Classification

By

Dr Shaik A Qadeer

Learning Objectives

- Understanding the concepts logistic regression and its applications in predictive analytics.
- Building Binary classification models using Python package such as *sklearn* APIs.
- Learning to incorporate model deployment operation.
- Applying Decision tree and ensemble algorithm to SLR

Overview of classification

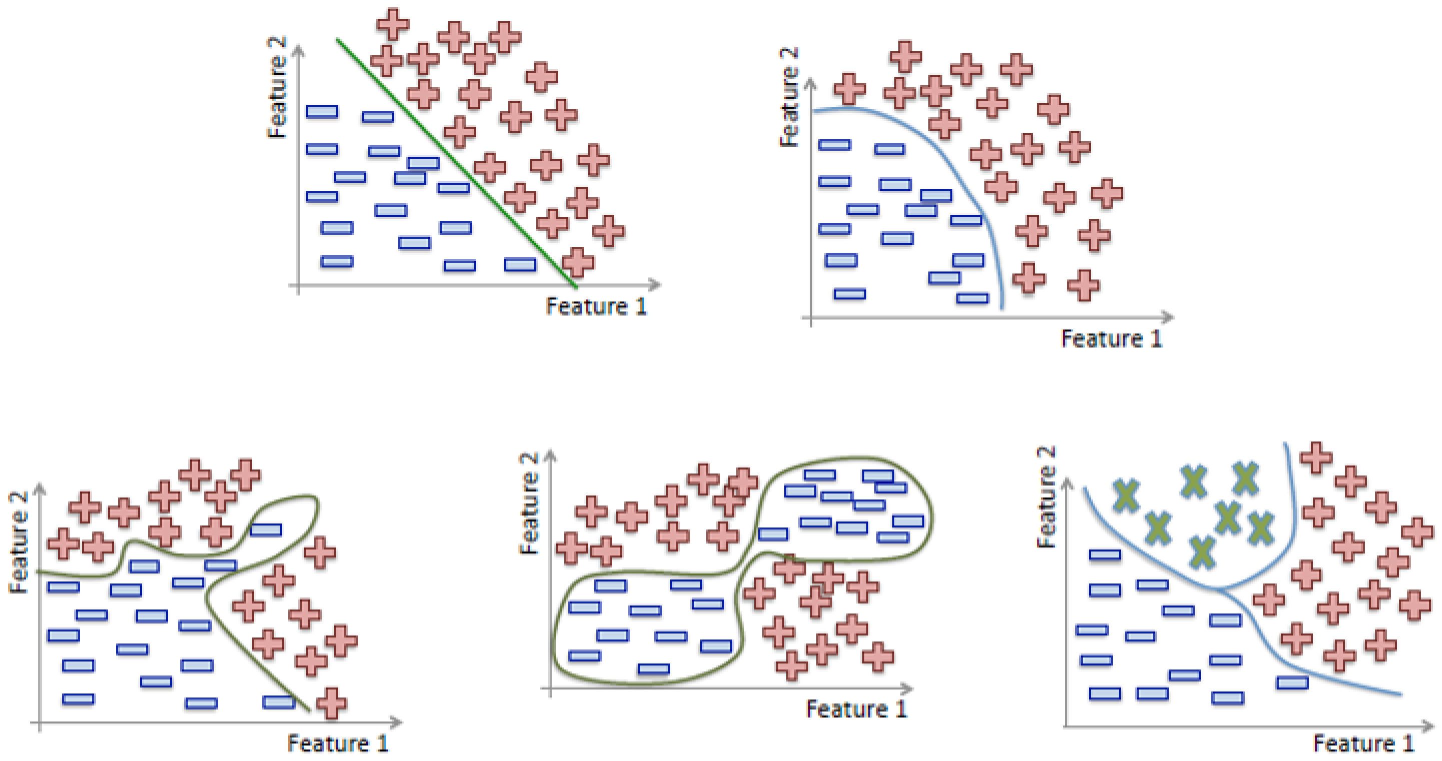
- Classification problems are an important category of problems where outcome variable takes discrete values.
- Primary objective is to predict the probability of an observation belonging to a class, known as class probability.
- Few examples of classification problems are:
    1. A bank would to classify the low-risk and high-risk customers.
    2. E-commerce providers would predict whether a customer is likely to churn or not.
    3. The HR department of a firm may want to predict if an applicant would accept an offer or not.

Overview of classification..

- Classification problems with binary outcomes are called binary classification.
- Classification problems with multiple outcomes are called multinomial classification.
- Techniques used for solving classification problems
    1. Logistic Regression
    2. Classification Trees
    3. Discriminant Analysis
    4. Neural Networks
    5. Support Vector Machines (SVM)

Overview of classification..

Classification:



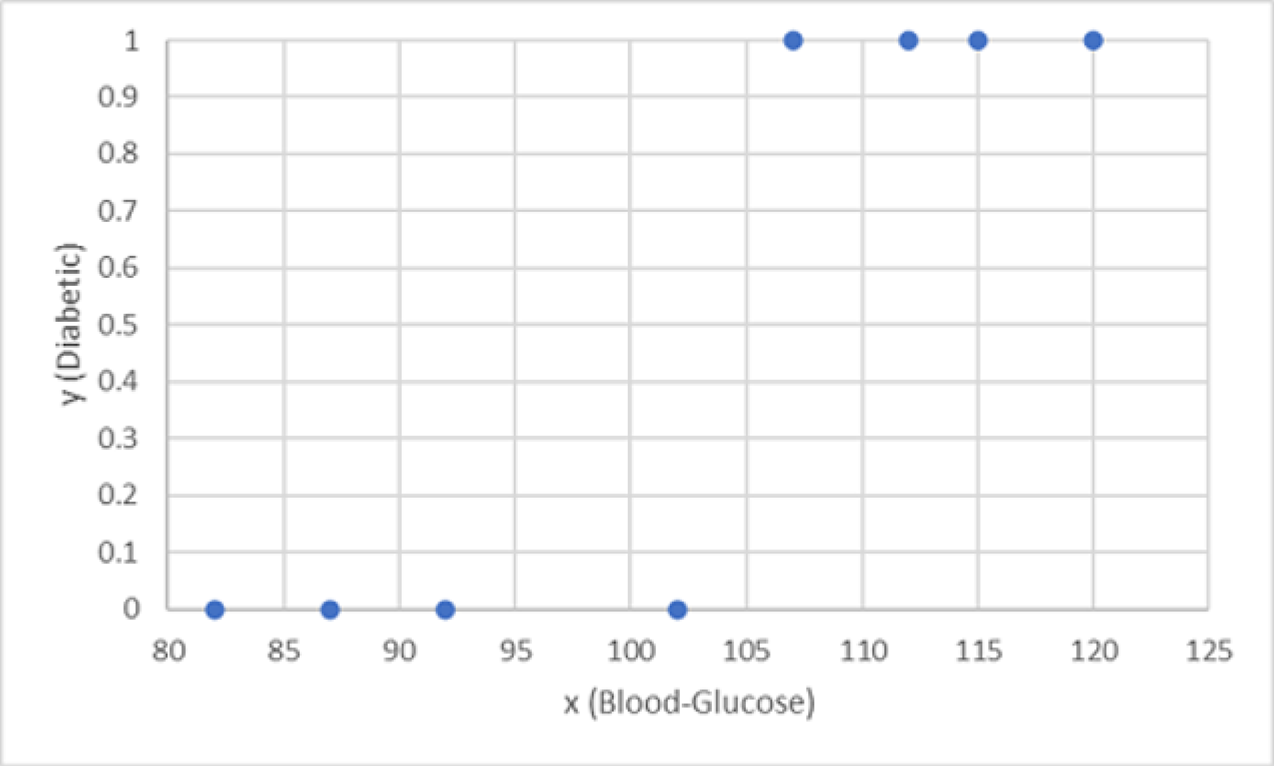
Diabetes classification. A simple example

- Suppose we have the following patient data, which consists of a single feature (blood glucose level) and a class label 0 for non-diabetic, 1 for diabetic.

Blood-Glucose	Diabetic
82	0
92	0
112	1
102	0
115	1
107	1
87	0

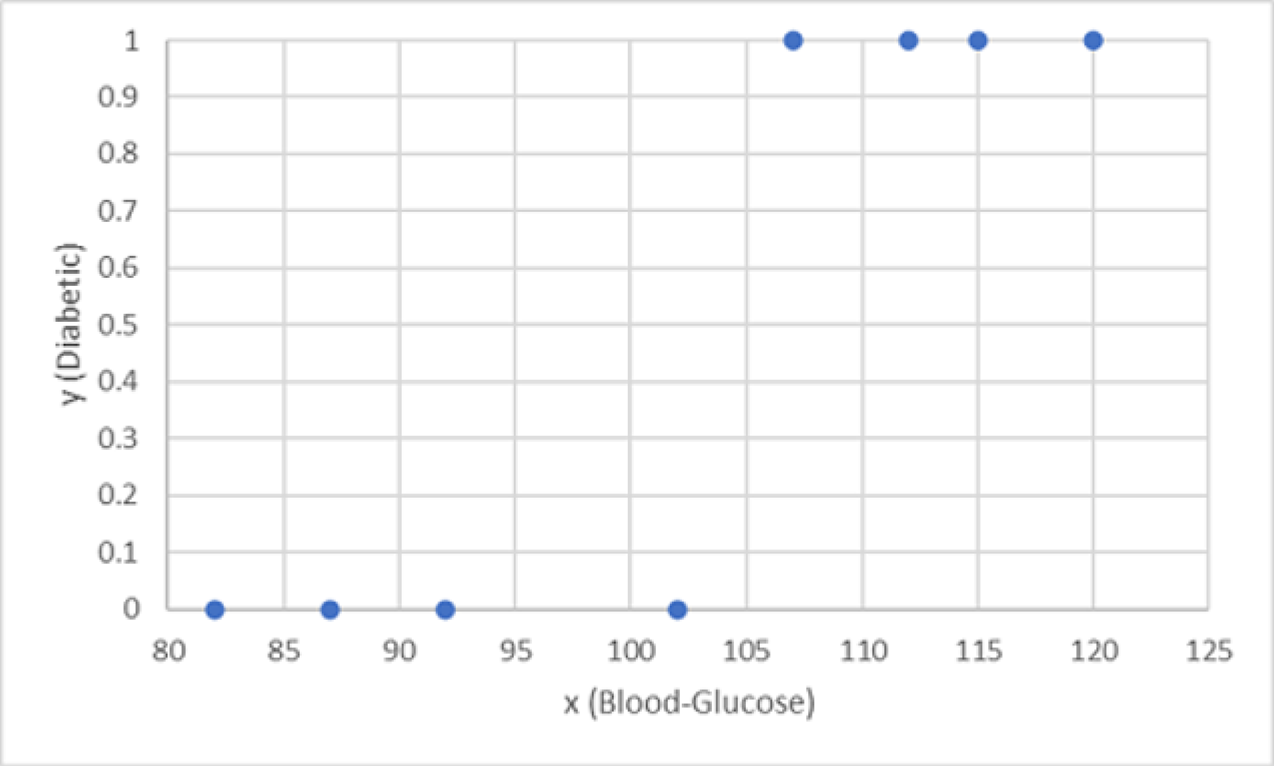
Diabetes classification. A simple example

- We use the first eight observations to train a classification model, and we start by plotting the blood glucose feature (***x***) and the predicted diabetic label (***y***).



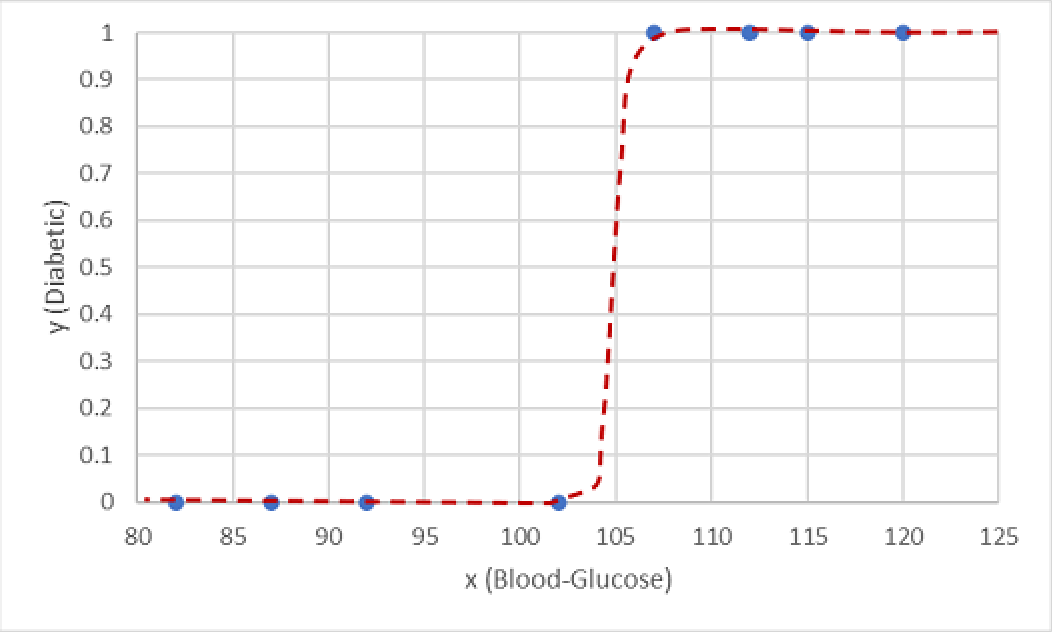
Diabetes classification. A simple example

- We use the first eight observations to train a classification model, and we start by plotting the blood glucose feature (***x***) and the predicted diabetic label (***y***).



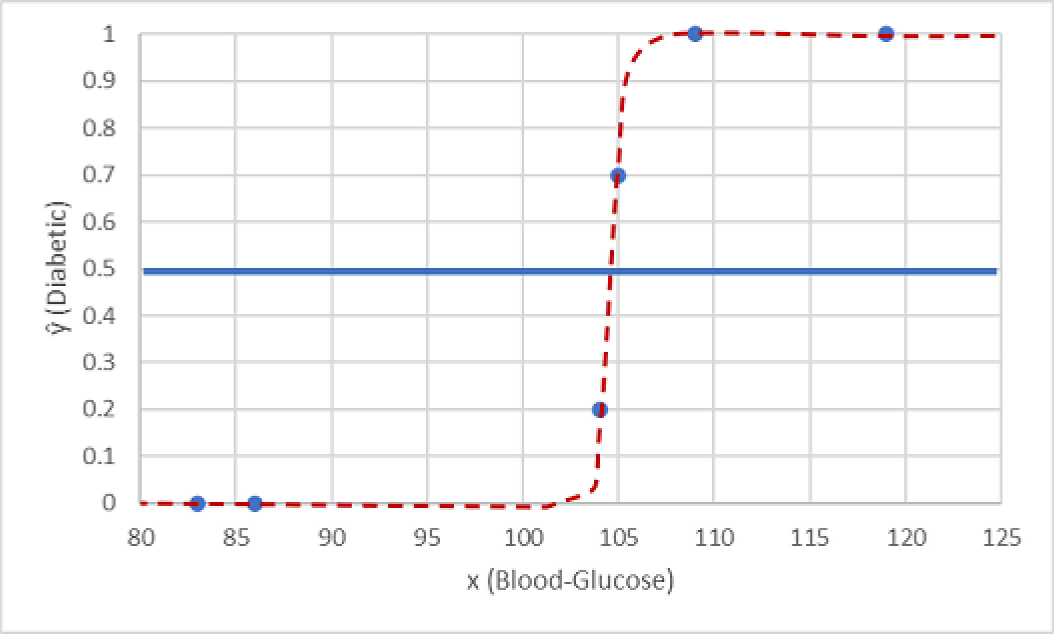
Diabetes classification. A simple example

- What we need is a function that calculates a probability value for ***y*** based on ***x*** (in other words, we need the function ***f(x) = y***).
- One such function is a *logistic* function, which forms a sigmoidal (S-shaped) curve.



Diabetes classification. A simple example

- Now we can use the function to calculate a probability value that ***y*** is positive, meaning the patient is diabetic, from any value of ***x*** by finding the point on the function line for ***x***. We can set a threshold value of 0.5 as the cut-off point for the class label prediction.
- Let's test it with the two data values we held back.



Diabetes classification. A simple example

- Points plotted below the threshold line yield a predicted class of 0 (non-diabetic) and points above the line are predicted as 1 (diabetic).
- Now we can compare the label predictions (***ŷ***, or "y-hat"), based on the logistic function encapsulated in the model, to the actual class labels (***y***).

x	y	ŷ
83	0	0
119	1	1
104	1	0
105	0	1
86	0	0
109	1	1

Diabetes classification. Example

- Dataset: diabetes dataset
- The diabetes dataset used in this exercise is based on data originally collected by the National Institute of Diabetes and Digestive and Kidney Diseases.
- Predicting the probability of diabetes, when a patient consult doctor for diagnosis operation.
- The data contains several attributes of the patients who is having such history.

Example.. Steps:

1. Import *pandas* and *numpy* libraries.
2. Use *read_csv* to load the dataset into DataFrame.
3. Identify the feture(s) (X) and outcome (Y) variable in the DataFrame for building the model.
4. Split the dataset into training and validation sets using *train_test_split().*
5. Import Scikit-Learn library and fit the model using **LogisticRegression** estimator.
6. Print model summary and conduct model diagnostics.

Diabetes classification. ...

- Reading and displaying few records from the dataset

```
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


# load the training dataset

!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning/main/Data/ml-basics/diabetes.csv
diabetes = pd.read_csv('diabetes.csv')
diabetes.head()
```

	PatientID	Pregnancies	PlasmaGlucose	DiastolicBloodPressure	TricepsThickness	SerumInsulin	BMI	DiabetesPedigree	Age	Diabetic
0	1354778	0	171	80	34	23	43.509726	1.213191	21	0
1	1147438	8	92	93	47	36	21.240576	0.158365	23	0
2	1640031	7	115	47	52	35	41.511523	0.079019	23	0
3	1883350	9	103	78	25	304	29.582192	1.282870	43	1
4	1424119	1	85	59	27	35	42.604536	0.549542	22	0

Diabetes classification. ...

- Creating Feature X and label y

```
# Separate features and labels
features = ['Pregnancies','PlasmaGlucose','DiastolicBloodPressure','TricepsThickness','SerumInsulin','BMI','DiabetesPedigree','Age']
label = 'Diabetic'
X, y = diabetes[features].values, diabetes[label].values
```

Diabetes classification. …

- Splitting the Dataset into Training and Validation Sets
- *train_test_split()* function from *skelearn.model_selection* module

provides the ability to split the dataset randomly into training and validation datasets.

- *train_test_split()* method returns four variable as below
    1. *train_X* contains X features of the training set
    2. *train_Y* contains the values of response variable for the training set
    3. *test_X* contains X features of the test set
    4. *test_Y* contains the values of response variable for the test set.

Diabetes classification...

- Splitting of data

```
# Split data 70%-30% into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=0)
print ('Training cases: %d\nTest cases: %d' % (X_train.shape[0], X_test.shape[0]))
```

```
Training cases: 10500
Test cases: 4500
```

Diabetes classification...

- Building model

```
# Train the model
from sklearn.linear_model import LogisticRegression

# Set regularization rate
reg = 0.01

# train a logistic regression model on the training set
model = LogisticRegression(C=1/reg, solver="liblinear").fit(X_train, y_train)
print(model)
```

```
LogisticRegression(C=100.0, solver='liblinear')
```

Evaluate the Trained Model

- Predicting on validation set to calculate accuracy

```
#Prediction
predictions = model.predict(X_test)
print('Predicted labels: ', predictions)
print('Actual labels:    ' ,y_test)
print('Accuracy: ', accuracy_score(y_test, predictions))
```

```
Predicted labels:  [0 0 0 ... 0 1 0]
Actual labels:     [0 0 1 ... 1 1 1]
Accuracy:  0.7888888888888889
```

Evaluate the Trained Model

- Predicting on validation set to calculate accuracy

```
#Prediction
predictions = model.predict(X_test)
print('Predicted labels: ', predictions)
print('Actual labels:    ' ,y_test)
print('Accuracy: ', accuracy_score(y_test, predictions))
```

```
Predicted labels:  [0 0 0 ... 0 1 0]
Actual labels:     [0 0 1 ... 1 1 1]
Accuracy:  0.7888888888888889
```

Evaluate the Trained Model

- Calculating the classification report to determine precision, recall F1-score etc..
- **Recall**: TP/(TP+FN) - of all the cases that *are* positive, how many did the model identify?(Quantity)
- **Precision**: TP/(TP+FP) - of all the cases that the model predicted to be positive, how many actually *are* positive?(Quality)
- F1-Score: It is collection of Recall and Precision