Logistic regression demo for computer-generated stroke data

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
#from sklearn.cross_validation import cross_val_score
from sklearn.model_selection import cross_val_score

df = pd.read_csv('stroke_v1.csv', sep=',')
df.head(10)
```

Out[1]:

	ID	Weight	Smoking	Exercise	Cholesterol	Income	Exphappiness	Birthyear	Sex	Strol
0	1	117	1	2	8.0	1080	27	1913	M	
1	2	62	0	8	5.5	2120	55	1949	M	
2	3	74	0	6	4.8	3170	65	1976	M	
3	4	77	0	5	4.2	4740	61	1973	F	
4	5	67	0	8	4.5	1900	53	1929	M	
5	6	76	0	6	6.2	3410	72	1959	F	
6	7	63	0	7	4.1	3640	71	1979	F	
7	8	75	0	5	5.2	2500	99	1960	F	
8	9	70	0	6	4.9	2110	48	1922	F	
9	10	82	0	5	5.8	2560	34	2007	F	

In [2]:

```
# re-encode gender column
df['Sex'].replace(['M','F'], [1,2], inplace=True)

# drop ID
df.drop('ID', axis=1, inplace=True)

df.head(10)
```

Out[2]:

	Weight	Smoking	Exercise	Cholesterol	Income	Exphappiness	Birthyear	Sex	Stroke
0	117	1	2	8.0	1080	27	1913	1	1
1	62	0	8	5.5	2120	55	1949	1	0
2	74	0	6	4.8	3170	65	1976	1	0
3	77	0	5	4.2	4740	61	1973	2	0
4	67	0	8	4.5	1900	53	1929	1	0
5	76	0	6	6.2	3410	72	1959	2	0
6	63	0	7	4.1	3640	71	1979	2	0
7	75	0	5	5.2	2500	99	1960	2	0
8	70	0	6	4.9	2110	48	1922	2	0
9	82	0	5	5.8	2560	34	2007	2	1

In [3]:

```
df.describe()
```

Out[3]:

	Weight	Smoking	Exercise	Cholesterol	Income	Exphappiness	E
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	100
mean	78.752000	0.223000	5.119000	5.658400	2827.990000	51.024000	195
std	13.939038	0.416467	1.924199	1.312262	1105.714549	16.805888	2
min	27.000000	0.000000	0.000000	0.100000	-790.000000	0.000000	187
25%	70.000000	0.000000	4.000000	4.800000	2087.500000	40.000000	194
50%	78.000000	0.000000	5.000000	5.600000	2830.000000	51.000000	195
75%	88.000000	0.000000	6.000000	6.500000	3562.500000	62.000000	197
max	130.000000	1.000000	10.000000	9.900000	5860.000000	100.000000	202

In [4]:

```
# split into explanatory and response variables
X = df.iloc[:,:8]
Y = df.iloc[:,8]
```

In [5]:

```
# build and fit model
reg = LogisticRegression(solver="liblinear")
reg.fit(X,Y)

print("Coefficients: ",reg.coef_)
print("Intercept: ", reg.intercept_)

# compute predicted values from training set
Y_pred = reg.predict(X)

cm = confusion_matrix(Y, Y_pred)
print("Confusion matrix:\n",cm)

accuracy = (cm[0][0]+cm[1][1])/(cm[0][0]+cm[1][1]+cm[0][1]+cm[1][0])
print("Accuracy calculated from the training set = %.3f" % (accuracy))

print(classification_report(Y, Y_pred, target_names=['no', 'yes']))
```

```
Coefficients:  [[ 7.39700165e-02  1.84023310e-01 -3.04569922e-01  1.722108
45e-01
   1.17005610e-05 -2.69957499e-03 -2.90730721e-03 -4.83472447e-02]]
Intercept:  [-0.006133]
Confusion matrix:
 [[509  91]
 [147 253]]
Accuracy calculated from the training set = 0.762
              precision    recall  f1-score   support

          no       0.78      0.85      0.81       600
         yes       0.74      0.63      0.68       400

    accuracy                           0.76      1000
   macro avg       0.76      0.74      0.75      1000
weighted avg       0.76      0.76      0.76      1000
```

In [6]:

```
# cross-validate
# number of folds
k = 10
scores = cross_val_score(estimator=reg,
                         X=X,
                         y=Y,
                         scoring="accuracy",
                         cv=k)
print("Accuracies from %d individual folds:" % k)
print(scores)
print("Accuracy calculated using %d-fold cross validation = %.3f" % (k, scores.mean()))
```

```
Accuracies from 10 individual folds:
[0.75 0.81 0.75 0.7  0.7  0.79 0.77 0.81 0.71 0.77]
Accuracy calculated using 10-fold cross validation = 0.756
```

In [7]:

```
# for demonstration, retrieve estimated probabilities (from training set)
reg.predict_proba(X)
```

Out[7]:

```
array([[0.01925293, 0.98074707],
       [0.93978627, 0.06021373],
       [0.81225461, 0.18774539],
       ...,
       [0.25819284, 0.74180716],
       [0.13354553, 0.86645447],
       [0.65902315, 0.34097685]])
```

In [9]:

```
# make a single prediction
Xnew = [[80.0, 0, 8, 5.2, 2900, 73, 1971, 1]]
reg.predict_proba(Xnew)
```

Out[9]:

```
array([[0.8279805, 0.1720195]])
```

In [ ]:

```

```