2. Binary logistic regression.

In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import time
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn.model_selection import LeaveOneOut
from sklearn.neighbors import KNeighborsClassifier
import matplotlib
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
from sklearn.feature_selection import chi2
from sklearn.model_selection import cross_val_score
import sys

if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")
```

In [2]:
```
dataframe = read_csv('heart.csv')
dataframe
```

Out[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

(a) Randomly partition the data into 200 training points and 103 test points. Fit a logistic regression model to the training data and display the coefficients of the model. If you had to choose the three features that were most influential in the model, what would they be?

In [3]:
```
# Separate features from labels
data = dataframe.values
X, y = data[:, :-1], data[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 103/303, random_state = 42)
```

In [4]:
```
clf = LogisticRegression()
clf.fit(X_train, y_train)
predict = clf.predict(X_test)
coef = clf.coef_
print('Predicted values:\n', predict)
print('\nCoefficients of the model:\n', coef)
```

```
Predicted values:
 [0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 0. 1.
 1. 1. 1. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1.
 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 1. 0. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 0.
 1. 0. 1. 0. 1. 1. 1.]

Coefficients of the model:
 [[ 2.32276008e-02 -1.04450857e+00  9.11340974e-01 -8.11154500e-03
  -6.53211612e-04  5.32287897e-02  5.21463825e-01  1.87937460e-02
  -8.27825714e-01 -4.83269274e-01  7.53282739e-01 -1.37895276e+00
  -1.22092016e+00]]
```

In [5]:
```
# Three most influential features in this model
n_most_influential_features = 3
coef_three_features = np.argpartition(coef, X.shape[1] - n_most_influential_features)
coef_three_features = coef_three_features.tolist()

# Feature index
index_1 = coef_three_features[0][-1]
index_2 = coef_three_features[0][-2]
index_3 = coef_three_features[0][-3]

# Feature name
feature_1 = dataframe.columns[index_1]
feature_2 = dataframe.columns[index_2]
feature_3 = dataframe.columns[index_3]

print('Three most influential features in this model:\n', feature_1, feature_2, feature_3)
```

```
Three most influential features in this model:
 slope cp restecg
```

In [6]:
```
# Another way to select features is to use the chi2, find three smallest pvalues
n_most_influential_features_pvalue = 3
scores, pvalues = chi2(X_train, y_train)
three_features = np.argpartition(pvalues, n_most_influential_features_pvalue)

# Feature index
p_index_1 = three_features[0]
p_index_2 = three_features[1]
p_index_3 = three_features[2]

# Feature name
p_feature_1 = dataframe.columns[p_index_1]
p_feature_2 = dataframe.columns[p_index_2]
p_feature_3 = dataframe.columns[p_index_3]

print('Three most influential features in this model:\n', p_feature_1, p_feature_2, p_feature_3)
```

```
Three most influential features in this model:
 thalach ca cp
```

(b) What is the test error of your model?

In [7]:
```
accuracy = accuracy_score(y_test,predict)
error = 1 - accuracy
print('The test error of the model is:\n', error)
```

```
The test error of the model is:
 0.19417475728155342
```

(c) Estimate the error by using 5-fold cross-validation on the training set. How does this compare to the test error?

In [8]:
```
kf = KFold(n_splits = 5, random_state = 42, shuffle = True)
acc_score = []

for train_index, test_index in kf.split(X):
    kf_predict_values = []
    X_train_kf, X_test_kf = X[train_index, :], X[test_index, :]
    y_train_kf, y_test_kf = y[train_index], y[test_index]

    clf = LogisticRegression(solver = 'liblinear')
    clf.fit(X_train_kf, y_train_kf)
    kf_predict_labels = clf.predict(X_test_kf)
    acc = accuracy_score(y_test_kf, kf_predict_labels)
    kf_predict_values.append(kf_predict_labels)
    acc_score.append(acc)

avg_acc_score = sum(acc_score)/5
error_kf = 1-avg_acc_score

print('\nAvg accuracy : {}'.format(avg_acc_score))
print('\nError rate : {}'.format(error_kf))
```

```
Avg accuracy : 0.8282513661202187

Error rate : 0.17174863387978134
```

This error rate is a little bit lower than the pure logistic regression classifier. The 5-fold cross-validation has a higher accuracy than the pure logistic regression classifier.

In [ ]:
```
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js