Data-X Spring 2018: Homework 05

Linear regression, logistic regression, matplotlib.

In this homework, you will do some exercises with prediction and plotting.

REMEMBER TO DISLPAY ALL OUTPUTS. If the question asks you to do something, make sure to print your results so we can easily see that you have done it.

Part 1 - Regression

Data:

Data Source: Data file is uploaded to bCourses and is named: **Energy.csv**

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 in python. Check if there are any NaN values, and print the results.

Describe data features in terms of type, distribution range (max and min), and mean values.

Plot feature distributions. This step should give you clues about data sufficiency.

In [3]:
```
import pandas as pd
import matplotlib.pyplot as plt
df=pd.read_csv('/Users/chelseayang/Downloads/Energy.csv')
```

In [4]:
```
df.info() #There are no NaN values as the info shows below
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
X1    768 non-null float64
X2    768 non-null float64
X3    768 non-null float64
X4    768 non-null float64
X5    768 non-null float64
X6    768 non-null int64
X7    768 non-null float64
X8    768 non-null int64
Y1    768 non-null float64
dtypes: float64(7), int64(2)
memory usage: 54.1 KB
```

In [5]:
```
df.describe()
```

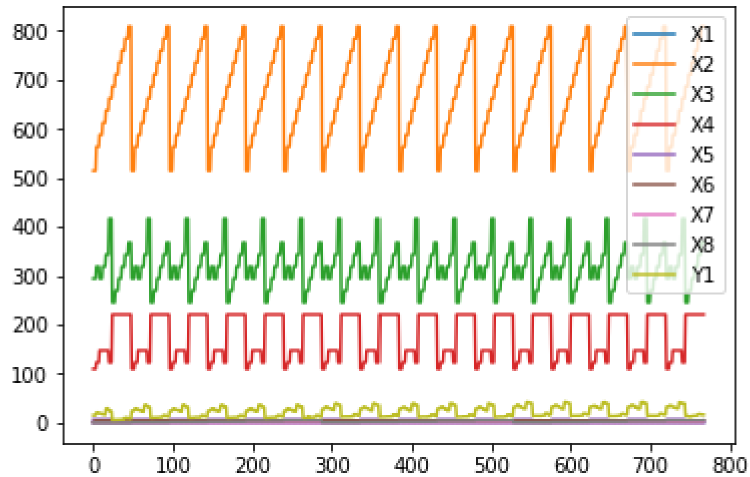
Out[5]:

	X1	X2	X3	X4	X5	X6	X7	X
count	768.000000	768.000000	768.000000	768.000000	768.00000	768.000000	768.000000	768.0000
mean	0.764167	671.708333	318.500000	176.604167	5.25000	3.500000	0.234375	2.8125
std	0.105777	88.086116	43.626481	45.165950	1.75114	1.118763	0.133221	1.5509
min	0.620000	514.500000	245.000000	110.250000	3.50000	2.000000	0.000000	0.0000
25%	0.682500	606.375000	294.000000	140.875000	3.50000	2.750000	0.100000	1.7500
50%	0.750000	673.750000	318.500000	183.750000	5.25000	3.500000	0.250000	3.0000
75%	0.830000	741.125000	343.000000	220.500000	7.00000	4.250000	0.400000	4.0000
max	0.980000	808.500000	416.500000	220.500000	7.00000	5.000000	0.400000	5.0000

In [6]:
```
plt.figure()
df.plot()
```

Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x110c36438>
```

```
<Figure size 432x288 with 0 Axes>
```



REGRESSION: LABELS ARE CONTINUOUS VALUES. Here the model is trained to predict a continuous value for each instance. On inputting a feature vector into the model, the trained model is able to predict a continuous value for that instance.

Q 1.2: Train a linear regression model on 80 percent of the given dataset, what is the intercept value and coefficient values.

In [7]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [8]:
```
X=df.iloc[:,0:7]
```

In [9]:
```
Y=df.iloc[:,8]
```

In [10]:
```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, rar
```

In [11]:
```
reg=LinearRegression()
```

In [12]:
```
reg.fit(x_train,y_train)
```

Out[12]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fal
se)
```

In [13]:
```
print(reg.intercept_)
```

```
78.99610163145373
```

In [14]:
```
print(reg.coef_)
```

```
[-6.31321604e+01 -5.82220359e-02  3.45388922e-02 -4.63804641e-02
  4.37781615e+00  2.27480244e-02  2.05075125e+01]
```

In [15]:
```
y_pred_test=reg.predict(x_test)
y_pred_train=reg.predict(x_train)
```

Q.1.3: Report model performance using 'ROOT MEAN SQUARE' error metric on:

1. Data that was used for training(Training error)
2. On the 20 percent of unseen data (test error)

In [21]:
```
from sklearn import metrics
from math import sqrt
```

In [23]:
```
print('Mean Squared Error of Ttest:', sqrt(metrics.mean_squared_error(y_test

print('Mean Squared Error of Training:', sqrt(metrics.mean_squared_error(y_t
```

```
Mean Squared Error of Ttest: 2.911459751034332
Mean Squared Error of Training: 2.943201027897067
```

Q1.4:

Lets us see the effect of amount of data on the performance of prediction model. Use varying amounts of Training data (100,200,300,400,500,all) to train regression models and report training error and validation error in each case. Validation data/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 validation error should be plotted. Comment on the relationship you observe in the plot, between the amount of data used to train the model and the validation accuracy of the model.

Hint: Use array indexing to choose varying data amounts

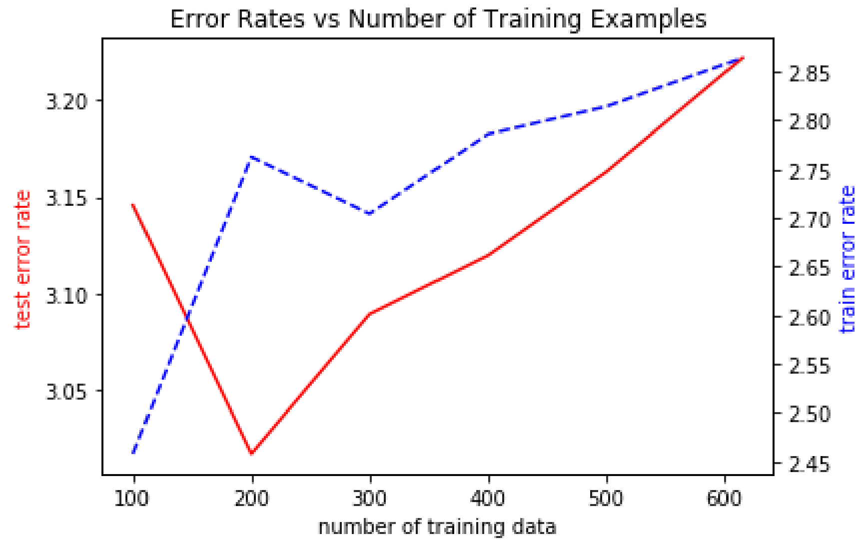
In [39]:
```
import numpy as np
from sklearn.model_selection import cross_val_score
```

In [40]:
```
train_num=np.array([100,200,300,400,500,615])
```

In [67]:
```
size=[]
test_error=[]
train_error=[]

fig, ax1=plt.subplots()

for i in [0,1,2,3,4,5]:
    size.append(1-train_num[i]/768)
    print(size[i])
    x_train2, x_test2, y_train2, y_test2 = train_test_split(X, Y, test_size=
    reg2=LinearRegression()
    reg2.fit(x_train2,y_train2)
    y_pred2_test=reg2.predict(x_test2)
    y_pred2_train=reg2.predict(x_train2)
    test_error.append(sqrt(metrics.mean_squared_error(y_test2, y_pred2_test)
    train_error.append(sqrt(metrics.mean_squared_error(y_train2, y_pred2_tra

print(test_error)
print(train_error)

ax1.set_ylabel('test error rate',color='red')
ax1.set_xlabel('number of training data')

ax1.plot(test_num,test_error,color='red')

ax2=ax1.twinx()

ax2.set_ylabel('train error rate',color='blue')


ax2.plot(test_num,train_error,linestyle='--',color='blue')

plt.title('Error Rates vs Number of Training Examples')
```

```
0.8697916666666666
0.7395833333333333
0.609375
0.47916666666666663
0.34895833333333337
0.19921875
[3.145798976749826, 3.0167535122587608, 3.089126592058018, 3.119661336284
1035, 3.1630888286691703, 3.2220010360754703]
[2.4577398853837518, 2.7627886770630803, 2.704301068700255, 2.78607181906
3714, 2.8143028387203017, 2.8641199145214484]
```

Out[67]:
```
Text(0.5,1,'Error Rates vs Number of Training Examples')
```



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. You can also output the probabilities of an instance belnging to a class.

Q 2.1: Bucket values of 'y1' i.e 'Heating Load' from the original dataset into 3 classes:

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

This converts the given dataset into a classification problem, classes being, Heating load is: *low, medium or high*. Use this datset with transformed 'heating load' for creating a logistic regression classifiction model that predicts heating load type of a building. Use test-train split ratio of 0.8 : 0.2.

Report training and test accuracies and confusion matrices.

HINT: Use pandas.cut

In [70]:
```
import numpy as np
from sklearn import linear_model, datasets
from sklearn.metrics import confusion_matrix

bucket=[0,14,28,max(df['Y1'])]

Y2=pd.cut(df['Y1'],bucket,labels=['0','1','2'])
x_train3, x_test3, y_train3, y_test3 = train_test_split(X, Y2, test_size=0.8

logreg =linear_model.LogisticRegression(C=1e5)

logreg.fit(x_train3,y_train3)
Z=logreg.predict(x_test3)


test_accuracy=logreg.score(x_test3,y_test3)
train_accuracy=logreg.score(x_train3,y_train3)


print('Train accuracy is:', train_accuracy)
print('Test accuracy is:',test_accuracy)

ConfustionMrMatrix=pd.DataFrame(confusion_matrix(y_test3,Z),columns=['Pred0'

print(ConfustionMrMatrix)
```

```
Train accuracy is: 0.9019607843137255
Test accuracy is: 0.8390243902439024
         Pred0  Pred1  Pred2
Actual0    151     16      0
Actual1     16    131     67
Actual2      0      0    234
```

Q2.2: 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 based classification, SVM or K means or those that involve gradient descent optimization. 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 [71]:
```
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler


scaler=MinMaxScaler()
X=scaler.fit_transform(X)

x_train4, x_test4, y_train4, y_test4 = train_test_split(X, Y2, test_size=0.2

logreg =linear_model.LogisticRegression(C=1e5)

logreg.fit(x_train4,y_train4)
Z2=logreg.predict(x_test4)

test_accuracy2=logreg.score(x_test4,y_test4)
train_accuracy2=logreg.score(x_train4,y_train4)


print('Train accuracy is:', train_accuracy2)
print('Test accuracy is:',test_accuracy2)

ConfustionMrMatrix=pd.DataFrame(confusion_matrix(y_test4,Z2),columns=['Pred(

print(ConfustionMrMatrix)
```

```
Train accuracy is: 0.8420195439739414
Test accuracy is: 0.8831168831168831
         Pred0  Pred1  Pred2
Actual0     38      5      0
Actual1      6     43      7
Actual2      0      0     55
```

Part 3 - Matplotlib

Q 3.1a. Create a dataframe called `icecream` that has column `Flavor` with entries `Strawberry`, `Vanilla`, and `Chocolate` and another column with `Price` with entries `3.50`, `3.00`, and `4.25`.

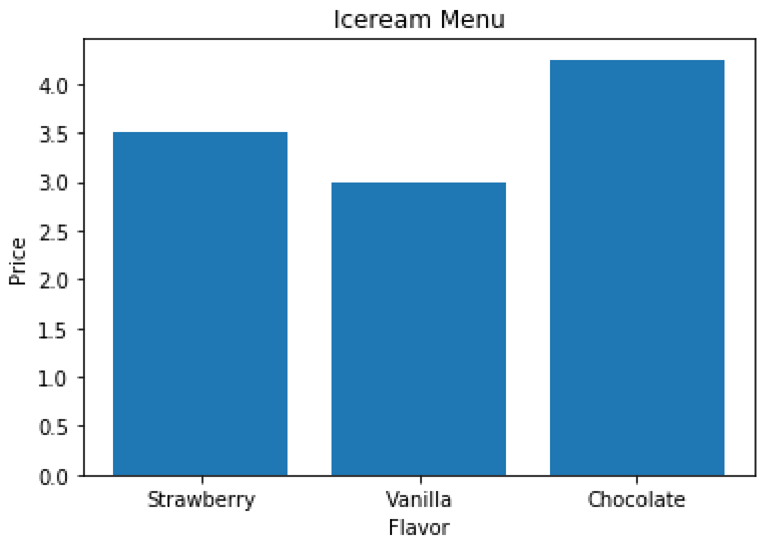
In [72]:
```
d={'Flavor':['Strawberry','Vanilla','Chocolate'],'Price':[3.50,3.00,4.25]}
icecream=pd.DataFrame(data=d)
print(icecream)
```

```
       Flavor  Price
0  Strawberry   3.50
1     Vanilla   3.00
2   Chocolate   4.25
```

Q 3.1b Create a bar chart representing the three flavors and their associated prices.

In [74]:
```
plt.bar(icecream.Flavor,icecream.Price)
plt.title('Icereams Menu')
plt.ylabel('Price')
plt.xlabel('Flavor')
```

Out[74]: Text(0.5,0,'Flavor')



Q 3.2 Create 9 random plots (Hint: There is a numpy function for generating random data). The top three should be scatter plots (one with green dots, one with purple crosses, and one with blue triangles. The middle three graphs should be a line graph, a horizontal bar chart, and a histogram. The bottom three graphs should be trignometric functions (one sin, one cosine, one tangent).

In [75]:
```
np.random.seed(0)
```

In [76]:
```
N=20
x=np.random.randn(N)
y=np.random.randn(N)

x1=np.linspace(0,2*np.pi,51)
y1 = np.sin(x1)
y2 = np.cos(x1)
y3 = np.tan(x1)
```

In [77]:

```
f, ax = plt.subplots(nrows=3,ncols=3)

ax[0,0].scatter(x, y,color='green',marker='o')
ax[0,1].scatter(x, y,color='purple',marker='+')
ax[0,2].scatter(x, y,color='blue',marker='^')


ax[1,0].plot(x, y)
ax[1,1].barh(x, y)
ax[1,2].hist(x,bins=6)

ax[2,0].plot(x1, y1)
ax[2,1].plot(x1, y2)
ax[2,2].plot(x1, y3)
```

Out[77]:

```
[<matplotlib.lines.Line2D at 0x1a23d4cf28>]
```