SupervisedLearning_LogisticRegression

October 10, 2016

Original notebook

You'll need to install a new module we haven't used before: Statsmodels.

```
pip3 install statsmodels
```

Already included

In [1]:
```
# Data Imports
import numpy as np
import pandas as pd
from pandas import Series,DataFrame

# Math
import math

# Plot imports
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline

# Machine Learning Imports
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import train_test_split

# For evaluating our ML results
from sklearn import metrics

# Dataset Import
import statsmodels.api as sm
```

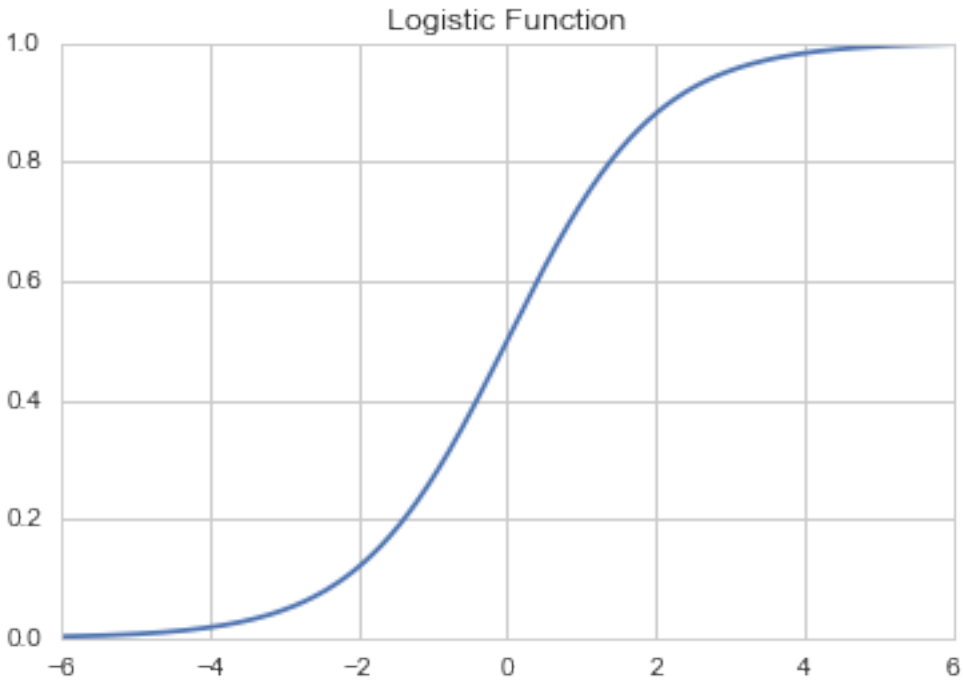
In [2]:
```
# Logistic Function
def logistic(t):
    return 1.0 / (1 + math.exp((-1.0)*t) )

# Set t from -6 to 6 ( 500 elements, linearly spaced)
t = np.linspace(-6,6,500)
```

```
# Set up y values (using list comprehension)
y = np.array([logistic(ele) for ele in t])

# Plot
plt.plot(t,y)
plt.title(' Logistic Function ')
```

Out[2]: <matplotlib.text.Text at 0xb8a8c18>



In [3]:
```
# Standard method of loading Statsmodels datasets into a pandas DataFrame.
df = sm.datasets.fair.load_pandas().data
# Preview
df.head()
```

Out[3]:

	rate_marriage	age	yrs_married	children	religious	educ	occupation	occupation_husb	affairs
0	3.0	32.0	9.0	3.0	3.0	17.0	2.0	5.0	0.111111
1	3.0	27.0	13.0	3.0	1.0	14.0	3.0	4.0	3.230769
2	4.0	22.0	2.5	0.0	1.0	16.0	3.0	5.0	1.400000
3	4.0	37.0	16.5	4.0	3.0	16.0	5.0		
4	5.0	27.0	9.0	1.0	1.0	14.0	3.0		

```
3              5.0  0.727273
4              4.0  4.666666
```

In [4]:
```
# Create check function
def affair_check(x):
    if x != 0:
        return 1
    else:
        return 0

# Apply to DataFrame
df['Had_Affair'] = df['affairs'].apply(affair_check)
#Let's go ahead and see the result!

# DataFrame Check
df
```

Out[4]:

	rate_marriage	age	yrs_married	children	religious	educ	occupati
0	3.0	32.0	9.0	3.0	3.0	17.0	2
1	3.0	27.0	13.0	3.0	1.0	14.0	3
2	4.0	22.0	2.5	0.0	1.0	16.0	3
3	4.0	37.0	16.5	4.0	3.0	16.0	5
4	5.0	27.0	9.0	1.0	1.0	14.0	3
5	4.0	27.0	9.0	0.0	2.0	14.0	3
6	5.0	37.0	23.0	5.5	2.0	12.0	5
7	5.0	37.0	23.0	5.5	2.0	12.0	2
8	3.0	22.0	2.5	0.0	2.0	12.0	3
9	3.0	27.0	6.0	0.0	1.0	16.0	3
10	2.0	27.0	6.0	2.0	1.0	16.0	3
11	5.0	27.0	6.0	2.0	3.0	14.0	3
12	3.0	37.0	16.5	5.5	1.0	12.0	2
13	5.0	27.0	6.0	0.0	2.0	14.0	3
14	4.0	22.0	6.0	1.0	1.0	14.0	4
15	4.0	37.0	9.0	2.0	2.0	14.0	3
16	4.0	27.0	6.0	1.0	1.0	12.0	3
17	1.0	37.0	23.0	5.5	4.0	14.0	5
18	2.0	42.0	23.0	2.0	2.0	20.0	4
19	4.0	37.0	6.0	0.0	2.0	16.0	5
20	5.0	22.0	2.5	0.0	2.0	14.0	3
21	3.0	37.0	16.5	5.5	2.0	9.0	3
22	3.0	42.0	23.0	5.5	3.0	12.0	5
23	2.0	27.0	9.0	2.0	4.0	20.0	3
24	4.0	27.0	6.0	1.0	2.0	12.0	5
25	5.0	27.0	2.5	0.0	3.0	16.0	4
26	2.0	27.0	6.0	2.0	2.0	12.0	2
27	5.0	37.0	13.0	1.0	3.0	12.0	3
28	2.0	32.0	16.5	2.0	2.0	12.0	4
29	3.0	27.0	6.0	1.0	1.0	14.0	3

...	...	...	...	...	...	...	.
6336	5.0	42.0	23.0	4.0	3.0	14.0	5
6337	5.0	27.0	6.0	0.0	4.0	14.0	4
6338	5.0	42.0	23.0	2.0	3.0	12.0	2
6339	4.0	32.0	13.0	3.0	3.0	16.0	4
6340	5.0	27.0	13.0	3.0	3.0	16.0	4
6341	5.0	27.0	9.0	1.0	2.0	14.0	4
6342	4.0	22.0	2.5	0.0	2.0	16.0	4
6343	5.0	17.5	2.5	0.0	4.0	12.0	3
6344	4.0	32.0	16.5	2.0	2.0	12.0	3
6345	5.0	27.0	9.0	1.0	3.0	12.0	3
6346	4.0	22.0	2.5	0.0	4.0	14.0	4
6347	5.0	22.0	2.5	1.0	2.0	12.0	3
6348	5.0	27.0	0.5	0.0	4.0	20.0	4
6349	5.0	37.0	16.5	3.0	3.0	14.0	5
6350	5.0	32.0	13.0	2.0	4.0	14.0	3
6351	4.0	22.0	0.5	0.0	2.0	16.0	3
6352	5.0	42.0	23.0	2.0	4.0	12.0	3
6353	5.0	22.0	2.5	2.0	2.0	14.0	3
6354	5.0	42.0	23.0	4.0	4.0	12.0	3
6355	4.0	27.0	6.0	0.0	3.0	12.0	3
6356	5.0	32.0	13.0	3.0	3.0	12.0	3
6357	5.0	32.0	13.0	4.0	2.0	14.0	4
6358	3.0	27.0	6.0	2.0	4.0	14.0	3
6359	4.0	22.0	2.5	0.0	3.0	16.0	5
6360	5.0	22.0	2.5	0.0	2.0	14.0	3
6361	5.0	32.0	13.0	2.0	3.0	17.0	4
6362	4.0	32.0	13.0	1.0	1.0	16.0	5
6363	5.0	22.0	2.5	0.0	2.0	14.0	3
6364	5.0	32.0	6.0	1.0	3.0	14.0	3
6365	4.0	22.0	2.5	0.0	2.0	16.0	2

	occupation_husb	affairs	Had_Affair
0	5.0	0.111111	1
1	4.0	3.230769	1
2	5.0	1.400000	1
3	5.0	0.727273	1
4	4.0	4.666666	1
5	4.0	4.666666	1
6	4.0	0.852174	1
7	3.0	1.826086	1
8	3.0	4.799999	1
9	5.0	1.333333	1
10	5.0	3.266665	1
11	5.0	2.041666	1
12	3.0	0.484848	1
13	2.0	2.000000	1
14	4.0	3.266665	1

15	6.0	1.361111	1
16	5.0	2.000000	1
17	2.0	1.826086	1
18	4.0	1.826086	1
19	4.0	2.041666	1
20	4.0	7.839996	1
21	2.0	2.545454	1
22	4.0	0.532609	1
23	4.0	0.622222	1
24	4.0	0.583333	1
25	1.0	4.799999	1
26	5.0	0.166667	1
27	4.0	0.615385	1
28	2.0	1.187878	1
29	6.0	11.199999	1
...	...	...	...
6336	4.0	0.000000	0
6337	4.0	0.000000	0
6338	2.0	0.000000	0
6339	2.0	0.000000	0
6340	2.0	0.000000	0
6341	5.0	0.000000	0
6342	1.0	0.000000	0
6343	5.0	0.000000	0
6344	4.0	0.000000	0
6345	5.0	0.000000	0
6346	2.0	0.000000	0
6347	2.0	0.000000	0
6348	4.0	0.000000	0
6349	5.0	0.000000	0
6350	6.0	0.000000	0
6351	1.0	0.000000	0
6352	2.0	0.000000	0
6353	5.0	0.000000	0
6354	5.0	0.000000	0
6355	4.0	0.000000	0
6356	5.0	0.000000	0
6357	4.0	0.000000	0
6358	1.0	0.000000	0
6359	5.0	0.000000	0
6360	3.0	0.000000	0
6361	3.0	0.000000	0
6362	5.0	0.000000	0
6363	1.0	0.000000	0
6364	4.0	0.000000	0
6365	4.0	0.000000	0

[6366 rows x 10 columns]

In [5]:
```
# Groupby Had Affair column
df.groupby('Had_Affair').mean()
```

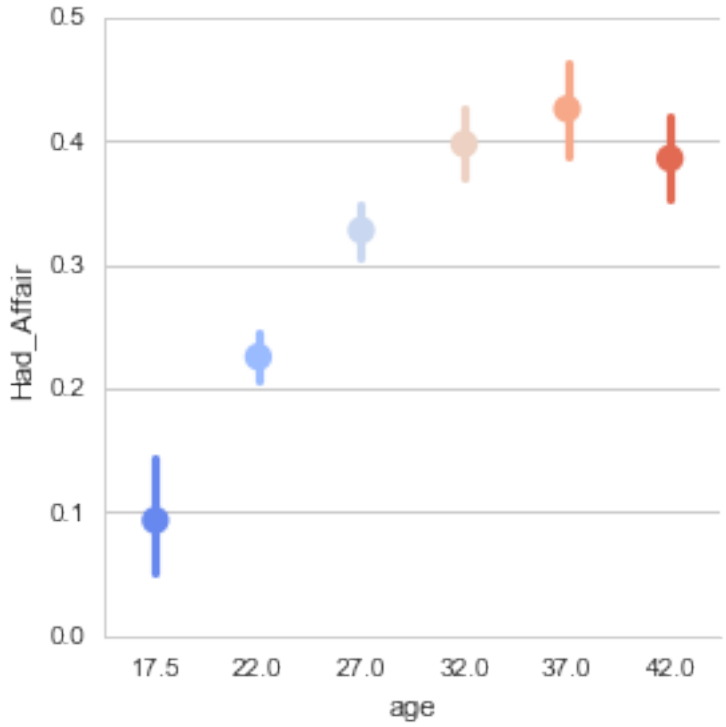
Out[5]:

Had_Affair	rate_marriage	age	yrs_married	children	religious	\
0	4.329701	28.390679	7.989335	1.238813	2.504521	
1	3.647345	30.537019	11.152460	1.728933	2.261568	

Had_Affair	educ	occupation	occupation_husb	affairs
0	14.322977	3.405286	3.833758	0.000000
1	13.972236	3.463712	3.884559	2.187243

In [25]:
```
# Factorplot for age with Had Affair hue
#sns.factorplot('age',data=df,hue='Had_Affair',palette='coolwarm')# print
sns.factorplot('age','Had_Affair',data=df,palette='coolwarm')#,hue='Had_Af
```

Out[25]: <seaborn.axisgrid.FacetGrid at 0x77ec898>



In [13]:
```
# Create new DataFrames for the Categorical Variables
occ_dummies = pd.get_dummies(df['occupation'])
hus_occ_dummies = pd.get_dummies(df['occupation_husb'])

# Let's take a quick look at the results
```

```
occ_dummies.head()
#hus_occ_dummies.head()
```

Out[13]:

	1.0	2.0	3.0	4.0	5.0	6.0
0	0.0	1.0	0.0	0.0	0.0	0.0
1	0.0	0.0	1.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	1.0	0.0	0.0	0.0
5	0.0	0.0	1.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	1.0	0.0
7	0.0	1.0	0.0	0.0	0.0	0.0
8	0.0	0.0	1.0	0.0	0.0	0.0
9	0.0	0.0	1.0	0.0	0.0	0.0
10	0.0	0.0	1.0	0.0	0.0	0.0
11	0.0	0.0	1.0	0.0	0.0	0.0
12	0.0	1.0	0.0	0.0	0.0	0.0
13	0.0	0.0	1.0	0.0	0.0	0.0
14	0.0	0.0	0.0	1.0	0.0	0.0
15	0.0	0.0	1.0	0.0	0.0	0.0
16	0.0	0.0	1.0	0.0	0.0	0.0
17	0.0	0.0	0.0	0.0	1.0	0.0
18	0.0	0.0	0.0	1.0	0.0	0.0
19	0.0	0.0	0.0	0.0	1.0	0.0
20	0.0	0.0	1.0	0.0	0.0	0.0
21	0.0	0.0	1.0	0.0	0.0	0.0
22	0.0	0.0	0.0	0.0	1.0	0.0
23	0.0	0.0	1.0	0.0	0.0	0.0
24	0.0	0.0	0.0	0.0	1.0	0.0
25	0.0	0.0	0.0	1.0	0.0	0.0
26	0.0	1.0	0.0	0.0	0.0	0.0
27	0.0	0.0	1.0	0.0	0.0	0.0
28	0.0	0.0	0.0	1.0	0.0	0.0
29	0.0	0.0	1.0	0.0	0.0	0.0
...	...	...	...	...	...	...
6336	0.0	0.0	0.0	0.0	1.0	0.0
6337	0.0	0.0	0.0	1.0	0.0	0.0
6338	0.0	1.0	0.0	0.0	0.0	0.0
6339	0.0	0.0	0.0	1.0	0.0	0.0
6340	0.0	0.0	0.0	1.0	0.0	0.0
6341	0.0	0.0	0.0	1.0	0.0	0.0
6342	0.0	0.0	0.0	1.0	0.0	0.0
6343	0.0	0.0	1.0	0.0	0.0	0.0
6344	0.0	0.0	1.0	0.0	0.0	0.0
6345	0.0	0.0	1.0	0.0	0.0	0.0
6346	0.0	0.0	0.0	1.0	0.0	0.0
6347	0.0	0.0	1.0	0.0	0.0	0.0
6348	0.0	0.0	0.0	1.0	0.0	0.0

```
6349  0.0  0.0  0.0  0.0  1.0  0.0
6350  0.0  0.0  1.0  0.0  0.0  0.0
6351  0.0  0.0  1.0  0.0  0.0  0.0
6352  0.0  0.0  1.0  0.0  0.0  0.0
6353  0.0  0.0  1.0  0.0  0.0  0.0
6354  0.0  0.0  1.0  0.0  0.0  0.0
6355  0.0  0.0  1.0  0.0  0.0  0.0
6356  0.0  0.0  1.0  0.0  0.0  0.0
6357  0.0  0.0  0.0  1.0  0.0  0.0
6358  0.0  0.0  1.0  0.0  0.0  0.0
6359  0.0  0.0  0.0  0.0  1.0  0.0
6360  0.0  0.0  1.0  0.0  0.0  0.0
6361  0.0  0.0  0.0  1.0  0.0  0.0
6362  0.0  0.0  0.0  0.0  1.0  0.0
6363  0.0  0.0  1.0  0.0  0.0  0.0
6364  0.0  0.0  1.0  0.0  0.0  0.0
6365  0.0  1.0  0.0  0.0  0.0  0.0

[6366 rows x 6 columns]
```

In [15]:
```
# Create column names for the new DataFrames
occ_dummies.columns = ['occ1','occ2','occ3','occ4','occ5','occ6']
hus_occ_dummies.columns = ['hocc1','hocc2','hocc3','hocc4','hocc5','hocc6'

# Set X as new DataFrame without the occupation columns or the Y target
X = df.drop(['occupation','occupation_husb','Had_Affair'],axis=1)

# Concat the dummy DataFrames Together
dummies = pd.concat([occ_dummies,hus_occ_dummies],axis=1)

# Now Concat the X DataFrame with the dummy variables
X = pd.concat([X,dummies],axis=1)

# Preview of Result
X.head()
```

Out[15]:

	rate_marriage	age	yrs_married	children	religious	educ	affairs
0	3.0	32.0	9.0	3.0	3.0	17.0	0.111111
1	3.0	27.0	13.0	3.0	1.0	14.0	3.230769
2	4.0	22.0	2.5	0.0	1.0	16.0	1.400000
3	4.0	37.0	16.5	4.0	3.0	16.0	0.727273
4	5.0	27.0	9.0	1.0	1.0	14.0	4.666666

	occ1	occ2	occ3	occ4	occ5	occ6	hocc1	hocc2	hocc3	hocc4	hocc5
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0

```
4   0.0   0.0   1.0   0.0   0.0   0.0   0.0   0.0   0.0   1.0   0.0

   hocc6
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
```

In [16]:
```
# Set Y as Target class, Had Affair
Y = df.Had_Affair

# Preview
Y.head()
```

Out[16]:
```
0    1
1    1
2    1
3    1
4    1
Name: Had_Affair, dtype: int64
```

In [17]:
```
# Dropping one column of each dummy variable set to avoid multicollinearit
X = X.drop('occ1',axis=1)
X = X.drop('hocc1',axis=1)

# Drop affairs column so Y target makes sense
X = X.drop('affairs',axis=1)

# PReview
X.head()

#In order to use the Y with SciKit Learn, we need to set it as a 1-D array

# Flatten array
Y = np.ravel(Y)

# Check result
Y
```

Out[17]:
```
array([1, 1, 1, ..., 0, 0, 0], dtype=int64)
```

In [19]:
```
#Let's start by initiating the model!

# Create LogisticRegression model
log_model = LogisticRegression()

# Fit our data
log_model.fit(X,Y)
```

```
In [19]: # Check our accuracy
         log_model.score(X,Y)

Out[19]: 0.72588752748978946

In [20]: # Check percentage of women that had affairs
         Y.mean()

Out[20]: 0.32249450204209867

In [21]: # Use zip to bring the column names and the np.transpose function to bring
         coeff_df = DataFrame(zip(X.columns, np.transpose(log_model.coef_)))

         ---------------------------------------------------------------------------
         TypeError                                 Traceback (most recent call last)

         <ipython-input-21-13fbffd0779d> in <module>()
           1 # Use zip to bring the column names and the np.transpose function to br
         ----> 2 coeff_df = DataFrame(zip(X.columns, np.transpose(log_model.coef_)))


         c:\Python\WinPython-64bit-3.4.4.4Qt5\python-3.4.4.amd64\lib\site-packages\p
         281                 mgr = self._init_dict({}, index, columns, dtype=dtype)
         282         elif isinstance(data, collections.Iterator):
         --> 283             raise TypeError("data argument can't be an iterator")
         284         else:
         285             try:


         TypeError: data argument can't be an iterator


In [22]: # Split the data
         X_train, X_test, Y_train, Y_test = train_test_split(X, Y)

         # Make a new log_model
         log_model2 = LogisticRegression()

         # Now fit the new model
         log_model2.fit(X_train, Y_train)

Out[22]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

In [23]:
```
# Predict the classes of the testing data set
class_predict = log_model2.predict(X_test)

# Compare the predicted classes to the actual test classes
print metrics.accuracy_score(Y_test,class_predict)
```

```
  File "<ipython-input-23-dfe55fa34995>", line 5
    print metrics.accuracy_score(Y_test,class_predict)
                ^
SyntaxError: invalid syntax
```

In [ ]: