('Avg training accuracy: ', train_ac*100, "%")
print('Avg testing accuracy: ', test_ac*100, "%")
```

```
Avg training accuracy:  82.8125 %
Avg testing accuracy:  82.8125 %
```

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 [53]:

```
# your code
from sklearn.preprocessing import normalize
X2 = normalize(X)

x2_trainC, x2_testC, y_trainC, y_testC = train_test_split(X2, Yn, test_s
ize=0.2, random_state=100)

log = linear_model.LogisticRegression()
log.fit(x2_trainC, y_trainC)

train_ac = log.score(x2_trainC, y_trainC)
test_ac = log.score(x2_testC, y_testC)

print('Training accuracy is: ' , train_ac)
print('Testing accuracy is: ', test_ac)
```

```
Training accuracy is:  0.7117263843648208
Testing accuracy is:  0.6493506493506493
```

In [ ]:

```
```