Data Science with Python Programming

- Presentation By Uplatz
- Contact us: https://training.uplatz.com
- Email: info@uplatz.com
- Phone: +44 7836 212635

Classification

Learning outcomes:

- What is Classification?
- Classification algorithms
- Logistic Regression
- Implementing Logistic Regression
- Decision Tree
- Implementing Decision Tree
- Support Vector Machine (SVM)
- Implementing SVM

What is Classification?

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.

What is Classification?

An easy to understand example is classifying emails as "*spam*" or "*not spam*."

Let us try to understand classification with a simple example. **Heart disease detection** can be identified as a **classification problem**, this is a binary classification since there can be only two classes i.e. has heart disease or does not have heart disease. The classifier, in this case, needs training data to understand how the given input variables are related to the class. And once the classifier is trained accurately, it can be used to detect whether heart disease is there or not for a particular patient.

What is Classification?

Classification is again divided into three other categories or problems which are: ***Binary classification***, ***Multi-class/Multinomial classification*** *and* ***Multi-label classification***.

Binary classification: Predicting whether an email is Spam or not are Binary classification problems.
Multi-class/Multinomial classification: classifying news into different categories (sports /   entertainment /political).
Multi-label classification: Multi-label classification is a predictive modeling task that involves predicting zero or more mutually non-exclusive class labels.

Classification algorithms

There is a lot of classification algorithms available now but it is not possible to conclude which one is superior to other. It depends on the application and nature of available data set. Some of the important classification algorithms are:

Logistic Regression

Decision Tree

Support Vector Machines

K-Nearest Neighbors

Naive Bayes Classifier

Random Forest

Let us take a look at the first classification algorithms in machine learning.

Logistic Regression

Logistic regression is analogous to linear regression but tries to predict a categorical or discrete target field instead of a numeric one.
Logistic Regression is a **classification** and **not** a regression algorithm. It estimates discrete values **(Binary values like 0/1, yes/no, true/false)** based on a given set of an independent variable(s). Simply put, it basically predicts the probability of occurrence of an event by fitting data to a ***logit function*.** Hence it is also known as **logit regression**. The values obtained would always lie within 0 and 1 since it predicts the probability.

Logistic Regression

We use logistic regression for the binary classification of data-points. We perform categorical classification such that an output belongs to either of the two classes (1 or 0). **For example –** we can predict whether it will rain today or not, based on the current weather conditions. The goal of logistic regression is to find a best-fitting relationship between the dependent variable and a set of independent variables. It is better than other binary classification algorithms like nearest neighbour since it quantitatively explains the factors leading to classification.

Implementing Logistic Regression

Example:

Let’s now see how to apply logistic regression in Python using a practical example.

Our goal is to build a logistic regression model in Python in order to determine whether candidates would get admitted to a prestigious university based on 'gmat‘ score, ‘gpa', and 'work_experience’

Implementing Logistic Regression

Step 1: Import the needed Python packages
Before you start, make sure that the following packages are installed in Python:

pandas – used to create the DataFrame to capture the dataset in Python
sklearn – used to build the logistic regression model in Python
matplotlib – used to display charts

Implementing Logistic Regression

Step 1: Import the needed Python packages
You'll then need to import all the packages as follows:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
```

Here `from sklearn.model_selection import train_test_split` is useful to split arrays or matrices into random train and test subsets.

Implementing Logistic Regression

Step 2: Gather your data

Let’s look at the dataset:

Here, there are two possible outcomes: **Admitted** (represented by the value of ‘1’) vs. **Rejected** (represented by the value of ‘0’). You can then build a logistic regression in Python, where:

- The 3 independent variables are the GMAT score, GPA and Years of work experience
- The dependent variable represents whether a person gets admitted or not.

Implementing Logistic Regression

Step 3:
Here, you need to divide the given columns into two types of variables, **dependent(or target variable or response variable)** and **independent variable(or feature variables).**
Now, set the independent variables (represented as X) and the dependent variable (represented as y):

```
X = df[['gmat', 'gpa','work_experience']]
y = df['admitted']
```

Implementing Logistic Regression

Step 4: 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.

1) **Features(independent variable),**

2) target, and

3) test_size.

Additionally, you can use **random_state** to select records randomly.

Implementing Logistic Regression

Step 4: Splitting Data

```
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**.

Implementing Logistic Regression

Step 5: Model Development and Prediction

First, import the Logistic Regression module and create a Logistic Regression classifier object using **LogisticRegression()** function.

Then, fit your model on the train set using **fit()** and perform prediction on the test set using **predict().**

```
logistic_regression= LogisticRegression()
logistic_regression.fit(X_train,y_train)
y_pred=logistic_regression.predict(X_test)
```

Implementing Logistic Regression

Step 6: Model Evaluation using Confusion Matrix
A confusion matrix is a table that is used to evaluate the performance of a classification model. You 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.

```
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test,
y_pred)
print(cnf_matrix)
```

Implementing Logistic Regression

Step 6: Model Evaluation using Confusion Matrix
In the case of binary classification, the confusion matrix shows the numbers of the following:
True negatives in the upper-left position: We predicted no, and they don’t get admission.
False negatives in the lower-left position: We predicted no, but they actually get the admission.
False positives in the upper-right position: We predicted yes, but they don’t get the admission.
True positives in the lower-right position: These are cases in which we predicted and they get the admission.

Implementing Logistic Regression

Step 6: Model Evaluation using Confusion Matrix
Here, you can see the confusion matrix in the form of the array object. The dimension of this matrix is 2*2 because this model is binary classification. You have two classes 0 and 1. Diagonal values represent accurate predictions, while non-diagonal elements are inaccurate predictions. In the output, 4 and 4 are actual predictions, and 1 and 1 are incorrect predictions.
This means out of 10, we got 8 correct prediction and 2 incorrect prediction.

Implementing Logistic Regression

Step 7: Visualizing Confusion Matrix using Heatmap

Let's visualize the results of the model in the form of a confusion matrix using matplotlib and seaborn.

```
import seaborn as sn
%matplotlib inline
sn.heatmap(pd.DataFrame(cnf_matrix))
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Implementing Logistic Regression

Step 8: Let's evaluate the model using model evaluation metrics such as accuracy.

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Accuracy: Overall, how often is the classifier correct.
Well, you got a classification rate of 80%, considered as good accuracy.

Implementing Logistic Regression

Let's predict whether the candidate with **gmat** score of 690, GPA of 3.8 and work experience 3 years will get the admission into the university or not.

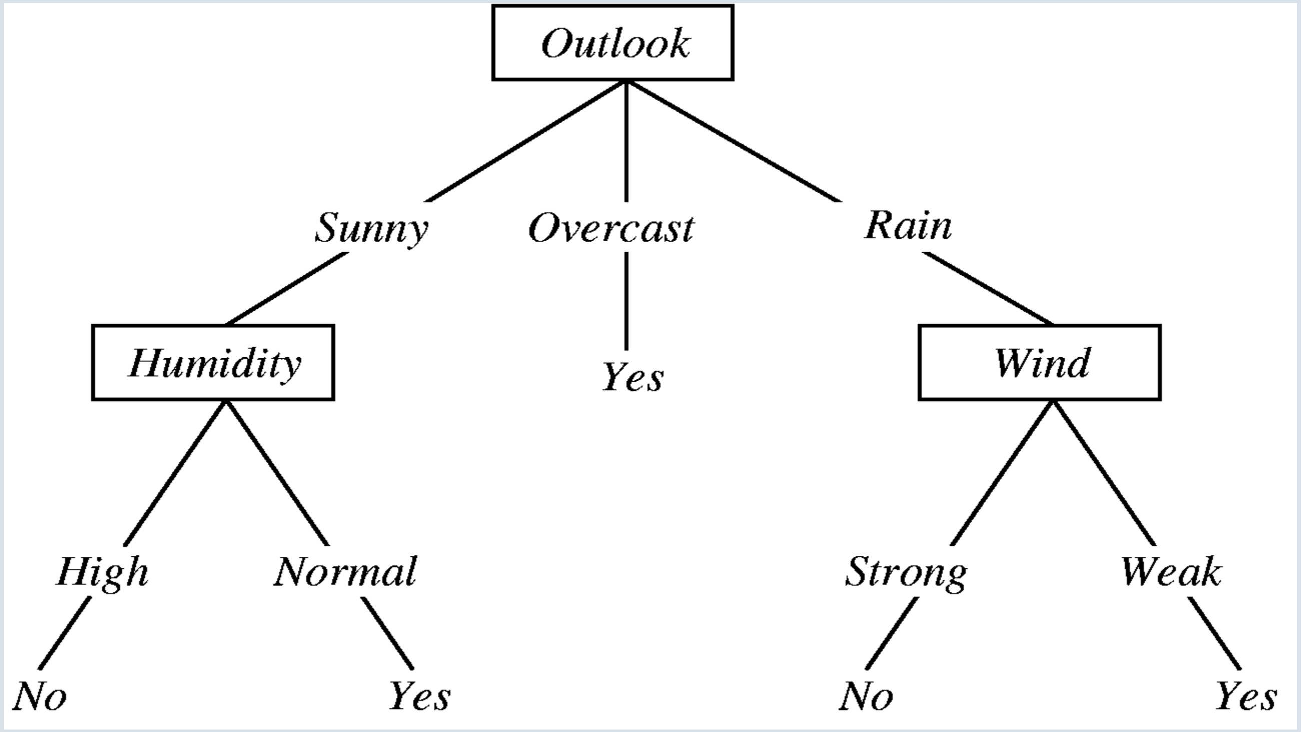
```
adm = logistic_regression.predict([[690,3.8,3]])
print(adm)
```

Decision Tree

Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving **Classification** problems. Decision Tree is one of the most powerful and popular algorithm. A decision tree is a flowchart-like structure in which each internal node represents a test on a feature (e.g. whether a coin flip comes up heads or tails) , each leaf node represents a class label (decision taken after computing all features) and branches represent conjunctions of features that lead to those class labels.

Decision Tree

The paths from root to leaf represent classification rules. Below diagram illustrate the basic flow of decision tree for decision making with labels (Rain(Yes), No Rain(No)).



Decision Tree

While making decision tree, at each node of tree we ask different type of questions. Based on the asked question we will calculate the information gain corresponding to it.
Tree models where the target variable can take a discrete set of values are called **classification trees**.
Decision trees where the target variable can take continuous values (typically real numbers) are called **regression trees**.

Implementing Decision Tree

Now we will implement the Decision tree using Python. We'll predict whether a person has diabetes or not, based on information about the patient such as blood pressure, body mass index (BMI), age, etc.

The data was collected and made available by "National Institute of Diabetes and Digestive and Kidney Diseases" as part of the Pima Indians Diabetes Database. Several constraints were placed on the selection of these instances from a larger database.

Implementing Decision Tree

Importing Required Libraries

Let's first load the required libraries.

```
import pandas as pd
# Import Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
# Import train_test_split function
from sklearn.model_selection import
train_test_split
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
```

Implementing Decision Tree

Loading Data

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

```
df = pd.read_csv('F:\Alison R Studio\Diabetes.csv')
df.head()
```

Implementing Decision Tree

Feature Selection

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

```
X = df[['Pregnancies', 'Insulin', 'BMI',
'Age','Glucose','BloodPressure','DiabetesPedigreeF
unction']]# Features

y = df['Outcome'] # Target variable
```

Implementing Decision Tree

Splitting Data

To understand model performance, dividing the dataset into a training set and a test set is a good strategy.

Let's split the dataset by using function **train_test_split().** You need to pass 4 parameters features, target, test_set size, and random_sate.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

Implementing Decision Tree

Building Decision Tree Model

Let's create a Decision Tree Model using Scikit-learn.

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()
# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

Implementing Decision Tree

Evaluating Model

Let's estimate, how accurately the classifier or model can predict the type of cultivars.

Accuracy can be computed by comparing actual test set values and predicted values.

```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Well, you got a classification rate of 75.00%, considered as good accuracy.

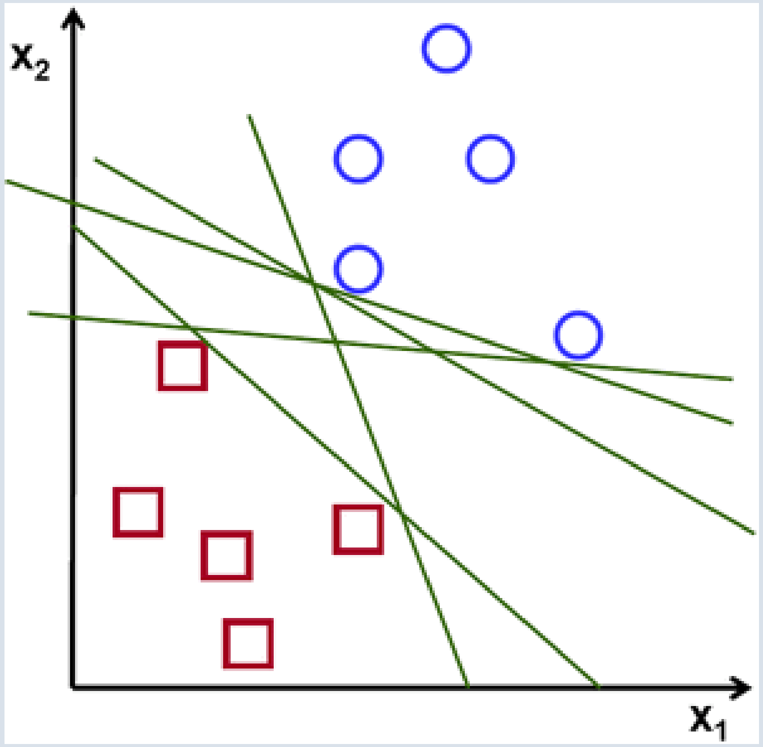
Support Vector Machine (SVM)

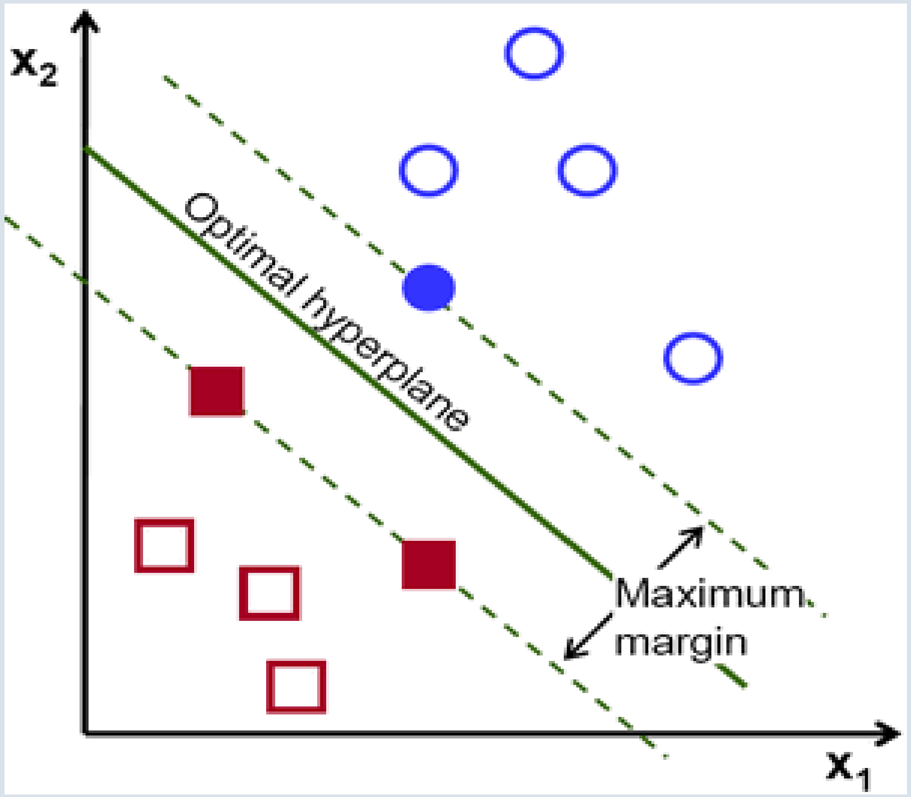
"**Support Vector Machine**" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems.
Support vector machine is highly preferred by many as it produces significant accuracy with less computation power.

Support Vector Machine (SVM)

SVM offers very high accuracy compared to other classifiers such as logistic regression, and decision trees. It is known for its kernel trick to handle nonlinear input spaces. It is used in a variety of applications such as face detection, intrusion detection, classification of emails, news articles and web pages, classification of genes, and handwriting recognition.
The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.

Support Vector Machine (SVM)





Support Vector Machine (SVM)

To separate the two classes of data points, there are many possible hyper planes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e. the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence. Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane.

Implementing SVM

Until now, you have learned about the theoretical background of SVM. Now you will learn about its implementation in Python using scikit-learn.

Let's use our previous dataset of Diabetes to predict whether a person has diabetes or not, based on information about the patient such as blood pressure, body mass index (BMI), age, etc.

Implementing SVM

We will perform the following steps.

1) Importing Required Libraries

2) Loading and reading data from csv file

3) Feature Selection

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

4) Splitting Data into training set and test set

5) Building SVM Model

6) Evaluating Model

Implementing SVM

We have already performed till the step-4 in our previous example of decision tree.
Now let’s perform the remaining steps i.e.

5) Building SVM Model
6) Evaluating Model

Implementing SVM

5) Building SVM Model

Let's build support vector machine model. First, import the SVM module and create support vector classifier object by passing argument kernel as the linear kernel in SVC() function.

Then, fit your model on train set using fit() and perform prediction on the test set using predict().

Implementing SVM

6) Evaluating Model

Let's estimate, how accurately the classifier or model can predict the type of cultivars.

Accuracy can be computed by comparing actual test set values and predicted values.

# Model Accuracy, how often is the classifier correct?

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Well, you got a classification rate of 80.20%, considered as good accuracy and also it’s accuracy is better than decision tree model.

Thank you

Uplatz