Data-X Spring 2018: Homework 02

Regression, Classification, Webscraping

Authors: Sana Iqbal (Part 1, 2, 3), Alexander Fred-Ojala (Extra Credit)

In this homework, you will do some exercises with prediction-classification, regression and web-scraping.

Part 1

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

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#import xgboost as xgb
from xgboost import XGBClassifier
from sklearn import linear_model, svm, metrics
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
```

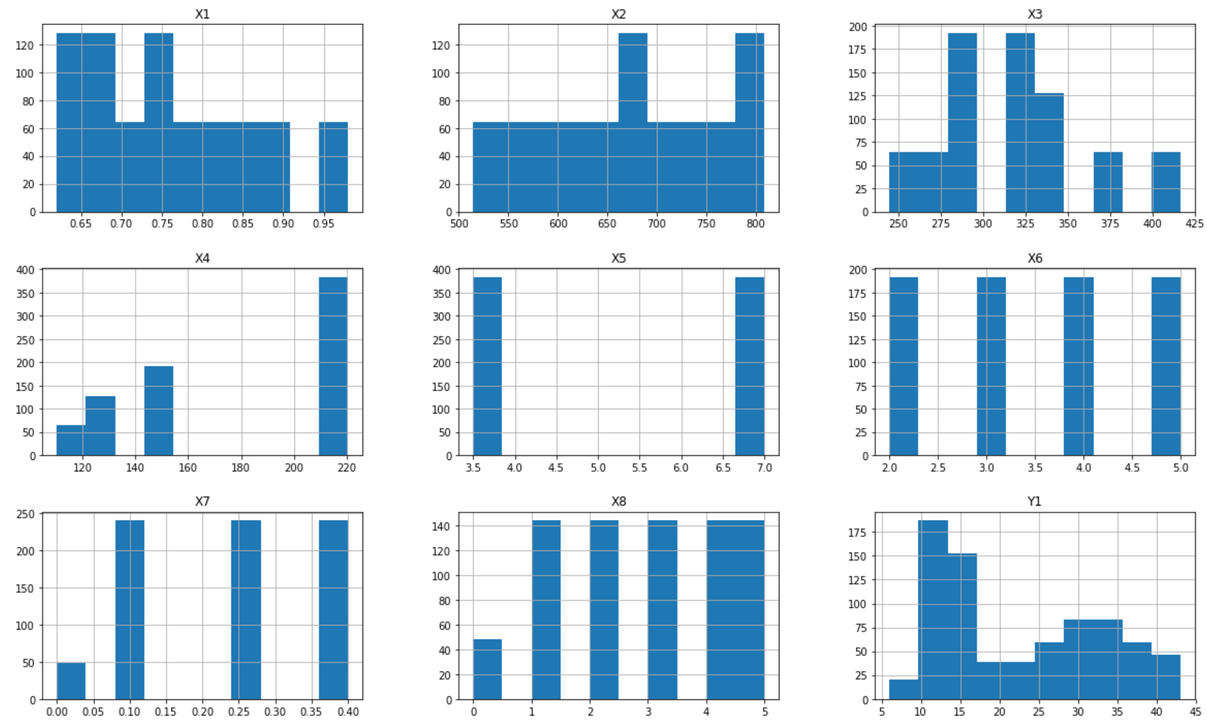
```
/Users/tylerlarsen/miniconda3/lib/python3.6/site-packages/sklearn/cross
_validation.py:41: DeprecationWarning: This module was deprecated in ve
rsion 0.18 in favor of the model_selection module into which all the re
factored classes and functions are moved. Also note that the interface
of the new CV iterators are different from that of this module. This mo
dule will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Q1:Read the data file in python. Describe data features in terms of type, distribution range and mean values. Plot feature distributions.This step should give you clues about data sufficiency.

In [2]:
```
energy = pd.read_csv("energy.csv")
print(energy.describe().loc[['mean', 'min', '25%', '50%', '75%', 'max'],
 :], "\n")
energy.info()
```

```
            X1          X2     X3          X4    X5    X6        X7
  X8  \
mean  0.764167  671.708333  318.5  176.604167  5.25  3.50  0.234375  2.
8125
min   0.620000  514.500000  245.0  110.250000  3.50  2.00  0.000000  0.
0000
25%   0.682500  606.375000  294.0  140.875000  3.50  2.75  0.100000  1.
7500
50%   0.750000  673.750000  318.5  183.750000  5.25  3.50  0.250000  3.
0000
75%   0.830000  741.125000  343.0  220.500000  7.00  4.25  0.400000  4.
0000
max   0.980000  808.500000  416.5  220.500000  7.00  5.00  0.400000  5.
0000

             Y1
mean  22.307201
min    6.010000
25%   12.992500
50%   18.950000
75%   31.667500
max   43.100000

<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 [3]: energy.hist(figsize=(20,12))
        plt.show()
```



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.

Q2.1: Train a linear regression model on 85 percent of the given dataset, what is the intercept value and coefficient values.

```
In [4]: np.random.seed(111)
        x = energy.loc[:, 'X1':'X8']
        y = energy.loc[:, 'Y1']

        X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.15
        )

        lm = linear_model.LinearRegression()
        model = lm.fit(X_train, y_train)
        y_predict = lm.predict(X_test)

        print("Intercept Value: ", lm.intercept_, "\n")
        print("Coefficient Values: \n", lm.coef_)
```

```
Intercept Value:  82.0803137582

Coefficient Values:
 [ -6.45549042e+01  -6.07388384e-02   3.42561940e-02  -4.74975162e-02
   4.32021254e+00   1.48582578e-02   2.03990702e+01   1.48916899e-01]
```

Q.2.2: Report model performance using 'ROOT MEAN SQUARE' error metric on:

1. Data that was used for training(Training error)
2. On the 15 percent of unseen data (test error)

In [5]:
```
def print_errors(X_train, y_train, y_test, y_predict, lm):
    # predicted y_train values are the X_train values * coefficents + in
tercept
    print ("\nRMS Training Error: ", np.sqrt(metrics.mean_squared_error(
y_train, np.matmul(X_train, lm.coef_)+lm.intercept_)))
    print ("RMS Test Error: ", np.sqrt(metrics.mean_squared_error(y_test
, y_predict)))
    print ("R Squared Error: ", model.score(X_test, y_test))

print_errors(X_train, y_train, y_test, y_predict, lm)
```

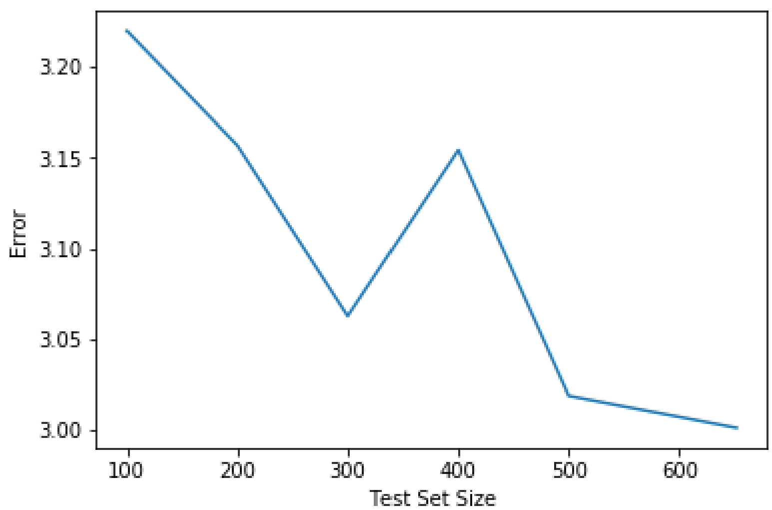
```
RMS Training Error:  2.90912280669
RMS Test Error:  3.00117871383
R Squared Error:  0.902937495014
```

Q2.3: 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.Comment on the relationshipyou 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 [6]:
```
sizes = [100, 200, 300, 400, 500, 652]
rms_errors = []

for size in sizes:

    temp_X_train = X_train.iloc[:size, :]
    temp_y_train = y_train.iloc[:size]

    lm = linear_model.LinearRegression()
    model = lm.fit(temp_X_train, temp_y_train)
    y_predict = lm.predict(X_test)

    # is it correct to be using the test error?
    rms_errors.append(np.sqrt(metrics.mean_squared_error(y_test, y_predi
ct)))

plt.plot(sizes, rms_errors)
plt.xlabel("Test Set Size")
plt.ylabel("Error")
plt.show()
```



Generally, the larger the size of the test set, the lower the error. However, the size of the error actually increased as the training data size increased from 300 to 400, beofre decreasing again.

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

0: 'Low' (< 15),
1: 'Medium' (15-30),
2: 'High' (>30)

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 0.15.

Report training and test accuracies and confusion matrices.

HINT: Use pandas.cut

In [7]:
```
bins = [0, 15, 30, max(energy['Y1'])]
energy['y1_classes'] = pd.cut(energy['Y1'], bins, labels=['0' , '1', '2'
])
energy['y1_classes'].value_counts()
```

Out[7]:
```
0    285
1    282
2    201
Name: y1_classes, dtype: int64
```

In [8]:
```
# logistic regression
np.random.seed(222)
X_train, X_test, y_train, y_test = train_test_split(x, energy['y1_classe
s'], test_size=0.15)

lgr = linear_model.LogisticRegression()
model = lgr.fit(X_train, y_train)
y_predict = lgr.predict(X_test)

print("Intercept Value: ", lm.intercept_, "\n")
print("Coefficient Values: \n", lm.coef_)

print ("\nTraining Error:", metrics.accuracy_score(y_train, lgr.predict(
X_train)))
print ("Testing Error: ", metrics.accuracy_score(y_test, y_predict))
```

```
Intercept Value:  82.0803137582

Coefficient Values:
 [ -6.45549042e+01  -6.07388384e-02   3.42561940e-02  -4.74975162e-02
   4.32021254e+00   1.48582578e-02   2.03990702e+01   1.48916899e-01]

Training Error: 0.773006134969
Testing Error:  0.681034482759
```

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 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)
more at: https://en.wikipedia.org/wiki/Feature_scaling (https://en.wikipedia.org/wiki/Feature_scaling)

In [9]:
```
# why didnt my error change at all?
np.random.seed(333)
scaler = MinMaxScaler(feature_range=(0, 1))
rescaled_x = pd.DataFrame(scaler.fit_transform(x), columns=['X1', 'X2',
'X3', 'X4', 'X5', 'X6', 'X7', 'X8'])

X_train, X_test, y_train, y_test = train_test_split(rescaled_x, y, test_
size=0.15)

lm = linear_model.LinearRegression()
model = lm.fit(X_train, y_train)
y_predict = lm.predict(X_test)

print_errors(X_train, y_train, y_test, y_predict, lm)
```

```
RMS Training Error:  2.84494521585
RMS Test Error:  3.31425962956
R Squared Error:  0.896356783008
```

Part 2

1. Read `diabetesdata.csv` file into a pandas dataframe. Analyze the data features, check for NaN values. About the data:

1. **TimesPregnant**: Number of times pregnant
2. **glucoseLevel**: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. **BP**: Diastolic blood pressure (mm Hg)
4. **insulin**: 2-Hour serum insulin (mu U/ml)
5. **BMI**: Body mass index (weight in kg/(height in m)^2)
6. **pedigree**: Diabetes pedigree function
7. **Age**: Age (years)
8. **IsDiabetic**: 0 if not diabetic or 1 if diabetic)

2. Preprocess data to replace NaN values in a feature(if any) using mean of the feature.
Train logistic regression, SVM, perceptron, kNN, xgboost and random forest models using this preprocessed data with 20% test split.Report training and test accuracies.

In [10]:
```
dbs = pd.read_csv('diabetesdata.csv')
age_wNAs = dbs['Age'].copy()
gl_wNAs = dbs['glucoseLevel'].copy()
print(dbs.isnull().sum())
for col_index in dbs.columns:
    dbs[col_index] = dbs[col_index].fillna(dbs[col_index].mean())

x = dbs.loc[:, 'TimesPregnant':'Age']
y = dbs.loc[:, 'IsDiabetic']
```

```
TimesPregnant     0
glucoseLevel     34
BP                0
insulin           0
BMI               0
Pedigree          0
Age              33
IsDiabetic        0
dtype: int64
```

In [11]:
```
dbs['BP'].dtype
```

Out[11]: dtype('int64')

In [12]:
```
# logistic regression     http://bit.ly/2FarzmU
np.random.seed(444)
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.15
)

lgr = linear_model.LogisticRegression()
model = lgr.fit(X_train, y_train)
y_predict = lgr.predict(X_test)

print ("\nTraining Error:", metrics.accuracy_score(y_train, lgr.predict(
X_train)))
print ("Testing Error: ", metrics.accuracy_score(y_test, y_predict))
```

```
Training Error: 0.759202453988
Testing Error:  0.810344827586
```

In [13]:
```
# SVM     http://bit.ly/2mY6Tqx
np.random.seed(555)
model = svm.SVC(kernel='linear', C=1, gamma=1)
model.fit(X_train, y_train)
y_predict = model.predict(X_test)

print ("\nTraining Error:", metrics.accuracy_score(y_train, model.predic
t(X_train)))
print ("Testing Error: ", metrics.accuracy_score(y_test, y_predict))
```

```
Training Error: 0.768404907975
Testing Error:  0.810344827586
```

In [14]:
```
# perceptron
np.random.seed(666)
ptron = linear_model.Perceptron(max_iter=5)
model = ptron.fit(X_train, y_train)
y_predict = ptron.predict(X_test)

print ("\nTraining Error:", metrics.accuracy_score(y_train, model.predic
t(X_train)))
print ("Testing Error: ", metrics.accuracy_score(y_test, y_predict))
```

```
Training Error: 0.443251533742
Testing Error:  0.344827586207
```

In [15]:
```
# kNN
np.random.seed(777)
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, y_train)
y_predicted= neigh.predict(X_test)

print ("\nTraining Error:", metrics.accuracy_score(y_train, neigh.predic
t(X_train)))
print ("Testing Error: ", metrics.accuracy_score(y_test, y_predict))
```

```
Training Error: 0.831288343558
Testing Error:  0.344827586207
```

In [16]:
```
# xgboost(Extreme Gradient Boosting)     http://bit.ly/2BTQvx0
np.random.seed(888)
model = XGBClassifier()
model.fit(X_train, y_train)

y_predicted= model.predict(X_test)

print ("\nTraining Error:", metrics.accuracy_score(y_train, model.predic
t(X_train)))
print ("Testing Error: ", metrics.accuracy_score(y_test, y_predict))
```

```
Training Error: 0.86963190184
Testing Error:  0.344827586207
```

In [17]:
```
# random forest
np.random.seed(999)
model= RandomForestClassifier(n_estimators=1000)
model.fit(X_train, y_train)
y_predicted= model.predict(X_test)

print ("\nTraining Error:", metrics.accuracy_score(y_train, model.predic
t(X_train)))
print ("Testing Error: ", metrics.accuracy_score(y_test, y_predict))
```

```
Training Error: 1.0
Testing Error:  0.344827586207
```

3. What is the ratio of diabetic persons in 3 equirange bands of 'BMI' and 'Pedigree' in the provided dataset.

Convert these features - 'BP','insulin','BMI' and 'Pedigree' into categorical values by mapping different bands of values of these features to integers 0,1,2.

HINT: USE pd.cut with bin=3 to create 3 bins

In [18]:
```
dbs['BP'].dtype
```

Out[18]:
```
dtype('int64')
```

In [19]:
```
dbs['BP'] = pd.cut(dbs['BP'], bins=3, labels=[0, 1, 2])
dbs['Insulin'] = pd.cut(dbs['insulin'], bins=3, labels=[0, 1, 2])
dbs['BMI'] = pd.cut(dbs['BMI'], bins=3, labels=[0, 1, 2])
dbs['Pedigree'] = pd.cut(dbs['Pedigree'], bins=3, labels=[0, 1, 2])
```

In [20]:
```
ratios = []
ratios.append(dbs.groupby('BMI').sum()['IsDiabetic'] / dbs['IsDiabetic']
.value_counts()[1])
ratios.append(dbs.groupby('Pedigree').sum()['IsDiabetic'] / dbs['IsDiabe
tic'].value_counts()[1])
ratios = pd.DataFrame(ratios)
ratios = ratios.T
ratios.index.name = ''
ratios.columns=['Diabetic_by_BMI', 'Diabetic_by_Pedigree']
ratios.head()
```

Out[20]:

	Diabetic_by_BMI	Diabetic_by_Pedigree
0	0.007463	0.835821
1	0.910448	0.149254
2	0.082090	0.014925

4. Now consider the original dataset again, instead of generalizing the NAN values with the mean of the feature we will try assigning values to NANs based on some hypothesis. For example for age we assume that the relation between BMI and BP of people is a reflection of the age group.We can have 9 types of BMI and BP relations and our aim is to find the median age of each of that group:

Your Age guess matrix will look like this:

BMI	**0**	**1**	**2**
BP			
0	a00	a01	a02
1	a10	a11	a12
2	a20	a21	a22

Create a guess_matrix for NaN values of '*Age*' (using 'BMI' and 'BP') and '*glucoseLevel*' (using 'BP' and 'Pedigree') for the given dataset and assign values accordingly to the NaNs in 'Age' or '*glucoseLevel*' .

Refer to how we guessed age in the titanic notebook in the class.

In [21]:
```
dbs.head(1)
```

Out[21]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	Insulin
0	6	148.0	1	0	1	0	50.0	1	0

In [22]:
```
dbs['Age'] = age_wNAs
dbs['glucoseLevel'] = gl_wNAs
age_guess = dbs.pivot_table(index = 'BP', columns= 'BMI', values= 'Age')
glucose_guess = dbs.pivot_table(index = 'Pedigree', columns= 'BP', value
s= 'glucoseLevel')
print(age_guess)
print(glucose_guess)
```

```
BMI          0          1          2
BP
0    29.000000  30.642857  33.000000
1    28.857143  32.325773  34.823529
2    49.750000  38.384058  32.812500
BP                 0           1           2
Pedigree
0         115.054054  117.201245  132.365672
1         127.500000  122.100000  133.800000
2         137.000000  141.500000  159.500000
```

In [23]:
```
dbs.isnull().sum()
dbs['BP'].value_counts()
```

Out[23]:
```
1    563
2    165
0     40
Name: BP, dtype: int64
```

In [24]:
```
type(dbs['BP'][1])
```

Out[24]:
```
numpy.int64
```

In [25]:
```
# Why wont the nulls get replaced??

#Let's replace Age NAN values with the appropriate age_guess matrix valu
es
for i in range(0, 3):
    for j in range(0, 3):
        dbs.loc[(dbs['Age'].isnull()) & (dbs['BP'] == i) \
                      & (dbs['BMI'] == j),'Age'] = age_guess.iloc[i,j]

#Let's replace glucoseLevel NAN values with the appropriate glucose_gues
s matrix values
for i in range(0, 3):
    for j in range(0, 3):
        dbs.loc[(dbs['glucoseLevel'].isnull()) & (dbs['Pedigree'] == i)
 \
                      & (dbs['BP'] == j),'glucoseLevel'] = glucose_guess.i
loc[i,j]
```

In [26]:
```
dbs.isnull().sum()
```

Out[26]:
```
TimesPregnant    0
glucoseLevel     0
BP               0
insulin          0
BMI              0
Pedigree         0
Age              0
IsDiabetic       0
Insulin          0
dtype: int64
```

5. Now, convert 'glucoseLevel' and 'Age' features also to categorical variables of 5 categories each.

Use this dataset (with all features in categorical form) to train perceptron, logistic regression and random forest models using 20% test split. Report training and test accuracies.

In [27]:
```
dbs['glucoseLevel'] = pd.cut(dbs['glucoseLevel'], bins=5, labels=['0' ,
'1', '2', '3', '4'])
dbs['Age'] = pd.cut(dbs['Age'], bins=5, labels=['0' , '1', '2', '3', '4'
])
```

In [28]:
```
x = dbs.loc[:, 'TimesPregnant':'Age']
y = dbs.loc[:, 'IsDiabetic']
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.20
)
```

In [29]:
```
# perceptron
np.random.seed(111)
ptron = linear_model.Perceptron(max_iter=5)
model = ptron.fit(X_train, y_train)
y_predict = ptron.predict(X_test)

print ("\nTraining Error:", metrics.accuracy_score(y_train, model.predic
t(X_train)))
print ("Testing Error: ", metrics.accuracy_score(y_test, y_predict))
```

```
Training Error: 0.548859934853
Testing Error:  0.564935064935
```

In [30]:
```
# logistic regression     http://bit.ly/2FarzmU
np.random.seed(222)

lgr = linear_model.LogisticRegression()
model = lgr.fit(X_train, y_train)
y_predict = lgr.predict(X_test)

print ("\nTraining Error:", metrics.accuracy_score(y_train, lgr.predict(
X_train)))
print ("Testing Error: ", metrics.accuracy_score(y_test, y_predict))
```

```
Training Error: 0.762214983713
Testing Error:  0.766233766234
```

In [31]:
```
# random forest
np.random.seed(333)
model= RandomForestClassifier(n_estimators=1000)
model.fit(X_train, y_train)
y_predicted= model.predict(X_test)

print ("\nTraining Error:", metrics.accuracy_score(y_train, model.predic
t(X_train)))
print ("Testing Error: ", metrics.accuracy_score(y_test, y_predict))
```

```
Training Error: 0.957654723127
Testing Error:  0.766233766234
```

Part 3

1. **Derive the expression for the optimal parameters in the linear regression equation, i.e. solve the normal equation for Ordinary Least Squares for the case of Simple Linear Regression, when we only have one input and one output**

Given a set of *n* points (X_i, Y_i) where Yi is dependent on Xi by a linear relation, find the best-fit line,

$$Z_i = aX_i + b$$

that minimizes the **sum of squared errors in Y**,i.e:

$$minimize \sum_i (Y_i - Z_i)^2$$

i. Show that

$$intercept \quad b = \overline{Y} - a.\overline{X} \quad and \quad slope \quad a = \frac{\sum_i (X_i - \overline{X})(Y_i - \overline{Y})^2}{\sum_i (X_i - \overline{X})}$$

where $\overline{X}$ and $\overline{Y}$ are the averages of the X values and the Y values, respectively.

ii. Show that slope *a* can be written as $a = r.(S_y/S_x)$ where S_y = the standard deviation of the Y values and S_x = the standard deviation of the X values and *r* is the correlation coefficient.

Please try to write a nice LateXed version of your answer, and do the derivations of the expressions as nicely as possible

i) The goal is to minimize this function:

$$S = min \sum_i (Y_i - Z_i)^2$$

where

$$Z_i = aX_i + b$$

Which equals

$$S = min \sum_i (Y_i - (aX_i + b))^2$$

$$S = min \sum_i (Y_i^2 - 2Y_i(aX_i + b) + (aX_i + b)^2)$$

Next, we take the derivative and set S equal to 0 to find our minimums.

$$\frac{\partial S}{\partial b} = \frac{\partial}{\partial b}(\sum_i Y_i^2 - \sum_i 2Y_i(aX_i + b) + \sum_i (aX_i + b)^2) = 0$$

$$\frac{\partial S}{\partial b} = \frac{\partial}{\partial b}(-2b \sum_i Y_i + 2ab \sum_i X_i + nb^2) = 0$$

$$\frac{\partial S}{\partial b} = -2 \sum_i Y_i + 2a \sum_i X_i + 2nb) = 0$$

$$nb = \sum_i Y_i - a \sum_i X_i$$

Note $\sum_i Y_i = n\overline{Y}$, (1) so after dividing by n we can rewrite the expression as:

$$b = \overline{Y} - a\overline{X}$$

Next we can derive with respect to a:

$$\frac{\partial S}{\partial a} = \frac{\partial}{\partial a}(\sum_i Y_i^2 - \sum_i 2Y_i(aX_i + b) + \sum_i (aX_i + b)^2) = 0$$

$$\frac{\partial S}{\partial a} = \frac{\partial}{\partial a}(-2a \sum_i Y_iX_i + a^2 \sum_i X_i^2 + 2ab \sum_i X_i) = 0$$

$$\frac{\partial S}{\partial a} = -2 \sum_i Y_iX_i + 2a \sum_i X_i^2 + 2b \sum_i X_i = 0$$

Next we can susbstite $b = \bar{Y} - a\bar{X}$:

$$\frac{\partial S}{\partial a} = -2 \sum_i Y_iX_i + 2a \sum_i X_i^2 + 2(\bar{Y} - a\bar{X}) \sum_i X_i = 0$$

$$\frac{\partial S}{\partial a} = -2 \sum_i Y_iX_i + 2a \sum_i X_i^2 + 2\bar{Y} \sum_i X_i - 2a\bar{X} \sum_i X_i = 0$$

$$\frac{\partial S}{\partial a} = - \sum_i Y_iX_i + \overline{Y} \sum_i X_i + a(\sum_i X_i^2 - \overline{X} \sum_i X_i) = 0$$

Solving for a we get:

$$a = \frac{\sum_i Y_iX_i - \overline{Y} \sum_i X_i}{\sum_i X_i^2 - \overline{X} \sum_i X_i}$$

From (1) we can make the following subsitution:

$$a = \frac{\sum_i Y_i X_i - n\overline{Y}\overline{X}}{\sum_i X_i^2 - n\overline{X}^2}$$

Substituting
$\sum_i (X_i - \overline{X})(Y_i - \overline{Y})$for $\sum_i Y_i X_i - n\overline{Y}\overline{X}$

and $\sum_i (X_i - \overline{X})^2$ for $\sum_i X_i^2 - n\overline{X}^2$.

We are left with:

$$a = \frac{\sum_i (X_i - \overline{X})(Y_i - \overline{Y})}{\sum_i (X_i - \overline{X})^2}$$

ii) Start by setting the slope equal to $r\frac{S_y}{S_x}$

$$a = \frac{\sum_i (X_i - \overline{X})(Y_i - \overline{Y})}{\sum_i (X_i - \overline{X})^2} = r\frac{S_y}{S_x}$$

We know that

$$r = \frac{cov(x, y)}{S_x S_Y}$$

so we can write the following:

$$\frac{\sum_i (X_i - \overline{X})(Y_i - \overline{Y})}{\sum_i (X_i - \overline{X})^2} = \frac{cov(x, y)}{S_x S_Y}\frac{S_y}{S_x}$$

$$\frac{\sum_i (X_i - \overline{X})(Y_i - \overline{Y})}{\sum_i (X_i - \overline{X})^2} = \frac{cov(x, y)}{var(S)}$$

We also know that

$$var(S) = \sum_i (X_i - \overline{X})^2$$

and

$$cov(x, y) = \sum_i (X_i - \overline{X})(Y_i - \overline{Y})$$

, So we can write the following:

$$\frac{\sum_i (X_i - \overline{X})(Y_i - \overline{Y})}{\sum_i (X_i - \overline{X})^2} = \frac{\sum_i (X_i - \overline{X})(Y_i - \overline{Y})}{\sum_i (X_i - \overline{X})^2}$$

Done!

Two Extra Credit Points: Fun with Webscraping & Text manipulation

(Mandatory for Grad students!)

`NOTE:` **If you are a Graduate Section student (enrolled in 290), the Extra Credit Questions are mandatory.**

1. Statistics in Presidential Debates

Your first task is to scrape Presidential Debates from the Commission of Presidential Debates website: http://www.debates.org/index.php?page=debate-transcripts (http://www.debates.org/index.php?page=debate-transcripts).

To do this, you are not allowed to manually look up the URLs that you need, instead you have to scrape them. The root url to be scraped is the one listed above, namely: http://www.presidency.ucsb.edu/debates.php (http://www.presidency.ucsb.edu/debates.php)

1. By using `requests` and `BeautifulSoup` find all the links / URLs on the website that links to transcriptions of **First Presidential Debates** from the years [2012, 2008, 2004, 2000, 1996, 1988, 1984, 1976, 1960]. In total you should find 9 links / URLs tat fulfill this criteria.
2. When you have a list of the URLs your task is to create a Data Frame with some statistics (see example of output below):
   A. Scrape the title of each link and use that as the column name in your Data Frame.
   B. Count how long the transcript of the debate is (as in the number of characters in transcription string). Feel free to include `\` characters in your count, but remove any breakline characters, i.e. `\n`. You will get credit if your count is +/- 10% from our result.
   C. Count how many times the word **war** was used in the different debates. Note that you have to convert the text in a smart way (to not count the word **warranty** for example, but counting **war.**, **war!**, **war,** or **War** etc.
   D. Also scrape the most common used word in the debate, and write how many times it was used. Note that you have to use the same strategy as in 3 in order to do this.

Tips:

In order to solve question 3 and 4 above it can be useful to work with Regular Expressions and explore methods on strings like `.strip(), .replace(), .find(), .count(), .lower()` etc. Both are very powerful tools to do string processing in Python. To count common words for example I used a `Counter` object and a Regular expression pattern for only words, see example:

```
from collections import Counter
    import re

    counts = Counter(re.findall(r"[\w']+", text.lower()))
```

Read more about Regular Expressions here: https://docs.python.org/3/howto/regex.html (https://docs.python.org/3/howto/regex.html)

Example output of all of the answers to EC Question 1:

	October 3, 2012: The First Obama-Romney Presidential Debate	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Debate char length	94627	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
war_count	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
most_common_w	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

In [32]:
```
import requests
import bs4 as bs
import re
```

In [33]:
```
source = requests.get("http://www.debates.org/index.php?page=debate-tran
scripts",)
# a GET request will download the HTML webpage.

print(source) # If <Response [200]> then
# the website has been downloaded succesfully
```

```
<Response [200]>
```

In [34]:
```
data = bs.BeautifulSoup(source.content, features='html.parser')
```

In [35]:
```
links = []
for item in data.find_all("p", text=re.compile('The First')):
    for link in item.find_all('a'):
        links += [link.get('href')]

#the 1992 debate css was formatted inconsistantly, and so had to be extr
acted seperately
for item in data.find_all(title="October 19, 1992 Debate Transcript"):
        links += [item.get('href')]

links
```

Out[35]:
```
['http://www.debates.org/index.php?page=october-3-2012-debate-transcrip
t',
 'http://www.debates.org/index.php?page=2008-debate-transcript',
 'http://www.debates.org/index.php?page=september-30-2004-debate-transc
ript',
 'http://www.debates.org/index.php?page=october-3-2000-transcript',
 'http://www.debates.org/index.php?page=october-6-1996-debate-transcrip
t',
 'http://www.debates.org/index.php?page=september-25-1988-debate-transc
ript',
 'http://www.debates.org/index.php?page=october-7-1984-debate-transcrip
t',
 'http://www.debates.org/index.php?page=september-26-1960-debate-transc
ript',
 'http://www.debates.org/index.php?page=october-19-1992-debate-transcri
pt']
```

2. Download and read in specific line from many data sets

Scrape the first 27 data sets from this URL http://people.sc.fsu.edu/~jburkardt/datasets/regression/ (http://people.sc.fsu.edu/~jburkardt/datasets/regression/) (i.e. `x01.txt` - `x27.txt`). Then, save the 5th line in each data set, this should be the name of the data set author (get rid of the # symbol, the white spaces and the comma at the end).

Count how many times (with a Python function) each author is the reference for one of the 27 data sets. Showcase your results, sorted, with the most common author name first and how many times he appeared in data sets. Use a Pandas DataFrame to show your results, see example.

Example output of the answer EC Question 2:

	Counts
Authors	
Helmut Spaeth	[redacted]
[redacted]	3
[redacted]	2
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]