**Logistic Regression**

**Logistic regression** is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

**Implementation in Python**

Anaconda Python 3 Package

**Dataset**

<https://www.kaggle.com/uciml/pima-indians-diabetes-database>

**Loading Data**

Let's first load the required libraries. Using sklearn package load the required Pima Indian Diabetes dataset using pandas' read CSV function.

#import pandas

import pandas as pd

col\_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']

# load dataset

pima = pd.read\_csv("pima-indians-diabetes.csv", header=None, names=col\_names)

pima.head()

#### Selecting Features

Here, we need to divide given columns into two types of variables dependent(or target variable) and independent variable(or feature variables).

#split dataset in features and target variable

feature\_cols=['pregnant','insulin', 'bmi', 'age','glucose','bp','pedigree']

X = pima[feature\_cols] # Features

y = pima.label # Target variable

#### Splitting Data

To understand model performance, dividing the dataset into a training set and a test set is a good strategy.Let's split dataset by using function train\_test\_split(). You need to pass 3 parameters features, target, and test\_set size. Additionally, you can use random\_state to select records randomly.

# split X and y into training and testing sets

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

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=0)

Here, the Dataset is broken into two parts in a ratio of 75:25. It means 75% data will be used for model training and 25% for model testing.

#### Model Development and Prediction

First, import the Logistic Regression module and create a Logistic Regression classifier object using LogisticRegression() function.Then, fit the model on the train set using fit() and perform prediction on the test set using predict().

# import the class

from sklearn.linear\_model import LogisticRegression

# instantiate the model (using the default parameters)

logreg = LogisticRegression()

# fit the model with data

logreg.fit(X\_train,y\_train)

# Predict the response for test dataset

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

### Model Evaluation using Confusion Matrix

A confusion matrix is a table that is used to evaluate the performance of a classification model. We can also visualize the performance of an algorithm. The fundamental of a confusion matrix is the number of correct and incorrect predictions are summed up class-wise.

# import the metrics class

from sklearn import metrics

cnf\_matrix = metrics.confusion\_matrix(y\_test, y\_pred)

cnf\_matrix

#### Confusion Matrix Evaluation Metrics

#### Accuracy

Accuracy (ACC) is calculated as the number of all correct predictions divided by the total number of the dataset. The best accuracy is 1.0, whereas the worst is 0.0.

#### Sensitivity (Recall or True positive rate)

Sensitivity (SN) is calculated as the number of correct positive predictions divided by the total number of positives. It is also called recall (REC) or true positive rate (TPR). The best sensitivity is 1.0, whereas the worst is 0.0.

#### Precision (Positive predictive value)

Precision (PREC) is calculated as the number of correct positive predictions divided by the total number of positive predictions. It is also called positive predictive value (PPV). The best precision is 1.0, whereas the worst is 0.0.

print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))

print("Precision:",metrics.precision\_score(y\_test, y\_pred))

print("Recall:",metrics.recall\_score(y\_test, y\_pred))