Bonjour with Support Vector Regression

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.

In this document we will be using **scikit-learn in Python**.

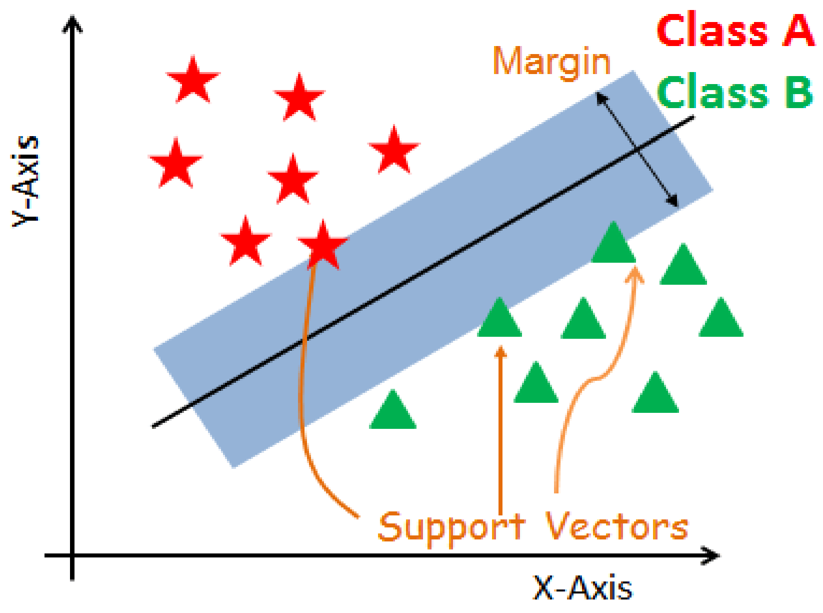
SVM is an exciting algorithm, and the concepts are relatively simple. The classifier separates data points using a hyperplane with the largest amount of margin. That's why an SVM classifier is also known as a discriminative classifier. SVM finds an optimal hyperplane which helps in classifying new data points.

In this, we are going to cover following topics:

Support Vector Machines

Generally, Support Vector Machines is considered to be a classification approach, it but can be employed in both types of classification and regression problems. It can easily handle multiple continuous and categorical variables. SVM constructs a hyperplane in multidimensional space to separate different classes. SVM generates optimal hyperplane in an iterative manner, which is used to minimize an error. The core idea of

SVM is to find a maximum marginal hyperplane (MMH) that best divides the dataset into classes.



Support Vectors

Support vectors are the data points, which are closest to the hyperplane. These points will define the separating line better by calculating margins. These points are more relevant to the construction of the classifier.

Hyperplane

A hyperplane is a decision plane which separates between a set of objects having different class memberships.

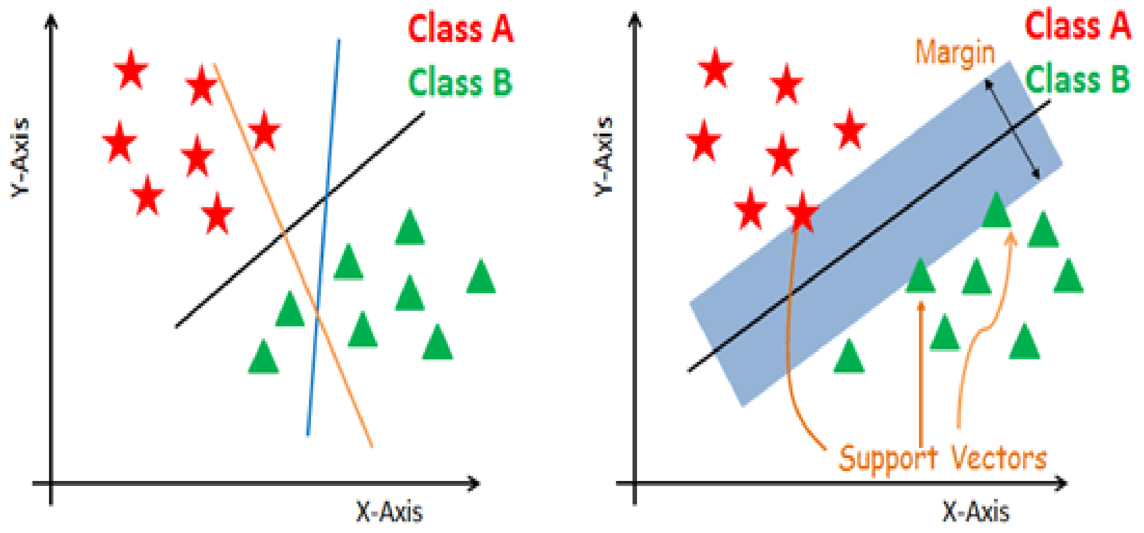
Margin

A margin is a gap between the two lines on the closest class points. This is calculated as the perpendicular distance from the line to support vectors or closest points. If the margin is larger in between the classes, then it is considered a good margin, a smaller margin is a bad margin.

How does SVM work?

The main objective is to segregate the given dataset in the best possible way. The distance between the both nearest points is known as the margin. The objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset. SVM searches for the maximum marginal hyperplane in the following steps:

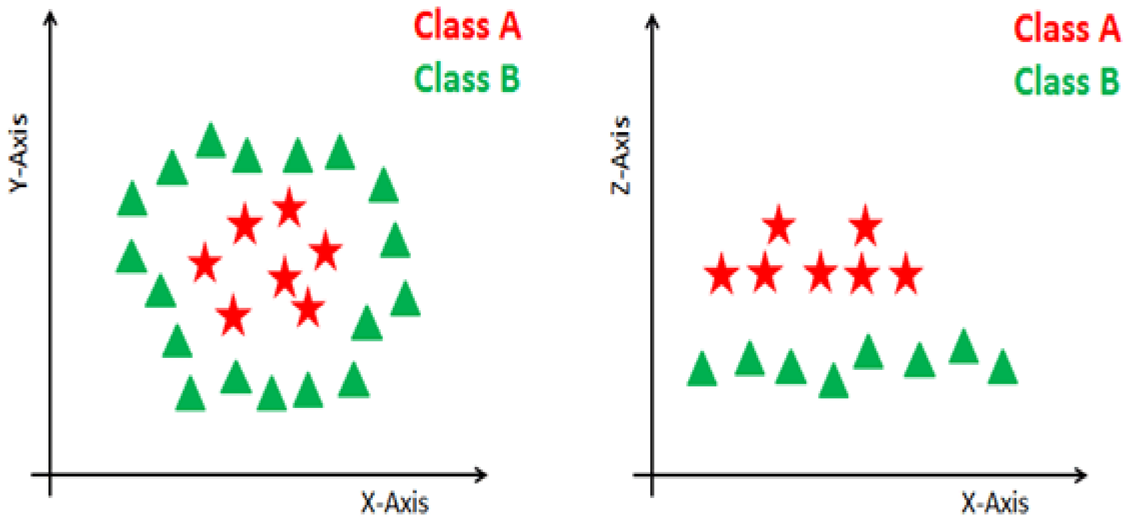
1. Generate hyperplanes which segregates the classes in the best way. Left-hand side figure showing three hyperplanes black, blue and orange. Here, the blue and orange have higher classification error, but the black is separating the two classes correctly.
2. Select the right hyperplane with the maximum segregation from both nearest data points as shown in the right-hand side figure.



Dealing with non-linear and inseparable planes

Some problems can’t be solved using linear hyperplane, as shown in the figure below (left-hand side).

In such situation, SVM uses a kernel trick to transform the input space to a higher dimensional space as shown on the right. The data points are plotted on the x-axis and z-axis (Z is the squared sum of both x and y: z=x^2=y^2). Now we can easily segregate these points using linear separation.



SVM Kernels

The SVM algorithm is implemented in practice using a kernel. A kernel transforms an input data space into the required form. SVM uses a technique called the kernel trick. Here, the kernel takes a low-dimensional input space and transforms it into a higher dimensional space. In other words, we can say that it converts non-separable problem to separable problems by adding more dimension to it. It is most useful in non-linear separation problem. Kernel trick helps us to build a more accurate classifier.

- **Linear Kernel** A linear kernel can be used as normal dot product any two given observations. The product between two vectors is the sum of the multiplication of each pair of input values.

$$\mathbf{K(x, xi) = sum(x * xi)}$$

- **Polynomial Kernel** A polynomial kernel is a more generalized form of the linear kernel. The polynomial kernel can distinguish curved or nonlinear input space.

$$\mathbf{K(x, xi) = 1 + sum(x * xi)^\wedge d}$$

Where d is the degree of the polynomial. d=1 is similar to the linear transformation. The degree needs to be manually specified in the learning algorithm.

- **Radial Basis Function Kernel** The Radial basis function kernel is a popular kernel function commonly used in support vector machine classification. RBF can map an input space in infinite dimensional space.

K(x, xi) = exp(-gamma * sum((x – xi^2))

Here gamma is a parameter, which ranges from 0 to 1. A higher value of gamma will perfectly fit the training dataset, which causes over-fitting. Gamma=0.1 is considered to be a good default value. The value of gamma needs to be manually specified in the learning algorithm.

Classifier Building in Scikit-learn

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

In the model the building part, we can use the cancer dataset, which is a very famous multi-class classification problem. This dataset is computed from a digitized image of a fine needle aspirate (FNA) of a body mass. They describe characteristics of the cell nuclei present in the image.

The dataset comprises **30** features (mean radius, mean texture, mean perimeter, mean area, mean smoothness, mean compactness, mean concavity, mean concave points, mean symmetry, mean fractal dimension, radius error, texture error, perimeter error, area error, smoothness error, compactness error, concavity error, concave points error, symmetry error, fractal dimension error, worst radius, worst texture, worst perimeter, worst area, worst smoothness, worst compactness, worst concavity, worst concave points, worst symmetry, and worst fractal dimension) and a target (type of cancer).

This data has two types of cancer classes: malignant (harmful) and benign (not harmful). Here, we can build a model to classify the type of cancer. The dataset is available in the scikit-learn library, or we can also download it from the UCI Machine Learning Library.

Loading Data

Let's first load the required dataset we will use.

```
#Import scikit-learn dataset library
from sklearn import datasets
#Load dataset
cancer = datasets.load_breast_cancer()
```

Exploring Data

After we have loaded the dataset, we might want to know a little bit more about it. We can check feature and target names.

```
# print the names of the 13 features
print("Features: ", cancer.feature_names)
# print the label type of cancer('malignant' 'benign')
print("Labels: ", cancer.target_names)
```

Let's explore it for a bit more. We can also check the shape of the dataset using shape.

```
# print data(feature)shape
cancer.data.shape
```

Let's check top 5 records of the feature set.

```
# print the cancer data features (top 5 records)
```

print(cancer.data[0:5])

Let's take a look at the target set.

# print the cancer labels (0: malignant, 1: benign)

print(cancer.target)

Splitting Data

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

Split the dataset by using the function train_test_split(). we need to pass 3 parameters features, target, and test_set size. Additionally, we can use random_state to select records randomly.

```
# Import train_test_split function
from sklearn.model_selection import train_test_split
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, test_size=0.3, random_state=1)
# 70% training and 30% test
```

Generating 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 our model on train set using **fit()** and perform prediction on the test set using **predict()**.

```
#Import svm model
from sklearn import svm
#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel
#Train the model using the training sets
clf.fit(X_train, y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

Evaluating the Model

Let's estimate how accurately the classifier or model can predict the breast cancer of patients.

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

```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test,
y_pred))
```

Well, we got a classification rate of 96.49%, considered as very good accuracy.

For further evaluation, we can also check precision and recall of model.

```
# Model Precision: what percentage of positive tuples are labelled
as such?
```

print("Precision:", metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?

print("Recall:", metrics.recall_score(y_test, y_pred))

Well, we got a precision of 98% and recall of 96%, which are considered as very good values.

Tuning Hyperparameters

- **Kernel**: The main function of the kernel is to transform the given dataset input data into the required form. There are various types of functions such as linear, polynomial, and radial basis function (RBF). Polynomial and RBF are useful for non-linear hyperplane. Polynomial and RBF kernels compute the separation line in the higher dimension. In some of the applications, it is suggested to use a more complex kernel to separate the classes that are curved or nonlinear. This transformation can lead to more accurate classifiers.
- **Regularization**: Regularization parameter in python's Scikit-learn C parameter used to maintain regularization. Here C is the penalty parameter, which represents misclassification or error term. The misclassification or error term tells the SVM optimization how much error is bearable. This is how we can control the trade-off between decision boundary and misclassification term. A smaller value of C creates a small-margin hyperplane, and a larger value of C creates a larger-margin hyperplane.
- **Gamma**: A lower value of Gamma will loosely fit the training dataset, whereas a higher value of gamma will exactly fit the training dataset, which causes over-fitting. In other words, we can say a low value of gamma considers only nearby points in calculating the separation line, while a value of gamma considers all the data points in the calculation of the separation line.

Advantages

SVM Classifiers offer good accuracy and perform faster prediction compared to Naïve Bayes algorithm. They also use less memory because they use a subset of training points in the decision phase. SVM works well with a clear margin of separation and with high dimensional space.

Disadvantages

SVM is not suitable for large datasets because of its high training time and it also takes more time in training compared to Naïve Bayes. It works poorly with overlapping classes and is also sensitive to the type of kernel used.

Way-2: With external Datasets

Step-1: Import necessary Lib

```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
```

Step-2: Load Dataset

```
df = pd.read_csv(“fruits.csv”)
df
```

Step-3: represent this data frame as a scatter plot.

```
plt.xlabel('Weight')
plt.ylabel('Label')
plt.scatter(df['Weight'],
df['Weight'],color="green",marker='+', linewidth='5')
plt.scatter(df['Label'],
df['Label'],color="blue",marker='.' , linewidth='5')
```

Step-4: Split the dataset into training and test samples with a ratio of 80% & 20%.

```
from sklearn.model_selection import
train_test_split
train_set, test_set = train_test_split(df,
test_size=0.2)
```

Step-5: classify the predictors and target.

```
x_train = train_set.iloc[:,0:2].values
y_train = train_set.iloc[:,2].values
x_test = test_set.iloc[:,0:2].values
y_test = test_set.iloc[:,2].values
```

Step-6: initialize the Support Vector Machine (SVM) and fitting the training data.

```
from sklearn.svm import SVC
model = SVC(kernel='rbf', random_state = 1)
model.fit(x_train, y_train)
```

Step-7: check the accuracy of our model.

```
model.score(x_test, y_test)
```

Step-8: predict the class of a fruit whose weight is 55 and size is 4.

```
model.predict([[55, 4]])
```