## Experiment **No. 9**

Support Vector Machine

**OBJECTIVE:**

* To be able to understand the basics of SVM.
* To be able to perform supervised learning and SVM classifiers on real dataset.

**Support Vector Machine (SVM):**

Support Vector Machines (SVMs in short) are supervised machine learning algorithms that are used for classification and regression purposes. In this Lab you should build a Support Vector Machines classifier to classify a Pulsar star. And use **Predicting a Pulsar Star** dataset for this LAB.

**Support Vector Machines** (SVMs in short) are machine learning algorithms that are used for classification and regression purposes. SVMs are one of the powerful machine learning algorithms for classification, regression and outlier detection purposes. An SVM classifier builds a model that assigns new data points to one of the given categories. Thus, it can be viewed as a non-probabilistic binary linear classifier.

The original SVM algorithm was developed by Vladimir N Vapnik and Alexey Ya. Chervonenkis in 1963. At that time, the algorithm was in early stages. The only possibility is to draw hyperplanes for linear classifier. In 1992, Bernhard E. Boser, Isabelle M Guyon and Vladimir N Vapnik suggested a way to create non-linear classifiers by applying the kernel trick to maximum-margin hyperplanes. The current standard was proposed by Corinna Cortes and Vapnik in 1993 and published in 1995.

SVMs can be used for linear classification purposes. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using the **kernel trick**. It enables us to implicitly map the inputs into high dimensional feature spaces.

**SVM Intuition**

**Hyperplane**[**¶**](https://www.kaggle.com/code/prashant111/svm-classifier-tutorial#Hyperplane)

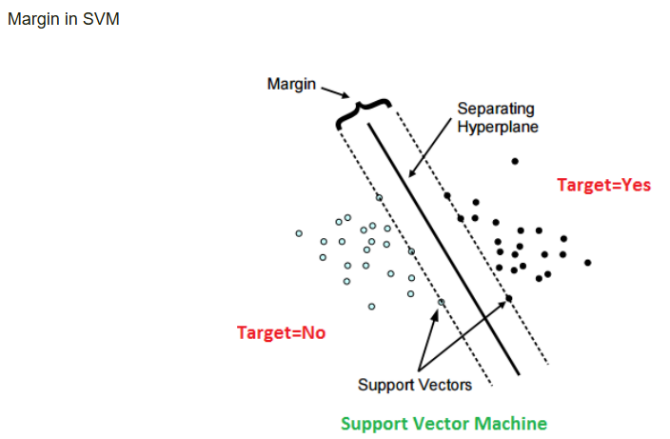
A hyperplane is a decision boundary which separates between given set of data points having different class labels. The SVM classifier separates data points using a hyperplane with the maximum amount of margin. This hyperplane is known as the maximum margin hyperplane and the linear classifier it defines is known as the maximum margin classifier.

**Support Vectors**

Support vectors are the sample data points, which are closest to the hyperplane. These data points will define the separating line or hyperplane better by calculating margins.

**Margin**

A margin is a separation gap between the two lines on the closest data points. It is calculated as the perpendicular distance from the line to support vectors or closest data points. In SVMs, we try to maximize this separation gap so that we get maximum margin.

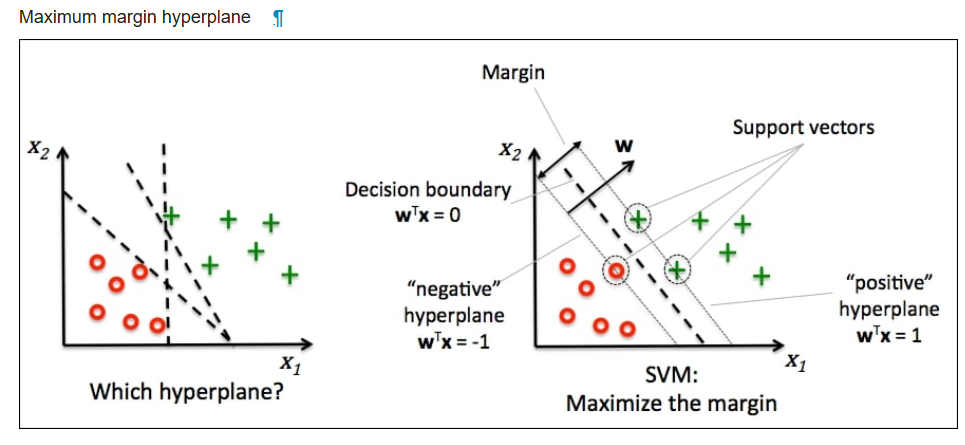
The following diagram illustrates these concepts visually.

**SVM Under the hood**

In SVMs, our main objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset. SVM searches for the maximum margin hyperplane in the following 2 step process –

1. Generate hyperplanes which segregates the classes in the best possible way. There are many hyperplanes that might classify the data. We should look for the best hyperplane that represents the largest separation, or margin, between the two classes.
2. So, we choose the hyperplane so that distance from it to the support vectors on each side is maximized. If such a hyperplane exists, it is known as the **maximum margin hyperplane** and the linear classifier it defines is known as a **maximum margin classifier**.

The following diagram illustrates the concept of **maximum margin** and **maximum margin hyperplane** in a clear manner.



**Problem with dispersed datasets**

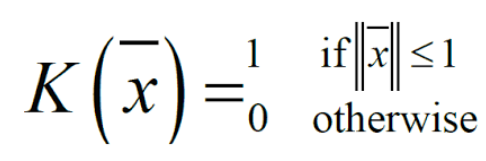
Sometimes, the sample data points are so dispersed that it is not possible to separate them using a linear hyperplane. In such a situation, SVMs uses a kernel trick to transform the input space to a higher dimensional space as shown in the diagram below. It uses a mapping function to transform the 2-D input space into the 3-D input space. Now, we can easily segregate the data points using linear separation.

Kernel trick - transformation of input space to higher dimensional space

**Kernel trick**

In practice, SVM algorithm is implemented using a kernel. It uses a technique called the kernel trick. In simple words, a kernel is just a function that maps the data to a higher dimension where data is separable. A kernel transforms a low-dimensional input data space into a higher dimensional space. So, it converts non-linear separable problems to linear separable problems by adding more dimensions to it. Thus, the kernel trick helps us to build a more accurate classifier. Hence, it is useful in non-linear separation problems.

We can define a kernel function as follows



In the context of SVMs, there are 4 popular kernels – **Linear kernel, Polynomial kernel, Radial Basis Function (RBF) kernel (also called Gaussian kernel) and Sigmoid kernel**. These are described below -

**SVM Scikit-Learn libraries**

Scikit-Learn provides useful libraries to implement Support Vector Machine algorithm on a dataset. There are many libraries that can help us to implement SVM smoothly. We just need to call the library with parameters that suit to our needs. In this project, I am dealing with a classification task. So, I will mention the Scikit-Learn libraries for SVM classification purposes.

First, there is a **LinearSVC()** classifier. As the name suggests, this classifier uses only linear kernel. In LinearSVC() classifier, we don’t pass the value of kernel since it is used only for linear classification purposes.

Scikit-Learn provides two other classifiers - **SVC()** and **NuSVC()** which are used for classification purposes. These classifiers are mostly similar with some difference in parameters. **NuSVC()** is similar to **SVC()** but uses a parameter to control the number of support vectors. We pass the values of kernel, gamma and C along with other parameters. By default kernel parameter uses rbf as its value but we can pass values like poly, linear, sigmoid or callable function.

Perform the Following operations on provided dataset to complete this task

**Exploratory Data Analysis (EDA)**

Perform following EDA operations

*# view dimensions of dataset*

*# preview the dataset*

*# view the column names of the dataframe*

*# remove leading spaces from column names(if exists)*

*# view column names again*

*# rename column names (if needed)*

*# view the renamed column names*

*# check distribution of target\_class column*

*# view the percentage distribution of target\_class column*

*# view summary of dataset i.e. (info)*

*# check for missing values in variables*

*# view summary statistics in numerical variables*

*# draw boxplots to visualize outliers*

*# split X and y into training and testing sets (80/20)*

*# check the shape of X\_train and X\_test*

*# Do Feature Scaling if required*

*# Run SVM with default hyperparameter means C=1.0, kernel=rbf and gamma=auto among other parameters.*

*# Run SVM with rbf kernel and C=100.0*

*# instantiate classifier with rbf kernel and C=1000*

*# instantiate classifier with linear kernel and C=1.0*

*# instantiate classifier with linear kernel and C=100.0*

*# instantiate classifier with linear kernel and C=1000.0*

*# Compare the train-set and test-set accuracy*

*# print the scores on training and test set*

*# instantiate classifier with polynomial kernel and C=1.0*

*# instantiate classifier with polynomial kernel and C=100.0*

*# instantiate classifier with sigmoid kernel and C=1.0*

*# instantiate classifier with sigmoid kernel and C=100.0*

***Comments on Experiments***

*We get maximum accuracy with rbf and linear kernel with C=100.0. and the accuracy is 0.9832. Based on the above analysis we can conclude that our classification model accuracy is very good. Our model is doing a very good job in terms of predicting the class labels.*

*But, this is not true. Here, we have an imbalanced dataset. The problem is that accuracy is an inadequate measure for quantifying predictive performance in the imbalanced dataset problem.*

*So, we must explore alternative metrices that provide better guidance in selecting models. In particular, we would like to know the underlying distribution of values and the type of errors our classifer is making.*

*One such metric to analyze the model performance in imbalanced classes problem is Confusion matrix.*