Class Final Project: NYC Taxi duration

In [1]:

```
# imports

import numpy as np
import pandas as pd

from sklearn.metrics import accuracy_score
from sklearn.cross_validation import train_test_split
from sklearn.cross_validation import cross_val_score
from sklearn import metrics
```

```
/Users/thp/anaconda3/lib/python3.6/site-packages/sklearn/cross_validati
on.py:41: DeprecationWarning: This module was deprecated in version 0.1
8 in favor of the model_selection module into which all the refactored
classes and functions are moved. Also note that the interface of the ne
w CV iterators are different from that of this module. This module will
be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Import the training dataset and start preprocessing the data

In [73]:

```
# import the dataset and check
taxi = pd.read_csv('/Users/thp/Documents/CSULA/5661 Data Science/Projec
t/train.csv')
print(taxi.shape)
taxi.head()
```

```
(1458644, 11)
```

Out[73]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_lo
0	id2875421	2	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.98215
1	id2377394	1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.98041
2	id3858529	2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.97902
3	id3504673	2	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.01004
4	id2181028	2	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.97305

in this data set, we found out that some columns are not necessary for the prediction.

For example, "id" and "vendor_id" and "store_and_fwd_flag" column, so we first drop those.

In [74]:
```
taxi_new = taxi.drop(['id','vendor_id','store_and_fwd_flag'],axis=1)
taxi_new.head()
```

Out[74]:

	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latituc
0	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937
1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564
2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939
3	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971
4	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209

In further observation, we found that Pick-up time - Drop-off time = trip duration,

so if we just need to find out what time the taxi pick people up, we can omit the drop-off column.

To easily capture the hour results, use the following code:

In [75]:
```python
taxi_new_h = pd.to_datetime(taxi_new["pickup_datetime"])
taxi_new['hour'] = taxi_new_h.map(lambda x: x.hour)
taxi_new.head()
```

Out[75]:

	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latituc
0	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937
1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564
2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939
3	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971
4	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209

we can see that we have a column 'hour' that captures the hour of the day when the taxi picks up the customer.

we believe that this timing of the ride is very important.

To avoid numeric relationship, we categorize the hour of a day by 5 differnt time zones:

In [14]:
```python
def cateHours(x):
    if 0 <= x <= 4:
        return "EM"    # Early Morning
    elif  5<= x <= 11:
        return "MP"    # Morning Peak, when everybody is driving to work and school
    elif 12 <= x <= 14:
        return "AF"    # Afternoon chill time
    elif 15 <= x <= 20:
        return "AP"   # AFternoon Peak, when people are going home from work and school
    elif 21 <= x <= 24:
        return "LN"   # Late Night
```

Then we apply this function to the dataset to change the 'hour' column to discrete values:

In [15]:
```
taxi_new['hour'] = taxi_new['hour'].apply(cateHours)
taxi_new.head()
```

Out[15]:

	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latituc
0	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937
1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564
2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939
3	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971
4	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209

Then we use OneHot Encoding for column 'hour'

In [64]:
```
# One hot encoding

taxi_new_onehotHour = pd.get_dummies(taxi_new['hour'])
taxi_new_onehotHour.head()
```

Out[64]:

	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22	23
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	1	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 24 columns

In [65]:
```
# put this into the dataset

taxi_new = pd.concat([taxi_new,taxi_new_onehotHour], axis=1)
taxi_new.head()
```

Out[65]:

	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latituc
0	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937
1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564
2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939
3	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971
4	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209

5 rows × 33 columns

After getting the pick-up time zone, we can drop the pickup_datetime and dropoff_datetime

but before we do that, let make the Label first.

In [66]:
```
label = taxi_new['trip_duration']
label.shape
```

Out[66]: (1458644,)

then drop all unnecessary columns, and make the feature matrix

In [76]:
```
taxi_new = taxi_new.drop(['pickup_datetime','dropoff_datetime','trip_dur
ation'],axis=1)
taxi_new.head()
```

Out[76]:

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitud
0	1	-73.982155	40.767937	-73.964630	40.765602
1	1	-73.980415	40.738564	-73.999481	40.731152
2	1	-73.979027	40.763939	-74.005333	40.710087
3	1	-74.010040	40.719971	-74.012268	40.706718
4	1	-73.973053	40.793209	-73.972923	40.782520

Scaling may help to normalize the data:

In [32]:
```
from sklearn import preprocessing
taxi_scaled = preprocessing.scale(taxi_new)
taxi_new = pd.DataFrame(taxi_scaled)
taxi_new.head()
```

Out[32]:

	0	1	2	3	4	5	6	7
0	-0.505637	-0.122261	0.517494	0.124369	0.384575	-0.418775	1.433246	-0.404068
1	-0.505637	-0.097727	-0.375819	-0.368970	-0.575303	-0.418775	-0.697717	2.474831
2	-0.505637	-0.078143	0.395910	-0.451805	-1.162220	-0.418775	-0.697717	-0.404068
3	-0.505637	-0.515558	-0.941274	-0.549976	-1.256071	-0.418775	1.433246	-0.404068
4	-0.505637	0.006112	1.286091	0.006974	0.855957	2.387919	-0.697717	-0.404068

Algorithm selection:

We select Linear Regression, Lasso, ElasticNet, and Ridge to perform regression, and compare them together

In [36]:
```
# Frist define the RMSE method.

def RMSE(y_test,y_predict):
    # Calculating "Mean Square Error" (MSE):
    mse = metrics.mean_squared_error(y_test, y_predict)
    # Using numpy sqrt function to take the square root and calculate "R
oot Mean Square Error" (RMSE)
    rmse = np.sqrt(mse)
    return(rmse)
```

In [42]:

```
# Define a method to run all 4 regression at the same time:

def Regressions(feature,label):

    # split the dataset into training and testing sets by 80-20 ratio
    X_train, X_test, y_train, y_test = train_test_split(feature, label,
test_size=0.2, random_state=3)


    #linear
    from sklearn.linear_model import LinearRegression
    myLinearReg = LinearRegression()
    myLinearReg.fit(X_train,y_train)
    y_predict = myLinearReg.predict(X_test)
    print('Linear ',RMSE(y_test,y_predict))

    y_predict = myLinearReg.predict(X_train)
    print('Linear Train',RMSE(y_train,y_predict),'\n')    ## Output the
 RMSE on the Training set.

    #Ridge
    from sklearn.linear_model import Ridge
    myRidge = Ridge()
    myRidge.fit(X_train,y_train)
    y_predict = myRidge.predict(X_test)
    print('Ridge ', RMSE(y_test,y_predict))

    y_predict = myRidge.predict(X_train)
    print('Ridge Train', RMSE(y_train,y_predict),'\n')    ## Output the
 RMSE on the Training set.

    #ElasticNet
    from sklearn.linear_model import ElasticNet
    myENet = ElasticNet()
    myENet.fit(X_train,y_train)
    y_predict = myENet.predict(X_test)
    print('ElasticNet ', RMSE(y_test,y_predict))

    y_predict = myENet.predict(X_train)
    print('ElasticNet Train', RMSE(y_train,y_predict),'\n')

    #Lasso
    from sklearn.linear_model import Lasso
    myLasso = Lasso()
    myLasso.fit(X_train,y_train)
    y_predict = myLasso.predict(X_test)
    print('Lasso ', RMSE(y_test,y_predict))

    y_predict = myLasso.predict(X_train)
    print('Lasso Train', RMSE(y_train,y_predict),'\n')


    # 10-fold Cross validation:
    rmse_list = cross_val_score(myLasso, X_train, y_train, cv=10, scorin
g='neg_mean_squared_error')
    #print(rmse_list)
```

```
    # Notice that "cross_val_score" by default provides "negative" value
s for "mse" to clarify that mse is error.
    # in order to calculate root mean square error (rmse), we have to ma
ke them positive!
    mse_list_positive = -rmse_list

    # using numpy sqrt function to calculate rmse:
    rmse_list = np.sqrt(mse_list_positive)
    #print(rmse_list)

    print('cross-validation',rmse_list.mean())
```

Also, in order to reduce work load and compare result, we create a method to split the dataset

In [33]:
```
def shrinkDataSet (train,label,times, splitSize):   #times= how many tim
es to split the original dataset
    X_train = train
    y_train = label
    for i in range (0,times):
        X_train, X_test, y_train, y_test = train_test_split(X_train, y_t
rain, test_size=splitSize, random_state=3)
    return X_train, y_train
```

Split the dataset to make it smaller:

In [61]:
```
# use spliting 5 times as example.

taxi_reduced, label_reduced = shrinkDataSet (taxi_new,label,5,0.5)    #
 split 5 times, into half

print('original: ',taxi_new.shape)
print('After: ',taxi_reduced.shape)
```

```
original:  (1458644, 10)
After:  (45582, 10)
```

start predicting:

First, Define a method for easy comparing:

In [77]:

```
def Run_compare(k):        # K = how many times to split

    taxi_reduced, label_reduced = shrinkDataSet (taxi_new,label, k ,0.5)
# split k times, into half
    print('Original shape: ',taxi_new.shape)
    print('After shape: ',taxi_reduced.shape,'\n')

    Regressions(taxi_reduced,label_reduced)
```

In [78]:

```
# with k=8

k=8

Run_compare(k)
```

```
Original shape:  (1458644, 6)
After shape:  (5697, 6)

Linear  1781.51831131
Linear Train 2877.84713674

Ridge  1784.69808899
Ridge Train 2878.11171685

ElasticNet  1837.5547643
ElasticNet Train 2892.35522215

Lasso  1788.45754825
Lasso Train 2878.39847003

cross-validation 2132.3395566
```

In [79]:

```
# with k=5

k=5

Run_compare(k)
```

```
Original shape:  (1458644, 6)
After shape:  (45582, 6)

Linear  2404.81766244
Linear Train 3173.99831691

Ridge  2404.76544278
Ridge Train 3173.99949758

ElasticNet  2409.92230982
ElasticNet Train 3177.59270167

Lasso  2404.79612419
Lasso Train 3174.20953396

cross-validation 3201.01362113
```

In [80]:

```
# with k=3

k=3

Run_compare(k)
```

```
Original shape:  (1458644, 6)
After shape:  (182330, 6)

Linear  3156.34673057
Linear Train 3031.27338788

Ridge  3156.3498263
Ridge Train 3031.27348151

ElasticNet  3164.61795738
ElasticNet Train 3037.9264342

Lasso  3157.00275639
Lasso Train 3031.50439215

cross-validation 3020.4657568
```

In [81]:

```
# with k=1

k=1

Run_compare(k)
```

```
Original shape:  (1458644, 6)
After shape:  (729322, 6)

Linear  3164.22500312
Linear Train 6845.44324993

Ridge  3164.22268921
Ridge Train 6845.44325583

ElasticNet  3172.01889884
ElasticNet Train 6849.21676245

Lasso  3164.36450035
Lasso Train 6845.64709536

cross-validation 5315.64243849
```

In [82]:
```
# with k=0 , not spliting at all

k=0

Run_compare(k)
```

```
Original shape:  (1458644, 6)
After shape:  (1458644, 6)

Linear  4792.48485427
Linear Train 5337.21470879

Ridge  4792.48477267
Ridge Train 5337.21470982

ElasticNet  4795.97657707
ElasticNet Train 5341.76515078

Lasso  4792.57522921
Lasso Train 5337.38714826

cross-validation 4483.48686124
```

From article, if we compare the RMSE from the Predicted RMSE and the Training RMSE, and if they are similar, it's good.

RMSE doesn't have a specific threadhold to say "below xxx is good".

Therefore, we can see that at k=3, the RMSE of predicted and training are the closest.

In this case, I can say that when we split the dataset into 3 times, providing about 180k rows of data, the model is trained to the best fit.