## #Simple logistic regression

### #load Data

import pandas as pd

url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/glass/glass.data'

col\_names = ['id','ri','na','mg','al','si','k','ca','ba','fe','glass\_type']

glass = pd.read\_csv(url, names=col\_names, index\_col='id')

glass.sort('al', inplace=True)

glass.head()

col\_names = ['id','ri','na','mg','al','si','k','ca','ba','fe','glass\_type']

glass=pd.read\_csv(‘glass.csv’,names=col\_names,index\_col='id')

glass.sort('al', inplace=True)

glass.head()

### # types 1, 2, 3 are window glass

### # types 5, 6, 7 are household glass

glass['household'] = glass.glass\_type.map({1:0, 2:0, 3:0, 5:1, 6:1, 7:1})

glass.head()

### #split data into train and test set and carryout LR

feature\_cols = ['ri','na','mg','al','si','k','ca','ba','fe']

X = glass[feature\_cols]

y = glass.household

from sklearn.cross\_validation import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y)

#### #Utility Details:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

... X, y, test\_size=0.33, random\_state=42)

**train\_size** : float, int, or None (default is None)

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.

**random\_state :** int or RandomState

Pseudo-random number generator state used for random sampling.

### #fitting data into logistic regression

from sklearn.linear\_model import LogisticRegression

logreg = LogisticRegression(C=1e9)

feature\_cols = ['ri','na','mg','al','si','k','ca','ba','fe']

X = glass[feature\_cols]

y = glass.household

logreg.fit(X, y)

glass['household\_pred\_class'] = logreg.predict(X)

### #model evaluation

y\_pred=logreg.predict(X\_test)

y\_pred=list(y\_pred)

y\_test=list(y\_test)

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

confusion\_matrix = confusion\_matrix(y\_test, y\_pred)

print 'Accuracy: ', accuracy\_score(y\_test, y\_pred)

Accuracy is :0.666666666667

Accuracy is :0.96511627907(when split train:test::60:40

### #intercept and coefficients

logreg.coef\_ <-type is array

logreg.intercept\_

### #utility Details

**LogisticRegressionCV**(*Cs=10*, *fit\_intercept=True*, *cv=None*, *dual=False*,*penalty='l2'*, *scoring=None*, *solver='lbfgs'*, *tol=0.0001*, *max\_iter=100*, *class\_weight=None*, *n\_jobs=1*, *verbose=0*,*refit=True*, *intercept\_scaling=1.0*, *multi\_class='ovr'*, *random\_state=None*)

***solver : {‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’}***

***Algorithm to use in the optimization problem.***

***For small datasets, ‘liblinear’ is a good choice, whereas ‘sag’ is***

***faster for large ones.***

Use ‘sag’ <- Explained by Andrew NJ stochastic average gradient .

***penalty : str, ‘l1’ or ‘l2’***

***Used to specify the norm used in the penalization. The newton-cg and lbfgs solvers support only l2 penalties.***

*Use ‘l2’ LASSO regularized <- Explained by Andrew NJ*

***Cs : list of floats | int***

***Each of the values in Cs describes the inverse of regularization strength. If Cs is as an int, then a grid of Cs values are chosen in a logarithmic scale between 1e-4 and 1e4. Like in support vector machines, smaller values specify stronger regularization.***