Deshpande_Charudatta_Splitting_Data

January 27, 2018

1 Lab 4: Random numbers, splitting data, evaluating model performance

- **Author:** Niall Keleher (nkeleher@uw.edu)
- **Date:** 22 Jan 2018
- **Course:** INFX 574: Core Methods in Data Science

1.0.1 Learning Objectives:

By the end of the lab, you will be able to: * create dummy variables for use in regressions * generate random numbers for use in randomization and train-test splits * identify measures for evaluating regression performance

1.0.2 Topics:

1. Qualitative/Categorical predictors
2. Generating random numbers
3. Splitting data into training and test sets
4. Running regressions & generating predictions
5. Model performance

1.0.3 References:

- Pandas - get_dummies()
- random library
- Sci-kit Learn Cross Validation
- Introduction to Statistical Learning, Lab #5

In [2]:
```
#
# Student Name - Charudatta Deshpande
#
# Collaborators - Ram Ganesan, Charles Hemstreet, Mehdi Muntazir
#
##############################################
# Import required libraries.                 #
##############################################
#
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os as os
os.chdir('C:\\Users\deshc\Desktop\INFX 574 Data Science 2\Splitting Data Lab')
```

In [4]:
```
auto_df = pd.read_csv('Auto.csv')
auto_df = auto_df[auto_df.horsepower != '?']
```

In [5]:
```
auto_df.head()
```

Out[5]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

1.0.4 1. Qualitative/Categorical predictors - Generate dummy variables in python

In [6]:
```
auto_df.cylinders.value_counts()
```

Out[6]:
```
4    199
8    103
6     83
3      4
5      3
Name: cylinders, dtype: int64
```

In [7]:
```
pd.get_dummies(auto_df.cylinders).head()
```

Out[7]:

	3	4	5	6	8
0	0	0	0	0	1
1	0	0	0	0	1
2	0	0	0	0	1
3	0	0	0	0	1
4	0	0	0	0	1

In [8]:
```
cyl_dummies = pd.get_dummies(auto_df.cylinders, prefix='cyl')
```

In [9]:
```
auto_df2 = pd.concat([auto_df, cyl_dummies], axis=1)
```

In [10]:
```
auto_df2.head()
```

Out[10]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name	cyl_3	cyl_4	cyl_5	cyl_6	cyl_8
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	0	0	0	0	1
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	0	0	0	0	1
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	0	0	0	0	1
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	0	0	0	0	1
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	0	0	0	0	1

1.0.5 2. Generating random numbers - randomizing treatment assignment

```
In [11]: import random

In [12]: random.random()  # Random float x, 0.0 <= x < 1.0
```

Out[12]: 0.6422685684408944

```
In [13]: random.uniform(1,100)  # Random float x, 0.0 <= x < 100.0
```

Out[13]: 39.130228724258

```
In [14]: random.randint(1, 10)  # Integer from 1 to 10, endpoints included
```

Out[14]: 8

```
In [15]: random.sample([1, 2, 3, 4, 5],  3)
```

Out[15]: [5, 2, 1]

```
In [16]: random.seed(47653)

In [17]: raw_data = {'first_name': ['Niall', 'Josh', 'Li', 'Lavi', 'Jevin', 'Emma'],
                     'sex': ['male', 'male', 'female', 'male', 'male', 'female']}
         df = pd.DataFrame(raw_data, columns = ['first_name', 'sex'])

In [18]: df
```

Out[18]:

	first_name	sex
0	Niall	male
1	Josh	male
2	Li	female
3	Lavi	male
4	Jevin	male
5	Emma	female

```
In [19]: df['rand'] = df.apply(lambda row: random.random(), axis=1)

In [20]: df
```

Out[20]:

	first_name	sex	rand
0	Niall	male	0.009981
1	Josh	male	0.897681
2	Li	female	0.804464
3	Lavi	male	0.147438
4	Jevin	male	0.942135
5	Emma	female	0.426891

```
In [21]: df['treat'] = (df['rand']<.5)

In [22]: df
```

Out[22]:

	first_name	sex	rand	treat
0	Niall	male	0.009981	True
1	Josh	male	0.897681	False
2	Li	female	0.804464	False
3	Lavi	male	0.147438	True
4	Jevin	male	0.942135	False
5	Emma	female	0.426891	True

1.0.6 3. Splitting data into training and test sets

```
In [23]: auto_df['rand'] = auto_df.apply(lambda row: random.random(), axis=1)

In [24]: auto_df['train'] = (auto_df['rand']>.33)

In [25]: len(auto_df)

Out[25]: 392

In [26]: len(auto_df[auto_df['train']])

Out[26]: 277

In [27]: auto_train = auto_df[auto_df['train']]
```

Using Scikit-Learn

```
In [28]: from sklearn.cross_validation import train_test_split

C:\Users\deshc\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning:
  "This module will be removed in 0.20.", DeprecationWarning)


In [29]: X = auto_df['weight']

In [30]: y = auto_df['mpg']
```

```
In [31]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state
```

```
In [32]: len(X_train)
```

```
Out[32]: 262
```

```
In [33]: len(y_train)
```

```
Out[33]: 262
```

```
In [34]: len(X_test)
```

```
Out[34]: 130
```

```
In [35]: len(y_test)
```

```
Out[35]: 130
```

1.0.7 4. Running regressions & generating predictions

```
In [36]: auto_df.head(1)
```

```
Out[36]:     mpg  cylinders  displacement horsepower  weight  acceleration  year  \
         0  18.0          8         307.0        130    3504          12.0    70

            origin                       name      rand  train
         0       1  chevrolet chevelle malibu  0.880276   True
```

```
In [37]: import statsmodels.api as sm
         import statsmodels.formula.api as smf
         from sklearn.metrics import mean_squared_error, r2_score
```

```
C:\Users\deshc\Anaconda3\lib\site-packages\statsmodels\compat\pandas.py:56: FutureWarning: The
  from pandas.core import datetools
```

```
In [38]: overfit_mod = smf.ols(formula='mpg ~ weight', data = auto_df)
         overfit_result = overfit_mod.fit()
         print(overfit_result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    mpg   R-squared:                       0.693
Model:                            OLS   Adj. R-squared:                  0.692
Method:                 Least Squares   F-statistic:                     878.8
Date:                Sat, 27 Jan 2018   Prob (F-statistic):          6.02e-102
Time:                        15:38:46   Log-Likelihood:                -1130.0
No. Observations:                 392   AIC:                             2264.
Df Residuals:                     390   BIC:                             2272.
Df Model:                           1
```

```
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     46.2165      0.799     57.867      0.000      44.646      47.787
weight        -0.0076      0.000    -29.645      0.000      -0.008      -0.007
==============================================================================
Omnibus:                       41.682   Durbin-Watson:                   0.808
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               60.039
Skew:                           0.727   Prob(JB):                     9.18e-14
Kurtosis:                       4.251   Cond. No.                     1.13e+04
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.13e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
In [39]: train_mod = smf.ols(formula='mpg ~ weight', data = auto_train)
         train_result = train_mod.fit()
         print(train_result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    mpg   R-squared:                       0.705
Model:                            OLS   Adj. R-squared:                  0.704
Method:                 Least Squares   F-statistic:                     657.9
Date:                Sat, 27 Jan 2018   Prob (F-statistic):           6.49e-75
Time:                        15:38:48   Log-Likelihood:                -799.34
No. Observations:                 277   AIC:                             1603.
Df Residuals:                     275   BIC:                             1610.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     47.0463      0.959     49.048      0.000      45.158      48.935
weight        -0.0079      0.000    -25.650      0.000      -0.008      -0.007
==============================================================================
Omnibus:                       35.551   Durbin-Watson:                   0.814
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               54.661
Skew:                           0.779   Prob(JB):                     1.35e-12
Kurtosis:                       4.519   Cond. No.                     1.15e+04
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
[2] The condition number is large, 1.15e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

1.0.8 Exercise

Use scikitlearn to train a model to predict mpg using weight, horsepower, cylinders, displacement, acceleration, origin and year Reference: http://scikit-learn.org/stable/modules/linear_model.html#ordinary-least-squares

```
In [44]: from sklearn import linear_model
         from sklearn.model_selection import train_test_split

In [45]: lin_mod = linear_model.LinearRegression()

In [65]: #
         # Charu's code begins here. Read the auto file all over again.
         #
         auto_df = pd.read_csv('Auto.csv')

In [66]: #
         # Drop the name since it is not being used
         # Also remove the NA values
         #
         auto_df = auto_df.drop('name', axis=1)

         auto_df = auto_df.replace('?', np.nan)

         auto_df = auto_df.dropna()

In [67]: #
         # Create binary values for origin.
         # Also, add a new column for century and combine with year
         # That way we will have a 4 digit year
         # Then we will drop original year column and added 'century' column.
         #
         auto_df['origin'] = auto_df['origin'].replace({1: 'america', 2: 'europe', 3: 'asia'})

         auto_df = pd.get_dummies(auto_df, columns=['origin'])

         auto_df['century']='19'

         auto_df["year4digit"] = auto_df["century"].map(str) + auto_df["year"].map(str)

         auto_df = auto_df.drop('year', axis=1)

         auto_df = auto_df.drop('century', axis=1)

         print(auto_df.head())
```

```
    mpg  cylinders  displacement horsepower  weight  acceleration  \
0  18.0          8         307.0        130    3504          12.0
1  15.0          8         350.0        165    3693          11.5
2  18.0          8         318.0        150    3436          11.0
3  16.0          8         304.0        150    3433          12.0
4  17.0          8         302.0        140    3449          10.5

   origin_america  origin_asia  origin_europe year4digit
0               1            0              0       1970
1               1            0              0       1970
2               1            0              0       1970
3               1            0              0       1970
4               1            0              0       1970
```

In [68]:
```
#
# Now we create test data and training data, along with predictor variables and
# response variable. Response variable is the one which you need to predict,
# in this case 'mpg'. So we create the training data as below -
# X = all fields but mpg
# y = mpg
# and then we split the data using 'train_test_split' library.
# Training data size = 30%
#
X = auto_df.drop('mpg', axis=1)

y = auto_df[['mpg']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state
```

In [69]:
```
#
# We have already created a linear regression model lin_mod above.
# First, we will train the model using .fit command
# Then we will score and predict it.
#
lin_mod.fit(X_train, y_train)
lin_mod.score(X_test, y_test)
```

Out[69]:
```
0.8099743848671288
```

In [78]:
```
#
# So the values scored is 0.8099743848671288 which means 80.99 percent
# of variation in response variable can be explained using our
# predictor variables.
#
# Now we will predict the mpg of the two cars that I own -
# 2010 Honda Civic
# 2015 Nissan Altima
# since these cars are made in 21st century I made the year 4 digit on
```

```
        # purpose. So the model can accurately predict these mpg.
        #
        honda_civic_2010 = lin_mod.predict([[4, 120, 140, 2754, 15, 0, 1, 0, 2010]])
        print(honda_civic_2010)

        nissan_altima_2015 = lin_mod.predict([[4, 150, 182, 3114, 16, 0, 1, 0, 2015]])
        print(nissan_altima_2015)
```

```
[[ 51.9528469]]
[[ 53.13083058]]
```

In [ ]:
```
        #
        # Interpretation of results
        #
        # Honda civic 2010 - The calculated mpg is 51.9528469 which, I wish, but is not true.
        # The actual mpg is at about 30.
        #
        # Nissan Altima 2015 - The predicted mpg is 53.13083058 which again is not true.
        # Actual mpg is about 35.
        #
        # The most likely reason is incorrect values of parameters. I was not sure of the unit
        # acceleration and displacement. I provided what I thought was appropeiate.
        # But most likely I do not have correct values.
        # Also, I played with the parameters a little and found that the year is the most domi
        # factor. This being a linear model is assuming a linear relationship between mpg and
        # which is likely not true.
        # The data in the dataset is for a very narrow number of years.
        # It is highly likely that years 2010 and 2015 are causing it to skew by that
        # large amount.
```