Cycle Data Feature selection

Declaration : The central idea and coding is abstract from Kevin mark ham youtube video seriese, Introduction to machine learning with scikit-learn video series. You can find link under resources section.

What are the **features**?

- trip_id: A unique number to identify each trip
- From station Number: From station number where the trip Start
- Day: Day of the trip for example Monday, Tuesday etc.
- Month: Which month trip took place
- Duration: Total trip duration in minutes
- birthyear: Birth year of user
- Sex: Gender identification of user
- age: Current age of user

What is the **response**?

- Station Number: To Station Number where the trip ends

The basic purpose of this exercise is by using linear regression and the value of RMSE to decided feature selection. The lower RMSE is better, If elimination of single column will produce in high RMSE we will keep that column and simultaneously, if the elimination of a column does not effect on RMSE or very slight increase we will get rid of that column.

Libraries used

In [3]:

```
import os,csv,io,mapsplotlib,time,folium,googlemaps,geopy,zipfile,requests,
warnings
import numpy as np
import pandas as pd
import datetime as dt
import seaborn as sns
import geopandas as gpd
from shapely.geometry import Point
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
import mysql.connector as sql
from sklearn.linear_model import LinearRegression
model = LinearRegression()
from sklearn import metrics
from sklearn.cross_validation import train_test_split
import numpy as np
warnings.simplefilter('ignore')
# display plots in the notebook
%matplotlib inline
```

Data Import

In [4]:

```
#importing data from database
db_connection = sql.connect(host='localhost', database='bike', user='root',
password='none')
db_cursor = db_connection.cursor()
db_cursor.execute('select trip_id,to_station_id_num
Station_Number,sthours,stphours,Day_num Day,bmonth Month,Year Trip_year,tri
pduration_minutes Duration,birthyear,Sex_num Sex,age from  trip_clean order
by to_station_id_num;;')
table_rows = db_cursor.fetchall()
data = pd.read_sql('select trip_id,to_station_id_num
Station_Number,sthours,stphours,Day_num Day,bmonth Month,Year Trip_year,tri
pduration_minutes Duration,birthyear,Sex_num Sex,age from  trip_clean order
by to_station_id_num;;', con=db_connection)
df = pd.DataFrame(data)
df.head()
```

Out[4]:

	trip_id	Station_Number	sthours	stphours	Day	Month	Trip_year	Duration	birthyear	Se
0	250220	0	18	18	0	8	2016	11.90	1981	1
1	250221	0	18	18	0	8	2016	9.15	1984	1
2	251032	0	18	19	3	8	2016	19.37	1986	1
3	251408	0	17	17	1	8	2016	8.04	1982	1
4	251472	0	19	19	1	8	2016	7.86	1988	1

In [5]:

```
print (model)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1,
normalize=False)
```

In [6]:

```
# print the shape of the DataFrame
df.shape
```

Out[6]:

```
(118931, 11)
```

There are 1,18,931 observations, and thus 1,18,931 Bicycle Trips in the dataset.

Feature selection

We're going to use **train/test split** (and eventually **cross-validation**).

Why not use of **p-values** or **R-squared** for feature selection?

Why not use of **p-values** or **R-squared** for feature selection?

- Linear models rely upon **a lot of assumptions** (such as the features being independent), and if those assumptions are violated, p-values and R-squared are less reliable. Train/test split relies on fewer assumptions.
- Features that are unrelated to the response can still have **significant p-values**.
- Adding features to your model that are unrelated to the response will always **increase the R-squared value**, and adjusted R-squared does not sufficiently account for this.
- p-values and R-squared are **proxies** for our goal of generalization, whereas train/test split and cross-validation attempt to **directly estimate** how well the model will generalize to out-of-sample data.

More generally:

- There are different methodologies that can be used for solving any given data science problem, and this study follows a **machine learning methodology**.
- This study focuses on **general purpose approaches** that can be applied to any model, rather than model-specific approaches.

Evaluation metrics for regression problems

Evaluation metrics for classification problems, such as **accuracy**, are not useful for regression problems. We need evaluation metrics designed for comparing **continuous values**.

Here are three common evaluation metrics for regression problems:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac 1n\sum_{i=1}^n|y_i-\hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac 1n\sum_{i=1}^n(y_i-\hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac 1n\sum_{i=1}^n(y_i-\hat{y}_i)^2}$$

In [7]:

```
# create a list of features
feature_cols = ['trip_id','sthours','stphours','Day','Month','Trip_year','D
uration','birthyear','Sex','age']
```

In [8]:

```
from sklearn.linear_model import LinearRegression
# create X and y
X = df[feature_cols]
y = df.Station_Number

# instantiate and fit
linreg = LinearRegression()
linreg.fit(X, y)

# print the coefficients
print (linreg.intercept_)
print (linreg.intercept_)
print (linreg.coef_)
```

```
-15683.3422751
[ -4.93324802e-05  -1.75253517e-01   1.23605681e-01   1.95458729e-02
   7.68145256e-01   7.75352316e+00   5.08998408e-02   4.34878715e-02
   9.42687746e-01  -4.34878715e-02]
```

In [9]:

```
# example true and predicted response values
true = [1, 7, 7, 5]
pred = [9, 4, 5, 1]
```

In [10]:

```
# calculate these metrics by hand!
from sklearn import metrics
import numpy as np
print ('MAE:', metrics.mean_absolute_error(true, pred))
print ('MSE:', metrics.mean_squared_error(true, pred))
print ('RMSE:', np.sqrt(metrics.mean_squared_error(true, pred)))
```

```
MAE: 4.25
MSE: 23.25
RMSE: 4.8218253805
```

Comparing these metrics:

- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are **loss functions**, because we want to minimize them.

Here's an additional example, to demonstrate how MSE/RMSE punish larger errors:

In [11]:

```
# same true values as above
true = [1, 7, 7, 5]

# new set of predicted values
pred = [4, 9, 5, 13]

# MAE is the same as before
print ('MAE:', metrics.mean_absolute_error(true, pred))

# MSE and RMSE are larger than before
print ('MSE:', metrics.mean_squared_error(true, pred))
print ('RMSE:', np.sqrt(metrics.mean_squared_error(true, pred)))
```

```
MAE: 3.75
MSE: 20.25
RMSE: 4.5
```

Comparing models with train/test split and RMSE

In [12]:

```
from sklearn.cross_validation import train_test_split

# define a function that accepts a list of features and returns testing
RMSE
def train_test_rmse(feature_cols):
    X = df[feature_cols]
    y = df.Station_Number
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1
23)
    linreg = LinearRegression()
    linreg.fit(X_train, y_train)
    y_pred = linreg.predict(X_test)
    return np.sqrt(metrics.mean_squared_error(y_test, y_pred))
```

In [13]:

```
# compare different sets of features
print (train_test_rmse(['trip_id','sthours','stphours','Day','Month','Trip_
year','Duration','birthyear','Sex','age']))
print (train_test_rmse(['trip_id','sthours','stphours','Day','Month','Trip_
year','Duration','birthyear','Sex',]))
print (train_test_rmse(['trip_id','sthours','stphours','Day','Month','Trip_
year','Duration']))
print (train_test_rmse(['trip_id','sthours','stphours','Day','Month','Trip_
year']))
print (train_test_rmse(['trip_id','sthours','stphours','Day','Month']))
print (train_test_rmse(['trip_id','sthours','stphours','Day']))
print (train_test_rmse(['trip_id','sthours','stphours']))
print (train_test_rmse(['trip_id','sthours']))
print (train_test_rmse(['trip_id']))
```

```
16.2297820193
16.2297820193
16.2530495511
16.2703148872
16.2802509935
16.2805520282
16.2804742826
16.2812235793
16.2823565359
```

By elimination each columns RMSE keep increasing. Where we want to decrease RMSE value.

Column Selection

sthours and stphours

In [14]:

```
# compare different sets of features
print (train_test_rmse(['trip_id','sthours','stphours','Day','Month','Trip_
year','Duration','birthyear','Sex','age']))
```

```
16.2297820193
```

In [15]:

```
print (train_test_rmse(['trip_id','sthours','Day','Month','Trip_year','Dura
tion','birthyear','Sex','age']))
```

```
16.229785169
```

In [16]:

```
print (train_test_rmse(['trip_id','sthours','Day','Month','Trip_year','Dura
tion','birthyear','Sex','age']))
```

```
16.229785169
```

By elimination stphours RMSE was decrease that’s mean stphours column has no significance our model.

Trip year and Birth year

In [17]:

```
print (train_test_rmse(['trip_id','sthours','Day','Month','Trip_year','Dura
tion','birthyear','Sex','age']))
```

```
16.229785169
```

In [18]:

```
print (train_test_rmse(['trip_id','sthours','Day','Month','Duration','birth
year','Sex','age']))
```

```
16.2391988286
```

In [19]:

```
print ('RMSE Increased By:',(16.2391988286 - 16.229785169))
```

```
RMSE Increased By: 0.009413659599999846
```

By elimination Trip Year RMSE is increase that’s mean Trip Year column contributing significance our model and that’s why we are keeping it.

In [20]:

```
print (train_test_rmse(['trip_id','sthours','Day','Month','Trip_year','Dura
tion','Sex','age']))
```

```
16.229785169
```

In [21]:

```
print ('RMSE Increased By:',(16.229785169 - 16.229785169))
```

```
RMSE Increased By: 0.0
```

By elimination birth year RMSE is no change that’s mean birth Year column contributing no significance our model and that’s why we are taking out birth year column it significance our model and that's why we are taking out birth year column it.

Day and Month

In [22]:

```
print (train_test_rmse(['trip_id','sthours','Day','Month','Trip_year','Dura
tion','Sex','age']))
```

```
16.229785169
```

In [23]:

```
print (train_test_rmse(['trip_id','Day','Trip_year','Duration','Sex','age']
))
```

```
16.2395959146
```

In [24]:

```
print ('RMSE Increased By:',(16.2395959146 - 16.229785169))
```

```
RMSE Increased By: 0.00981074559999393
```

In [25]:

```
print (train_test_rmse(['trip_id','Month','Trip_year','Duration','Sex','age
']))
```

```
16.2312840081
```

In [26]:

```
print ('RMSE Increased By:',(16.2312840081 - 16.229785169))
```

```
RMSE Increased By: 0.0014988391000017032
```

In [27]:

```
print (train_test_rmse(['trip_id','sthours','Day','Month','Trip_year','Dura
tion','Sex','age']))
```

```
16.229785169
```

By elimination Month RMSE is increase that's mean Month column contributing significance into our model and that's why we are keeping Month column. Where Day column Day column contributing significance into our model and that's why we are not eliminating Day column.

Age and Sex

In [28]:

```
print (train_test_rmse(['trip_id','sthours','Day','Month','Trip_year','Dura
tion','Sex','age']))
```

```
16.229785169
```

In [29]:

```
print (train_test_rmse(['trip_id','sthours','Day','Month','Trip_year','Dura
tion','Sex']))
```

```
16.2484520327
```

In [30]:

```
print ('RMSE Increased By:',(16.2484520327 - 16.229785169))
```

```
RMSE Increased By: 0.018666863700001812
```

In [31]:

```
print (train_test_rmse(['trip_id','sthours','Day','Month','Trip_year','Dura
tion','age']))
```

```
16.235345827
```

In [32]:

```
print ('RMSE Increased By:',(16.235345827 - 16.229785169))
```

```
RMSE Increased By: 0.005560658000000274
```

By elimination of Age Column RMSE is increase and that's we do not want

In [33]:

```
print (train_test_rmse(['trip_id','Month','Trip_year','Duration','age']))
```

```
16.2369696569
```

In [34]:

```
print ('RMSE Increased By:',(16.2312840081 - 16.2369696569))
```

```
RMSE Increased By: -0.005685648800000109
```

It is seems to be that both columns contributing some significance into our model and that’s why we are keeping age and Sex column.

Trip Id

In [35]:

```
print (train_test_rmse(['trip_id','sthours','Day','Month','Trip_year','Dura
tion','Sex','age']))
```

```
16.229785169
```

In [36]:

```
print (train_test_rmse(['Month','Trip_year','Duration','Sex','age']))
```

```
16.2383508452
```

In [37]:

```
print ('RMSE Increased By:',(16.2383508452 - 16.229785169))
```

```
RMSE Increased By: 0.008565676199999928
```

We are keeping trip_id Column because after eliminating trip_id our error mean squared error is increased

Final columns for our model

In [38]:

```
print (train_test_rmse(['trip_id','sthours','Day','Month','Trip_year','Dura
tion','Sex','age']))
```

```
16.229785169
```

Resources

References:*From the video series: Introduction to machine learning with scikit-learn*

- scikit-learn documentation: Cross-validation, Model evaluation
- scikit-learn issue on GitHub: MSE is negative when returned by cross_val_score
- Section 5.1 of An Introduction to Statistical Learning (11 pages) and related videos: K-fold and leave-one-out cross-validation (14 minutes), Cross-validation the right and wrong ways (10 minutes)
- Scott Fortmann-Roe: Accurately Measuring Model Prediction Error
- Machine Learning Mastery: An Introduction to Feature Selection
- Harvard CS109: Cross-Validation: The Right and Wrong Way
- Journal of Cheminformatics: Cross-validation pitfalls when selecting and assessing regression and classification models