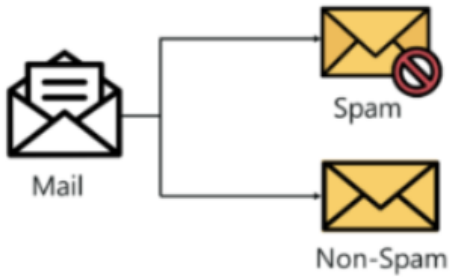
Introduction to Classification

Classification in machine learning and statistics is a supervised learning approach in which the computer program learns from the data given to it and make new observations or classifications.

What is Classification In Machine Learning ?

Classification is a process of categorizing a given set of data into classes, It can be performed on both structured or unstructured data. The process starts with predicting the class of given data points. The classes are often referred to as target, label or categories. The classification predictive modeling is the task of approximating the mapping function from input variables to discrete output variables. The main goal is to identify which class/category the new data will fall into.



Classification Terminologies In Machine Learning

- **Classifier** – It is an algorithm that is used to map the input data to a specific category.
- **Classification Model** – The model predicts or draws a conclusion to the input data given for training, it will predict the class or category for the data.
- **Feature** – A feature is an individual measurable property of the phenomenon being observed.
- **Binary Classification** – It is a type of classification with two outcomes, for eg – either true or false.
- **Multi-Class Classification** – The classification with more than two classes, in multi-class classification each sample is assigned to one and only one label or target.
- **Multi-label Classification** – This is a type of classification where each sample is assigned to a set of labels or targets.
- **Initialize** – It is to assign the classifier to be used.
- **Train the Classifier** – Each classifier in sci-kit learn uses the fit(X, y) method to fit the model for training the train X and train label y.
- **Predict the Target** – For an unlabeled observation X, the predict(X) method returns predicted label y.
- **Evaluate** – This basically means the evaluation of the model i.e classification report, accuracy score, etc.

Types Of Learners In Classification:

- **Lazy Learners** – Lazy learners simply store the training data and wait until a testing data appears. The classification is done using the most related data in the stored training data. They have more predicting time compared to eager learners. Eg – k-nearest neighbor, case-based reasoning.

- **Eager Learners** – Eager learners construct a classification model based on the given training data before getting data for predictions. It must be able to commit to a single hypothesis that will work for the entire space. Due to this, they take a lot of time in training and less time for a prediction. Eg – Decision Tree, Naive Bayes, Artificial Neural Networks.

Building a Classifier in Python

Scikit-learn, a Python library for machine learning can be used to build a classifier in Python. The steps for building a classifier in Python are as follows −

Step 1: Importing necessary python package

For building a classifier using scikit-learn, we need to import it. We can import it by using following script –

```
import sklearn
```

Step 2: Importing dataset

After importing necessary package, we need a dataset to build classification prediction model. We can import it from sklearn dataset or can use other one as per our requirement. We are going to use sklearn's Breast Cancer Wisconsin Diagnostic Database. We can import it with the help of following script –

```
data = load_breast_cancer()
```

We also need to organize the data and it can be done with the help of following scripts –

```
label_names = data['target_names']
labels = data['target']
feature_names = data['feature_names']
features = data['data']
```

The following command will print the name of the labels, **'malignant'** and **'benign'** in case of our database.

```
print(label_names)
```

The output of the above command is the names of the labels –

```
['malignant' 'benign']
```

These labels are mapped to binary values 0 and 1. **Malignant** cancer is represented by 0 and **Benign** cancer is represented by 1.

The feature names and feature values of these labels can be seen with the help of following commands

```
print(feature_names[0])
```

–

The output of the above command is the names of the features for label 0 i.e. **Malignant** cancer−

```
mean radius
```

Similarly, names of the features for label can be produced as follows −

```
print(features[0])
```

This will give the following output –

```
[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
 1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
 6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
 1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
 4.601e-01 1.189e-01]
```

Step 3: Organizing data into training & testing sets

As we need to test our model on unseen data, we will divide our dataset into two parts: a training set and a test set. We can use train_test_split() function of sklearn python package to split the data into sets. The following command will import the function –

```
from sklearn.model_selection import train_test_split
```

Now, next command will split the data into training & testing data. In this example, we are using taking 40 percent of the data for testing purpose and 60 percent of the data for training purpose −

```
train, test, train_labels, test_labels =
   train_test_split(features,labels,test_size = 0.40, random_state = 42)
```

Step 4: Model evaluation

After dividing the data into training and testing we need to build the model. We will be using *Naïve* Bayes algorithm for this purpose. The following commands will import the GaussianNB module –

```
from sklearn.naive_bayes import GaussianNB
```

```
gnb = GaussianNB()
```

```
model = gnb.fit(train, train_labels)
```

Next, with the help of following command we can train the model –

```
model = gnb.fit(train, train_labels)
```

```
preds = gnb.predict(test)
print(preds)
```

Output

```
[1 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0
 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 1 0 0 1 1 1 0 0 1 1 0 0 1 0
 1 1 1 1 1 1 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0
 1 1 0 0 0 1 1 1 0 0 1 1 0 1 0 0 1 1 0 0 0 1 1 1 0 1 1 0 0 1 0 1 1 0 1 0 0
 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 0 0
 0 1 1 0 1 0 1 1 1 1 0 1 1 0 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1
 0 0 1 1 0 1]
```

The above series of 0s and 1s in output are the predicted values for the **Malignant** and **Benign** tumor classes.

Step 5: Finding accuracy

We can find the accuracy of the model build in previous step by comparing the two arrays namely test_labels and preds. We will be using the accuracy_score() function to determine the accuracy.

```
from sklearn.metrics import accuracy_score
print(accuracy_score(test_labels,preds))
0.951754385965
```

Classification Evaluation Metrics

The job is not done even if you have finished implementation of your Machine Learning application or model. We must have to find out how effective our model is? There can be different evaluation metrics, but we must choose it carefully because the choice of metrics influences how the performance of a machine learning algorithm is measured and compared.

The following are some of the important classification evaluation metrics among which you can choose based upon your dataset and kind of problem

Confusion Matrix

It is the easiest way to measure the performance of a classification problem where the output can be of two or more type of classes.

The followings are some important ML classification algorithms –

- Logistic Regression
- Support Vector Machine (SVM)
- Decision Trees
- Naïve Bayes
- Random Forest

Applications

Some of the most important applications of classification algorithms are as follows :

- Speech Recognition
- Handwriting Recognition
- Biometric Identification
- Document Classifier