**Support Vector Machine: Explained with a binary classification problem.**

[[Rakesh M K](https://medium.com/@mkk.rakesh?source=post_page-----bb1d5be336c4--------------------------------)](https://medium.com/@mkk.rakesh?source=post_page-----bb1d5be336c4--------------------------------)

[Rakesh M K](https://medium.com/@mkk.rakesh?source=post_page-----bb1d5be336c4--------------------------------)

·

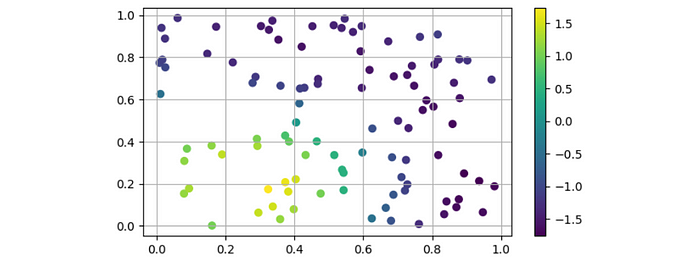
[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F5796472804b9&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40mkk.rakesh%2Fsupport-vector-machine-explained-with-a-binary-classification-problem-bb1d5be336c4&user=Rakesh+M+K&userId=5796472804b9&source=post_page-5796472804b9----bb1d5be336c4---------------------post_header-----------)

4 min read

·

Oct 18

100



SVM linear kernel Decision Values for a linearly separable data.

SVM (Support Vector Machines) distinguishes itself from other Machine Learning models with its proficiency in handling **high-dimensional data**, finding **complex decision boundaries**. SVM offers various kernels for both linear and non-linear data, and maintaining robust performance even when data is limited. All these properties make it a powerful tool for a wide range of applications.

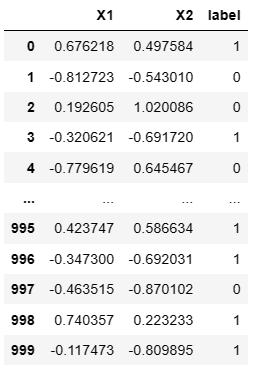
**Kernel Trick: Transformation of data to high dimensional space**

SVM operates by **transforming the data to a space where it is more easily separable**which might or might not be a higher-dimensional linear space.The transformation process is often called as**kernel trick.** It depends on the choice of kernel and the nature of the data. Linear kernels transform data linearly, while non-linear kernels can create more complex decision boundaries in the transformed space.

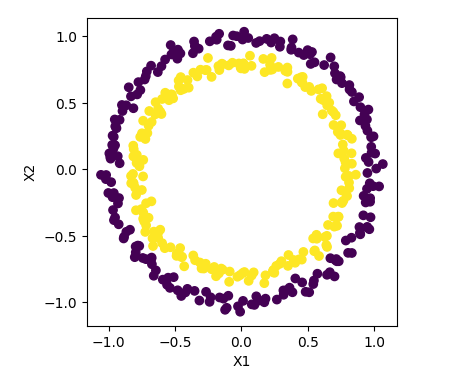
*For in depth understanding of kernel tricks, refer:*<https://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf>

For the experiment, data is generated using scikit-learns *make\_circle*function and visualized as below.

from sklearn.datasets import make\_circles  
import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
  
#1000 samples  
n\_samples=400  
#create circles  
X,y=make\_circles(  
 n\_samples,  
 noise=0.03,  
 random\_state=101)  
circle=pd.DataFrame({'X1':X[:,0],'X2':X[:,1],'label':y})  
circle



Generated data



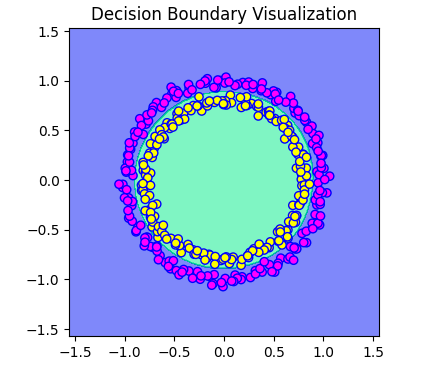
Generated data in 2D

Looking at the data in the above plot, it is not linearly separable. At this point we do have to make use of different kernel tricks offered by SVM. Among different kernels (**‘linear’, ‘poly’**(polynomial)**, ’sigmoid’, ‘rbf’ (**radial basis) ...), we logically turn to use ‘rbf’ as kernel since our data looks radial. Let’ see how it performs.

from sklearn import svm  
# create an instance of SVC  
sVC = svm.SVC( C=2.0, # regularization   
 kernel='rbf', # radial or gaussian kernel  
 random\_state=101)  
  
sVC.fit(X, y) # fit data to SVC  
yPred = sVC.predict(X) # predict using same data  
  
acc = np.mean(yPred == y) # Check accuracy  
print(f'accuracy:{acc}') # print accuracy  
  
accuracy: 1

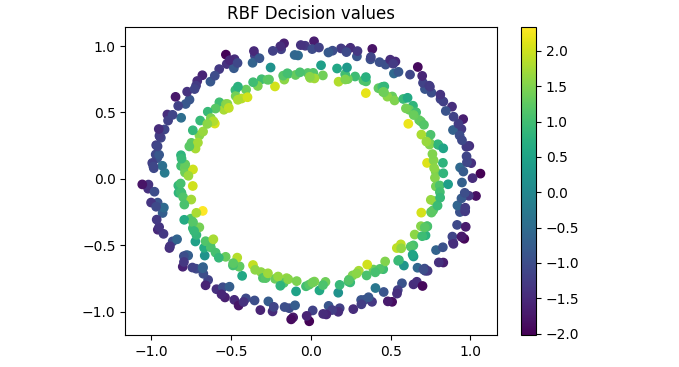
Since we got a high accuracy, we are more interested to see the decision boundary generated by SVM using the rbf kernel. Let’s plot using a mesh grid and visualize it.

# function to visualize decision boundary  
def decision\_boundary\_plot(X,y,model):  
 x\_min, x\_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5  
 y\_min, y\_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5  
  
 # Generate a grid of points  
 xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.01), np.arange(y\_min, y\_max, 0.01))  
  
 # Predict class labels for each point in the grid  
 x\_in = np.c\_[xx.ravel(), yy.ravel()]  
 y\_pred=model.predict(x\_in)  
 y\_pred=y\_pred.reshape(xx.shape)  
  
 # Plot decision boundary and data points  
 plt.contourf(xx, yy, y\_pred, alpha=.5,cmap='winter')  
 plt.scatter(X[:, 0], X[:, 1], c=y,cmap='spring',edgecolor='b')  
 plt.title('Decision Boundary Visualization')  
 plt.show()  
  
decision\_boundary\_plot(X,y,sVC)



The model could construct a fine radial decision boundary to classify data points. But how does it actually work? SVM produces **decision values** for each data points based on the chosen kernel. For a binary classification problem, data points with decision values above one will be classified as positive class and others to negative class. Let’s check the decision values generated for the data with rbf.

decision\_values = sVC.decision\_function(X) # generate decision values  
  
# Visualize the decision function values  
plt.scatter(X[:, 0], X[:, 1], c= decision\_values, cmap='viridis')  
plt.colorbar()  
plt.title('RBF Kernel Decision values')  
plt.show()

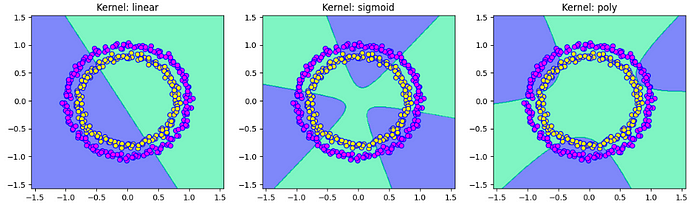


Decision values assigned to each data point can be seen from the above plot. Values above 0 will be classified as positive and below zero as negative *(see values in color map).*

**How other kernels work on same data?**

To see how other kernels make decision boundary on same data and for better understanding on kernel functions, SVM is modelled with different kernels and decision boundaries are plotted below.

kernels = ['linear', 'sigmoid' ,'poly'] # list of kernels  
for i, kernel in enumerate(kernels):  
 svc = svm.SVC( C=2.0,kernel= kernel)  
 svc.fit(X,y) # fit to SVC  
 decision\_boundary\_plot(X,y,svc) # plot decision boundary



Decision boundary for different kernels

From the plot we can see the decision boundary corresponds to each kernel functions. It is easy to understand from the first plot that a linear kernel tries to separate the data with a linear boundary and same logic with other kernels. It gives us a clear idea that, with respect to the nature or spread of data we can choose the kernel function accordingly…

There are many hyperparameters for SVM such as *C*(regularization parameter), *class\_weight*and many more which can be used according to the nature of data and complexity of pattern it follows. Also, visualizing the high dimensional transformation is often challenging when data dimension itself is large even though such effort is not really required once the innerworkings of SVM is well understood.

Hope this page helped to understand SVM at some basic level.*bests...*