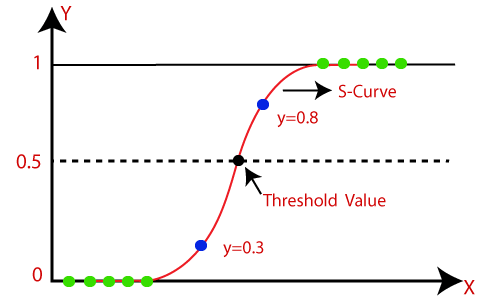
# Logistic Regression

* Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
* Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.
* Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.
* In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
* The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
* Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.
* Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



Ref: <https://www.javatpoint.com/logistic-regression-in-machine-learning>

## Logistic Function (Sigmoid Function):

* The sigmoid function is a mathematical function used to map the predicted values to probabilities.
* It maps any real value into another value within a range of 0 and 1.
* The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
* In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

## Assumptions for Logistic Regression:

* The dependent variable must be categorical in nature.
* The independent variable should not have multi-collinearity.

**Logistic Regression Equation:**

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

* We know the equation of the straight line can be written as:



* In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):



* But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:



The above equation is the final equation for Logistic Regression.

**Type of Logistic Regression:**

On the basis of the categories, Logistic Regression can be classified into three types:

* Binomial: In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
* Multinomial: In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
* Ordinal: In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

**Python Implementation of Logistic Regression (Binomial)**

To understand the implementation of Logistic Regression in Python, we will use the below example:

**Example:** There is a dataset given which contains the information of various users obtained from the social networking sites. There is a car making company that has recently launched a new SUV car. So the company wanted to check how many users from the dataset, wants to purchase the car.

For this problem, we will build a Machine Learning model using the Logistic regression algorithm. The dataset is shown in the below image. In this problem, we will predict the **purchased variable (Dependent Variable)** by using **age and salary (Independent variables)**.

# 

# Dataset

Reference : <https://www.javatpoint.com/logistic-regression-in-machine-learning>

**Steps in Logistic Regression:** To implement the Logistic Regression using Python, we will use the same steps as we have done in previous topics of Regression. Below are the steps:

* Data Pre-processing step
* Fitting Logistic Regression to the Training set
* Predicting the test result
* Test accuracy of the result (Creation of Confusion matrix)
* Visualizing the test set result.

**1. Data Pre-processing step:** In this step, we will pre-process/prepare the data so that we can use it in our code efficiently. It will be the same as we have done in Data pre-processing topic. The code for this is given below:

#Data Pre-procesing Step

# importing libraries

**import** numpy as nm

**import** matplotlib.pyplot as mtp

**import** pandas as pd

#importing datasets

data\_set= pd.read\_csv('user\_data.csv')

By executing the above lines of code, we will get the dataset as the output. Consider the given image:

# 

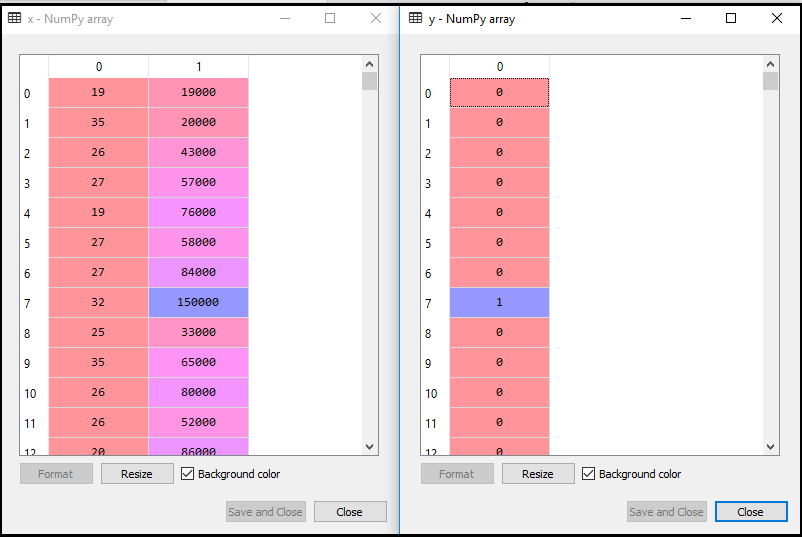
Now, we will extract the dependent and independent variables from the given dataset. Below is the code for it:

#Extracting Independent and dependent Variable

x= data\_set.iloc[:, [2,3]].values

y= data\_set.iloc[:, 4].values

In the above code, we have taken [2, 3] for x because our independent variables are age and salary, which are at index 2, 3. And we have taken 4 for y variable because our dependent variable is at index 4. The output will be:



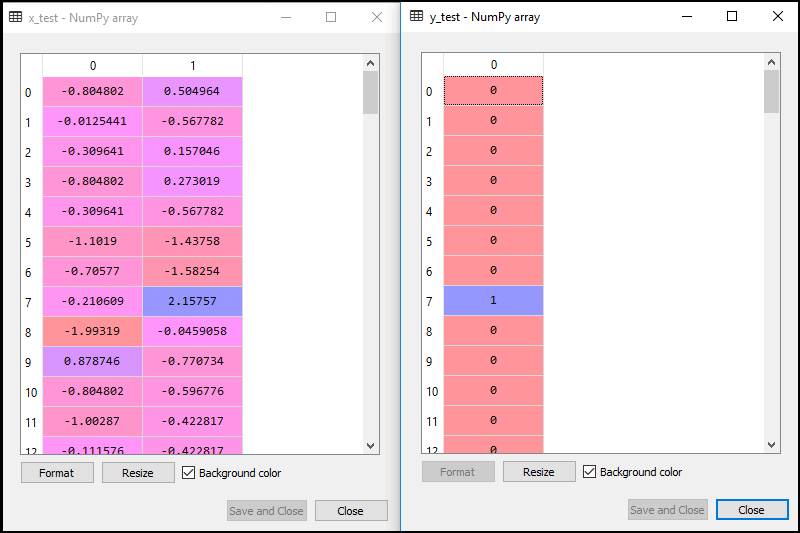
Now we will split the dataset into a training set and test set. Below is the code for it:

# Splitting the dataset into training and test set.

from sklearn.model\_selection **import** train\_test\_split

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

output for test set:



In logistic regression, we will do feature scaling because we want accurate result of predictions. Here we will only scale the independent variable because dependent variable have only 0 and 1 values. Below is the code for it:

#feature Scaling

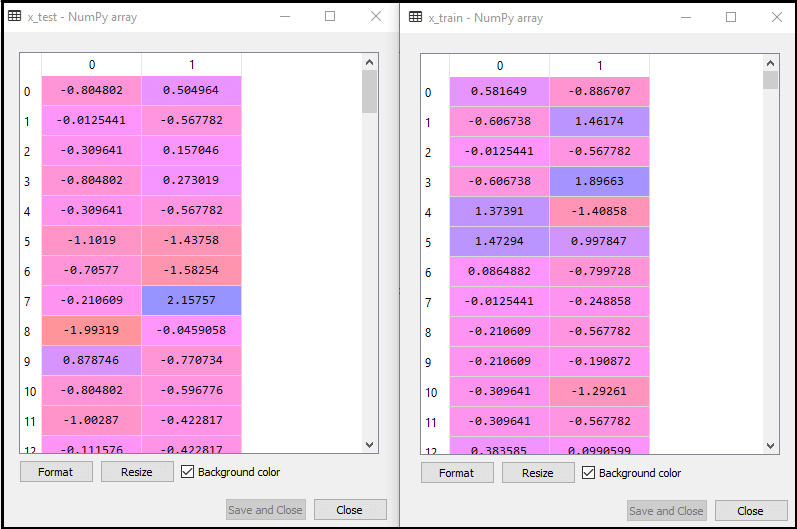
from sklearn.preprocessing **import** StandardScaler

st\_x= StandardScaler()

x\_train= st\_x.fit\_transform(x\_train)

x\_test= st\_x.transform(x\_test)

The scaled output is given below:



**2. Fitting Logistic Regression to the Training set:**

We have well prepared our dataset, and now we will train the dataset using the training set. For providing training or fitting the model to the training set, we will import the **LogisticRegression** class of the **sklearn** library.

After importing the class, we will create a classifier object and use it to fit the model to the logistic regression. Below is the code for it:

#Fitting Logistic Regression to the training set

from sklearn.linear\_model **import** LogisticRegression

classifier= LogisticRegression(random\_state=0)

classifier.fit(x\_train, y\_train)

**Output:** By executing the above code, we will get the below output:

**Out[5]:**

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

                   intercept\_scaling=1, l1\_ratio=None, max\_iter=100,

                   multi\_class='warn', n\_jobs=None, penalty='l2',

                   random\_state=0, solver='warn', tol=0.0001, verbose=0,

                   warm\_start=False)

Hence our model is well fitted to the training set.

**3. Predicting the Test Result**

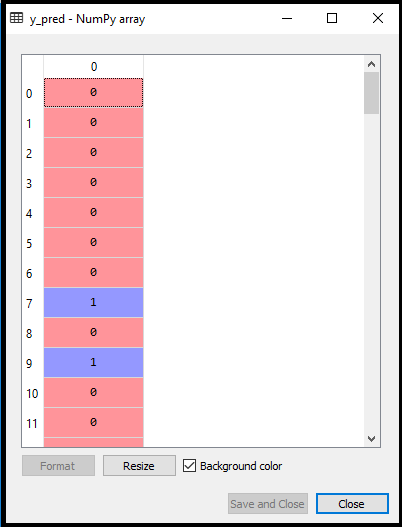
Our model is well trained on the training set, so we will now predict the result by using test set data. Below is the code for it:

#Predicting the test set result

y\_pred= classifier.predict(x\_test)

In the above code, we have created a y\_pred vector to predict the test set result.

**Output:** By executing the above code, a new vector (y\_pred) will be created under the variable explorer option. It can be seen as:



The above output image shows the corresponding predicted users who want to purchase or not purchase the car.

**4. Test Accuracy of the result**

Now we will create the confusion matrix here to check the accuracy of the classification. To create it, we need to import the **confusion\_matrix** function of the sklearn library. After importing the function, we will call it using a new variable **cm**. The function takes two parameters, mainly **y\_true**( the actual values) and **y\_pred** (the targeted value return by the classifier). Below is the code for it:

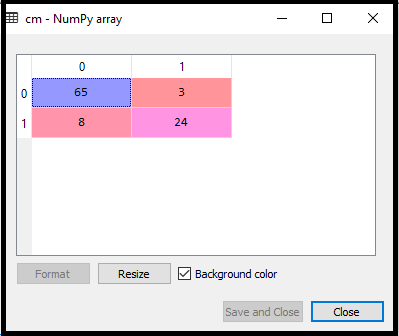
#Creating the Confusion matrix

from sklearn.metrics **import** confusion\_matrix

cm= confusion\_matrix()

**Output:**

By executing the above code, a new confusion matrix will be created. Consider the below image:



We can find the accuracy of the predicted result by interpreting the confusion matrix. By above output, we can interpret that 65+24= 89 (Correct Output) and 8+3= 11(Incorrect Output).

**5. Visualizing the training set result**

Finally, we will visualize the training set result. To visualize the result, we will use **ListedColormap** class of matplotlib library. Below is the code for it:

#Visualizing the training set result

from matplotlib.colors **import** ListedColormap

x\_set, y\_set = x\_train, y\_train

x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step  =0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('purple','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

**for** i, j in enumerate(nm.unique(y\_set)):

    mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

        c = ListedColormap(('purple', 'green'))(i), label = j)

mtp.title('Logistic Regression (Training set)')

mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

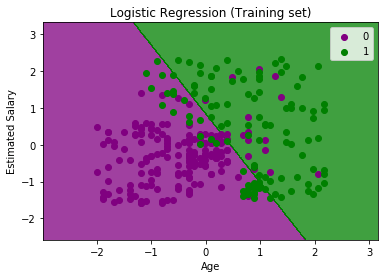
mtp.legend()

mtp.show()

In the above code, we have imported the **ListedColormap** class of Matplotlib library to create the colormap for visualizing the result. We have created two new variables **x\_set** and **y\_set** to replace **x\_train** and **y\_train**. After that, we have used the **nm.meshgrid** command to create a rectangular grid, which has a range of -1(minimum) to 1 (maximum). The pixel points we have taken are of 0.01 resolution.

To create a filled contour, we have used **mtp.contourf** command, it will create regions of provided colors (purple and green). In this function, we have passed the **classifier.predict** to show the predicted data points predicted by the classifier.

**Output:** By executing the above code, we will get the below output:



The graph can be explained in the below points:

* In the above graph, we can see that there are some **Green points** within the green region and **Purple points** within the purple region.
* All these data points are the observation points from the training set, which shows the result for purchased variables.
* This graph is made by using two independent variables i.e., **Age on the x-axis** and **Estimated salary on the y-axis**.
* The **purple point observations** are for which purchased (dependent variable) is probably 0, i.e., users who did not purchase the SUV car.
* The **green point observations** are for which purchased (dependent variable) is probably 1 means user who purchased the SUV car.
* We can also estimate from the graph that the users who are younger with low salary, did not purchase the car, whereas older users with high estimated salary purchased the car.
* But there are some purple points in the green region (Buying the car) and some green points in the purple region(Not buying the car). So we can say that younger users with a high estimated salary purchased the car, whereas an older user with a low estimated salary did not purchase the car.

**The goal of the classifier:**

We have successfully visualized the training set result for the logistic regression, and our goal for this classification is to divide the users who purchased the SUV car and who did not purchase the car. So from the output graph, we can clearly see the two regions (Purple and Green) with the observation points. The Purple region is for those users who didn't buy the car, and Green Region is for those users who purchased the car.

**Linear Classifier:**

As we can see from the graph, the classifier is a Straight line or linear in nature as we have used the Linear model for Logistic Regression. In further topics, we will learn for non-linear Classifiers.

**Visualizing the test set result:**

Our model is well trained using the training dataset. Now, we will visualize the result for new observations (Test set). The code for the test set will remain same as above except that here we will use **x\_test and y\_test** instead of **x\_train and y\_train**. Below is the code for it:

#Visulaizing the test set result

from matplotlib.colors **import** ListedColormap

x\_set, y\_set = x\_test, y\_test

x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step  =0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('purple','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

**for** i, j in enumerate(nm.unique(y\_set)):

    mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

        c = ListedColormap(('purple', 'green'))(i), label = j)

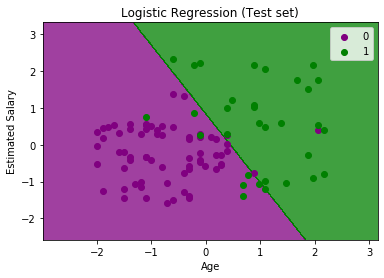
mtp.title('Logistic Regression (Test set)')

mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()



The above graph shows the test set result. As we can see, the graph is divided into two regions (Purple and Green). And Green observations are in the green region, and Purple observations are in the purple region. So we can say it is a good prediction and model. Some of the green and purple data points are in different regions, which can be ignored as we have already calculated this error using the confusion matrix (11 Incorrect output).

Hence our model is pretty good and ready to make new predictions for this classification problem.

# Evaluation- Confusion Matrix, Precision, Recall, F1 Score, Accuracy

model performance metrics that can be used to assess the model performance of a classification model.

Following Matrices will be used to assess the model performance of logistic regression classification model.

1. Confusion matrix
2. Precision
3. Recall
4. F1 Score
5. Accuracy

Let’s understand by example:

We’ll be using an example of a dataset having yes and no labels to be used to train a logistic regression model. This use case can be of any classification problem — spam detection, cancer prediction, attrition rate prediction, campaign target predictions, etc. We’ll be referring to special use-cases as and when required in this post. For now, we will take into consideration a simple logistic model which has to predict yes or no.

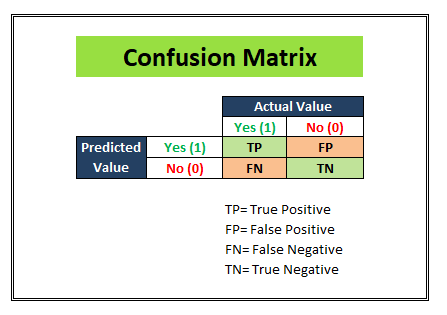
First things first, a logistic model can give two kinds of outputs:

1. It gives out class labels as output values (yes/no, 1/0, malignant/benign, attrited/retained, spam/not spam etc.)

2. It gives probability values between 0 to 1 as output values to signify how likely or how unlikely an event is for a particular observation.

The class labels scenario can be further segmented into the cases of balanced or imbalanced datasets, both of these cannot be judged/should not be judged basis on similar metrics. Some metrics are more suited for but not another and vice-versa. Similarly, the Probabilities scenario has different model performance metrics than the class labels one.

**1. Confusion Matrix**



Reference: <https://towardsdatascience.com/top-10-model-evaluation-metrics-for-classification-ml-models-a0a0f1d51b9>

We start with a development dataset while building any statistical or ML model. Divide that dataset into 2 parts: Training and Test. Keep aside the test dataset and train the model using the training dataset. Once the model is ready to predict, we try making predictions on the test dataset. And once we segment the results into a matrix similar to as shown in the above figure, we can see how much our model is able to predict right and how much of its predictions are wrong.

We populate the following 4 cells with the numbers from our test dataset(having 1000 observations for instance).

# 

Reference: <https://towardsdatascience.com/top-10-model-evaluation-metrics-for-classification-ml-models-a0a0f1d51b9>

1. **TP (True-positives):** Where the actual label for that column was “Yes” in the test dataset and our logistic regression model also predicted “Yes”. (500 observations)
2. **TN (True-negatives):**Where the actual label for that column was “No” in the test dataset and our logistic regression model also predicted “No”. (200 observations)
3. **FP (False-positives):** Where the actual label for that column was “No” in the test dataset but our logistic regression model predicted “Yes”. (100 observations)
4. **FN (False-negatives):** Where the actual label for that column was “Yes” in the test dataset but our logistic regression model predicted “No”. (200 observations)

# A confusion matrix is a table that is often used to **describe the performance of a classification model** (or “classifier”) on a set of test data for which the true values are known.

# 2. Accuracy

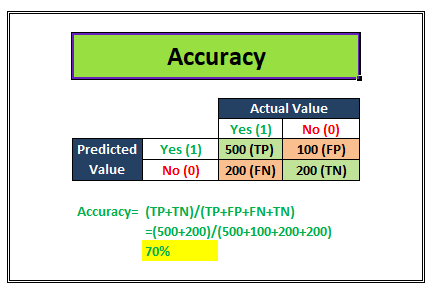
# Accuracy is a metric that is best used for a balanced dataset

# 

Reference: <https://medium.com/analytics-vidhya/what-is-balance-and-imbalance-dataset-89e8d7f46bc5>

As you can see, a balanced dataset is one where the 1’s and 0’s, yes’s and no’s, positive and negatives are equally represented by the training data. On the other hand, if the ratio of the two class-labels is skewed then our model will get biased towards one category.

Assuming we have a Balanced dataset, let’s learn what is Accuracy.



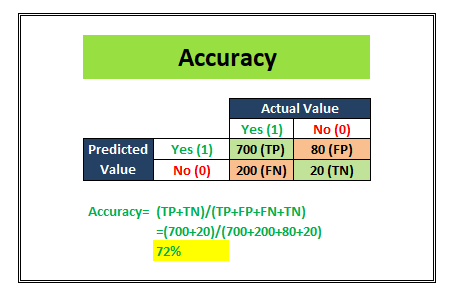
Accuracy is the proximity of measurement results to the true value. It tell us how accurate our classification model is able to predict the class labels given in the problem statement.

For example: Let’s suppose that our classification model is trying to predict for customer attrition scenario. In the image above, Out of the total 700 actually attrited customers (TP+FN) , the model was correctly able to classify 500 attrited customers correctly (TP). Similarly, out of the total 300 retained customers (FP+TN), the model was correctly able to classify 200 retained customers correctly (TN).

Accuracy= (TP+TN)/Total customers

In the above scenario, we see that the accuracy of the model on the test dataset of 1000 customers is 70%.

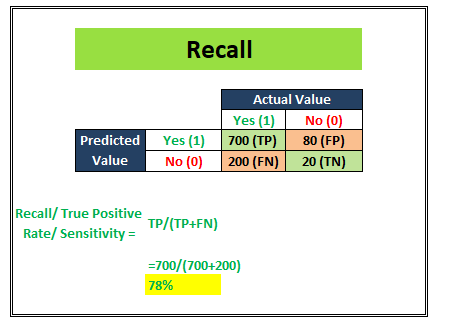
Now, we learned that Accuracy is a metric that should be used only for a balanced dataset. Why is that so? Let’s look at an example to understand that.



In this example, this model was trained on an imbalanced dataset and even the test dataset is imbalanced. The Accuracy metric has a score of 72% which might give us the impression that our model is doing a good job at the classification. But, look closer, this model is doing a terrible job out of predicting the Negative class labels. It only predicted 20 correct outcomes out of 100 total negative label observations. This is why the Accuracy metric should not be used if you have an imbalanced dataset.

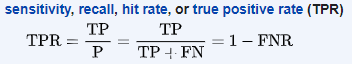
The next question is, then what is to be used if you have an imbalanced dataset? The answer is Recall and Precision. Let’s learn more about these.

# 3. Recall/ Sensitivity/ TPR



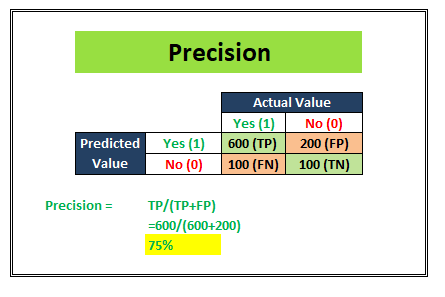
**Recall/ Sensitivity/ TPR (True Positive Rate)**attempts to answer the following question:

What proportion of actual positives was identified correctly?



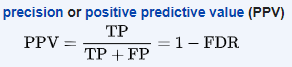
This metric gives us 78% as the Recall score in the above image. **Recall is generally used in use cases where the truth-detection is of utmost importance.**For example: The cancer prediction, the stock market classification, etc. over here the problem statement requires that the False negatives be minimized which implies Recall/Sensitivity be maximized.

# Precision



**Precision** attempts to answer the following question:

**What proportion of positive identifications was actually correct?**



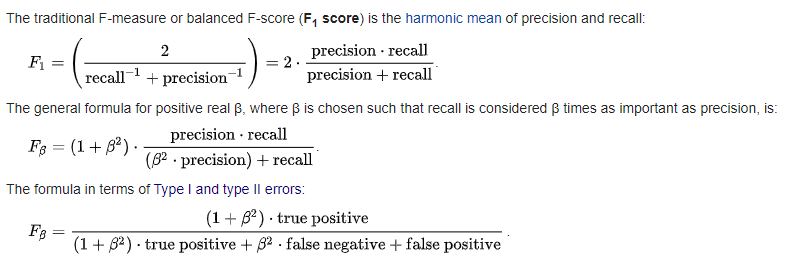
The example shown in the above image shows us that the Precision score is 75%. Precision is generally used in cases where it’s of utmost importance not to have a high number of False positives. For example: In spam detection cases, as we discussed above, a false positive would be an observation that was not spam but was classified as Spam by our classification model. Too many of the false positives can defeat the purpose of a spam classifier model. Thus, Precision comes handy here to judge the model performance in this scenario.

1. **F1 Score**

We talked about Recall and Precision in points numbers 6 and 7 respectively. We understand that there are some problem statements where a higher Recall takes precedence over a higher Precision and vice-versa.

But there are some use-cases, where the distinction is not very clear and as developers, we want to give importance to both Recall and Precision. In this case, there is another metric- F1 Score that can be used. It is dependent on both Precision and Recall.

In a statistical analysis of binary classification, the **F1 score** (also **F-score** or **F-measure**) is a measure of a test’s accuracy. It considers both the precision p and the recall r of the test to compute the score



# Python Library: Sci-Kit Learn

**What is Scikit-Learn?**

Open-source ML library for Python. Built on NumPy, SciPy, and Matplotlib.



[Scikit-learn](http://scikit-learn.org/stable/) is a library in Python that provides many unsupervised and supervised learning algorithms. It’s built upon some of the technology you might already be familiar with, like NumPy, pandas, and Matplotlib!

The functionality that scikit-learn provides include:

* **Regression**, including Linear and Logistic Regression
* **Classification**, including K-Nearest Neighbors
* **Clustering**, including K-Means and K-Means++
* **Model selection**
* **Preprocessing**, including Min-Max Normalization

**Prerequisites**

Before we start using scikit-learn latest release, we require the following −

* Python (>=3.5)
* NumPy (>= 1.11.0)
* Scipy (>= 0.17.0)li
* Joblib (>= 0.11)
* Matplotlib (>= 1.5.1) is required for Sklearn plotting capabilities.
* Pandas (>= 0.18.0) is required for some of the scikit-learn examples using data structure and analysis.

## Installation

If you already installed NumPy and Scipy, following are the two easiest ways to install scikit-learn −

## Using pip

Following command can be used to install scikit-learn via pip −

pip install -U scikit-learn

## Using conda

Following command can be used to install scikit-learn via conda −

conda install scikit-learn

On the other hand, if NumPy and Scipy is not yet installed on your Python workstation then, you can install them by using either **pip** or **conda**.

Another option to use scikit-learn is to use Python distributions like ***Canopy*** and ***Anaconda*** because they both ship the latest version of scikit-learn.

**Features of scikit-learn**

Rather than focusing on loading, manipulating and summarising data, Scikit-learn library is focused on modeling the data. Some of the most popular groups of models provided by Sklearn are as follows −

**Supervised Learning algorithms** − Almost all the popular supervised learning algorithms, like Linear Regression, Support Vector Machine (SVM), Decision Tree etc., are the part of scikit-learn.

**Unsupervised Learning algorithms** − On the other hand, it also has all the popular unsupervised learning algorithms from clustering, factor analysis, PCA (Principal Component Analysis) to unsupervised neural networks.

**Clustering** − This model is used for grouping unlabeled data.

**Cross Validation** − It is used to check the accuracy of supervised models on unseen data.

**Dimensionality Reduction** − It is used for reducing the number of attributes in data which can be further used for summarisation, visualisation and feature selection.

**Ensemble methods** − As name suggest, it is used for combining the predictions of multiple supervised models.

**Feature extraction** − It is used to extract the features from data to define the attributes in image and text data.

**Feature selection** − It is used to identify useful attributes to create supervised models.

**Open Source** − It is open source library and also commercially usable under BSD license.

**Dataset Loading**

A collection of data is called dataset. It is having the following two components −

**Features** − The variables of data are called its features. They are also known as predictors, inputs or attributes.

* **Feature matrix** − It is the collection of features, in case there are more than one.
* **Feature Names** − It is the list of all the names of the features.

**Response** − It is the output variable that basically depends upon the feature variables. They are also known as target, label or output.

* **Response Vector** − It is used to represent response column. Generally, we have just one response column.
* **Target Names** − It represent the possible values taken by a response vector.

Scikit-learn have few example datasets like **iris** and **digits** for classification and the **Boston house prices** for regression.

**Example**

Following is an example to load **iris** dataset −

from sklearn.datasets import load\_iris

iris = load\_iris()

X = iris.data

y = iris.target

feature\_names = iris.feature\_names

target\_names = iris.target\_names

print("Feature names:", feature\_names)

print("Target names:", target\_names)

print("\nFirst 10 rows of X:\n", X[:10])

Output

Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

Target names: ['setosa' 'versicolor' 'virginica']

First 10 rows of X:

[

[5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

[4.6 3.1 1.5 0.2]

[5. 3.6 1.4 0.2]

[5.4 3.9 1.7 0.4]

[4.6 3.4 1.4 0.3]

[5. 3.4 1.5 0.2]

[4.4 2.9 1.4 0.2]

[4.9 3.1 1.5 0.1]

]

**Splitting the dataset**

To check the accuracy of our model, we can split the dataset into two pieces-**a training set** and **a testing set**. Use the training set to train the model and testing set to test the model. After that, we can evaluate how well our model did.

**Example**

The following example will split the data into 70:30 ratio, i.e. 70% data will be used as training data and 30% will be used as testing data. The dataset is iris dataset as in above example.

from sklearn.datasets import load\_iris

iris = load\_iris()

X = iris.data

y = iris.target

from sklearn.model\_selection import train\_test\_split

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

X, y, test\_size = 0.3, random\_state = 1

)

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

Output

(105, 4)

(45, 4)

(105,)

(45,)

As seen in the example above, it uses **train\_test\_split()** function of scikit-learn to split the dataset. This function has the following arguments −

* **X, y** − Here, **X** is the **feature matrix** and y is the **response vector**, which need to be split.
* **test\_size** − This represents the ratio of test data to the total given data. As in the above example, we are setting **test\_data = 0.3** for 150 rows of X. It will produce test data of 150\*0.3 = 45 rows.
* **random\_size** − It is used to guarantee that the split will always be the same. This is useful in the situations where you want reproducible results.

**Train the Model**

Next, we can use our dataset to train some prediction-model. As discussed, scikit-learn has wide range of **Machine Learning (ML) algorithms** which have a consistent interface for fitting, predicting accuracy, recall etc.

Example

In the example below, we are going to use KNN (K nearest neighbors) classifier. Don’t go into the details of KNN algorithms, as there will be a separate chapter for that. This example is used to make you understand the implementation part only.

from sklearn.datasets import load\_iris

iris = load\_iris()

X = iris.data

y = iris.target

from sklearn.model\_selection import train\_test\_split

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

X, y, test\_size = 0.4, random\_state=1

)

from sklearn.neighbors import KNeighborsClassifier

from sklearn import metrics

classifier\_knn = KNeighborsClassifier(n\_neighbors = 3)

classifier\_knn.fit(X\_train, y\_train)

y\_pred = classifier\_knn.predict(X\_test)

# Finding accuracy by comparing actual response values(y\_test)with predicted response value(y\_pred)

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

# Providing sample data and the model will make prediction out of that data

sample = [[5, 5, 3, 2], [2, 4, 3, 5]]

preds = classifier\_knn.predict(sample)

pred\_species = [iris.target\_names[p] for p in preds] print("Predictions:", pred\_species)

Output:

Accuracy: 0.9833333333333333

Predictions: ['versicolor', 'virginica']

There will be many more function in scikit-learn to explore more Machine learning algorithms. Following some important features, you can use for machine learning model.

**Linear Regression**

This supervised ML model is used when the output variable is continuous and it follows linear relation with dependent variables. It can be used to forecast sales in the coming months by analyzing the sales data for previous months.

 With the help of sklearn, we can easily implement the Linear Regression model as follows:

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

regression\_model = LinearRegression()

regression\_model.fit(x\_train, y\_train)

y\_predicted = regression\_model.predict(x\_test)

rmse = mean\_squared\_error(y\_test, y\_predicted)

r2 = r2\_score(y\_test, y\_predicted)

LinerRegression() creates an object of linear regression. Then we fit the model on the training set. Finally, we predicted the model on the test dataset. “rmse” and “r\_score” can be used to check the accuracy of the model.

More function you can explore with machine learning examples.

References

1. <https://www.javatpoint.com/logistic-regression-in-machine-learning>
2. <https://towardsdatascience.com/top-10-model-evaluation-metrics-for-classification-ml-models-a0a0f1d51b9>
3. <https://www.tutorialspoint.com/scikit_learn/scikit_learn_introduction.htm>
4. <https://scikit-learn.org/stable/tutorial/basic/tutorial.html>
5. <https://www.analyticsvidhya.com/blog/2021/07/15-most-important-features-of-scikit-learn/#:~:text=Through%20scikit%2Dlearn%2C%20we%20can,ensemble%20techniques%2C%20and%20inbuilt%20datasets>.