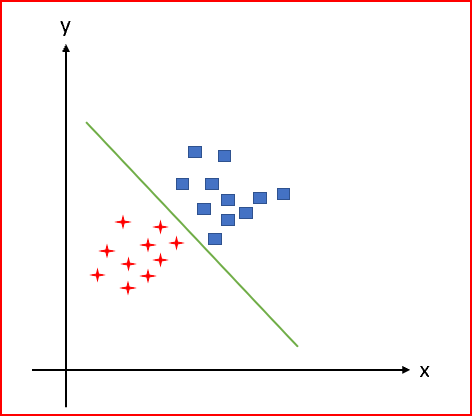
Support Vector Machines

***In this article we will understand intuition behind Support Vector Machines(SVM). Relevance of the SVM hyperparameters -margin, gamma, regularization and kernel. Pros and cons of SVM and finally an example in Python.***

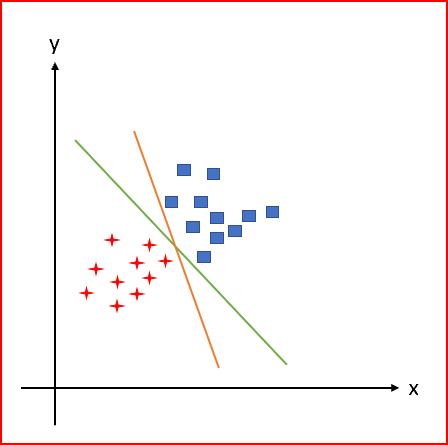
Our objective is to classify a dataset. To do that we plot the data set in n-dimensional space to come up with a linearly separable line. This line helps separate two classes of data.

***sounds simple right? This is the basis for SVM.***



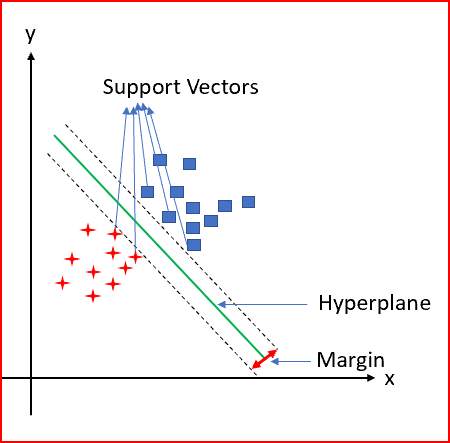
Support vector machines helps to find a hyperplane (line) to linearly separate data points into two classes.

***But we can have multiple lines as shown below that can separate the data points. How do we choose the best line that separates the dataset into two classes?***



SVM finds the hyperplane to maximizes the margin between support vectors of the two classes. Hyperplane are decision boundaries classifying the the dataset while maximizing the margin.

***What are support vectors?***



Support vectors are the data points in the dataset that are nearest to the hyperplane. Removing support vectors will alter the hyperplane separating two classes. Support vectors are critical elements of the dataset as SVM is built on them.

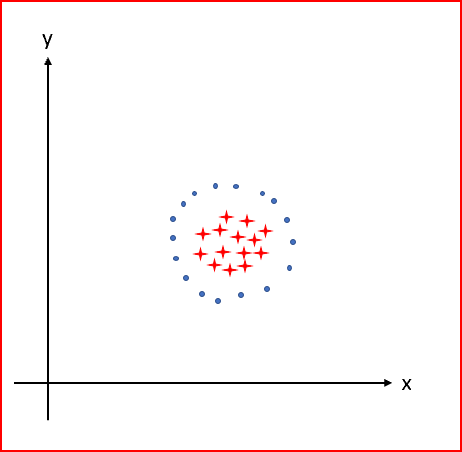
Support vector machine has two main objectives

* Find a hyperplane(line) that linearly separates the data points into two classes
* Maximize the margin between support vectors of the two classes

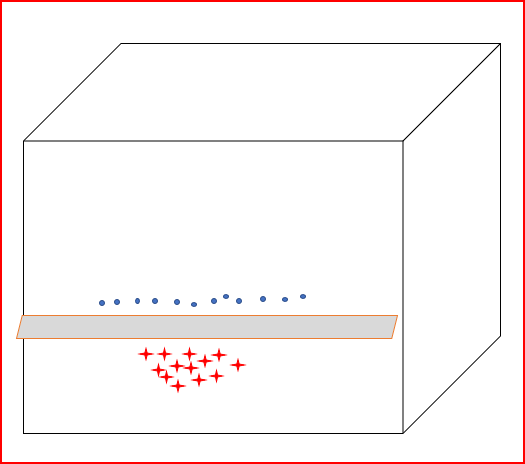
***How does SVM separate two classes when they are not linearly separable?***

Data below is not linearly separable. We cannot draw a straight line to separate the two classes.

To solve the issue, we take non linearly separable data in n- dimensional space. Transform it to a higher dimensional space to make it linearly separable.

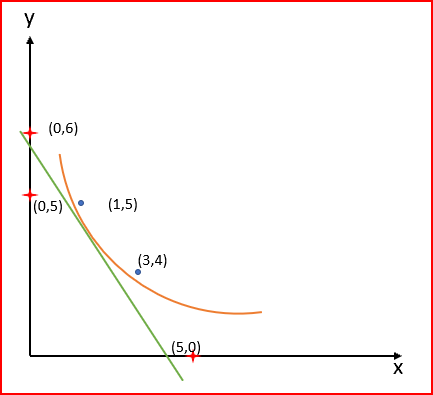


Data above is not linearly separable in 2 dimensional space. We transform the data into a 3 dimensional space and draw a hyperplane. The hyperplane now separates the two classes linearly. This is **kernel trick**.



***Performing kernel trick step by step***

let’s say we have 5 data points in a 2 dimensional space that are not linearly separable



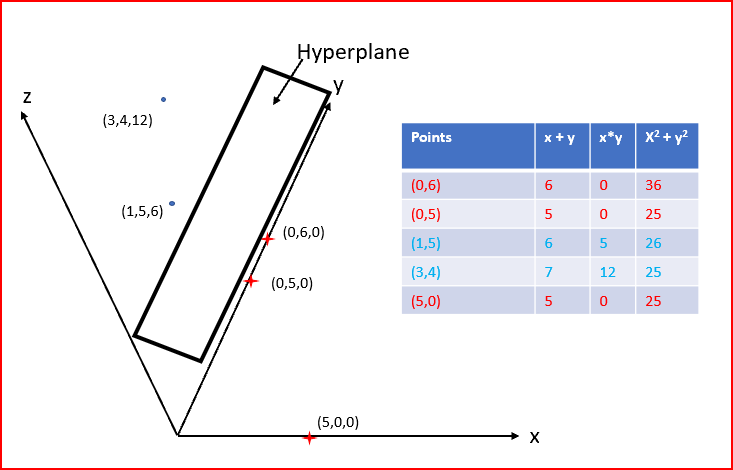
Data is not linearly separable

We will transform the data to a 3 dimensional space and see if the data is linearly separable.

Let’s try with three equations, x+ y, x\*y and x² + y² .

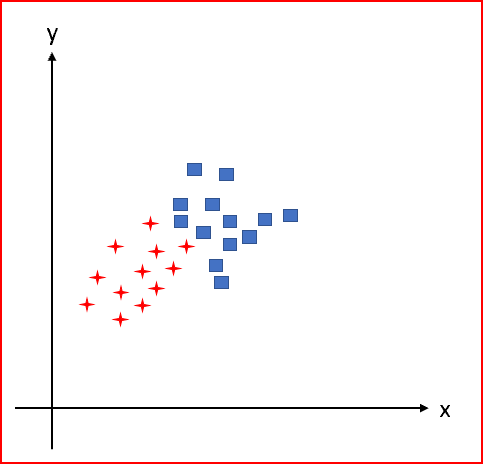
x\*y have product as either 0 or a positive number. Hence, if the product is 0 then we can classify the point in class 1. If the product is a positive number then we can classify the point in class 2

We take the x\*y as z coordinate and plot the data in a 3 dimensional space. We now see linearly separable data points .



Data transformed to a higher dimension becomes linearly separable

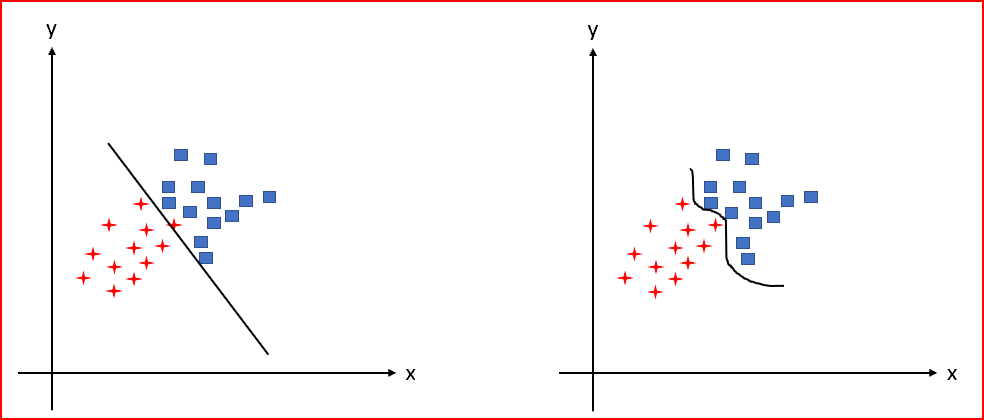
***What if the data of the two classes overlap?***



Overlapping data

We have a trade off between choosing a smooth decision boundary or classifying the training point correctly

* Plot on right side : Classifies all the training point correctly
* Plot on left side : Misclassifies a few training points



Left side plot where C is small and we have more misclassification. Right side plot where C is large and we have training data classified correctly

One way we handle this by **Regularization(C).**

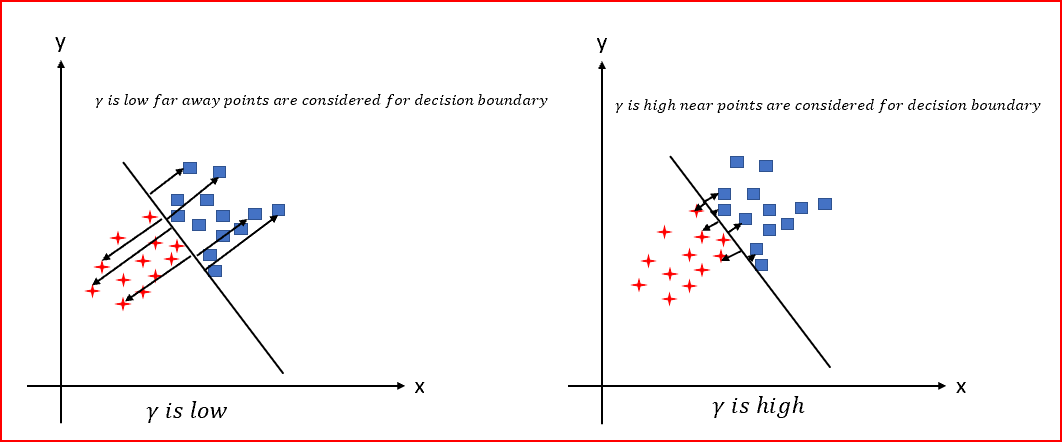
**Regularization parameters specifies the degree of allowed misclassification**. Regularization is inversely proportional to the margin.

* When Regularization(C) parameters tends to be close to 0 then more miss-classifications are allowed. Smaller value of C will force the optimizer to look for a large margin for hyperplanes separating the two classes.
* High Regularization (C) parameter will choose a hyperplane that does a good job of getting all data points classified correctly. A large value of C will force the optimizer to choose a smaller margin for the hyperplane. This causes more training data points to be classified correctly.

Another approach to handle overlapping data is to specify a hyperparameter called Gamma(γ).

**Gamma(γ) defines how far the influence of a training example reaches**.

* Low gamma value tells us that we should consider far way point when deciding the decision boundary. This gives a smooth linear separation between two classes
* High gamma value tells us that we should consider near points when deciding the decision boundary. This gives a curvy separation between two classes and may sometimes overfit the test data.



***Summarizing SVM***

**Support Vector Machine is**

* **Deterministic algorithm**: Output for a particular input is predetermined. Output will remain the same irrespective of the times you run the algorithm.
* **SVM is non probabilistic**
* **SVM is binary linear classification algorith**m
* **SVM is used for both regression and classification**

**Pros and Cons of SVM**

**Pros of SVM**

* Effective in high dimensional space
* Memory efficient as SVM uses a subset of training point for decision function
* Performs well with linear as well as non linear boundaries. For non linear boundaries we need to select the right kernel
* Robust against outliers(controlled using C)

**Cons of SVM**

* Large datasets requires more processing time
* Finding optimal values for different hyperparameters is not easy
* selecting right kernel for non linear boundaries can be tricky
* Multi class classification is not directly possible.

Now we will implement Support Vector Machine classifier using Python

I have used Skin segmentation [Dataset](http://archive.ics.uci.edu/ml/datasets/Skin+Segmentation). I have taken a subset of the data that contains the two classes

First importing the required libraries. we will add more libraries as we build SVM classifier

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn import preprocessing

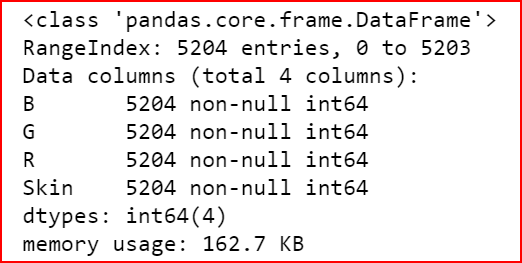
I have saved the data in default Jupyter folder as skin\_1.csv.

Reading the data into dataset\_1

dataset\_1 = pd.read\_csv(‘skin\_1.csv’)

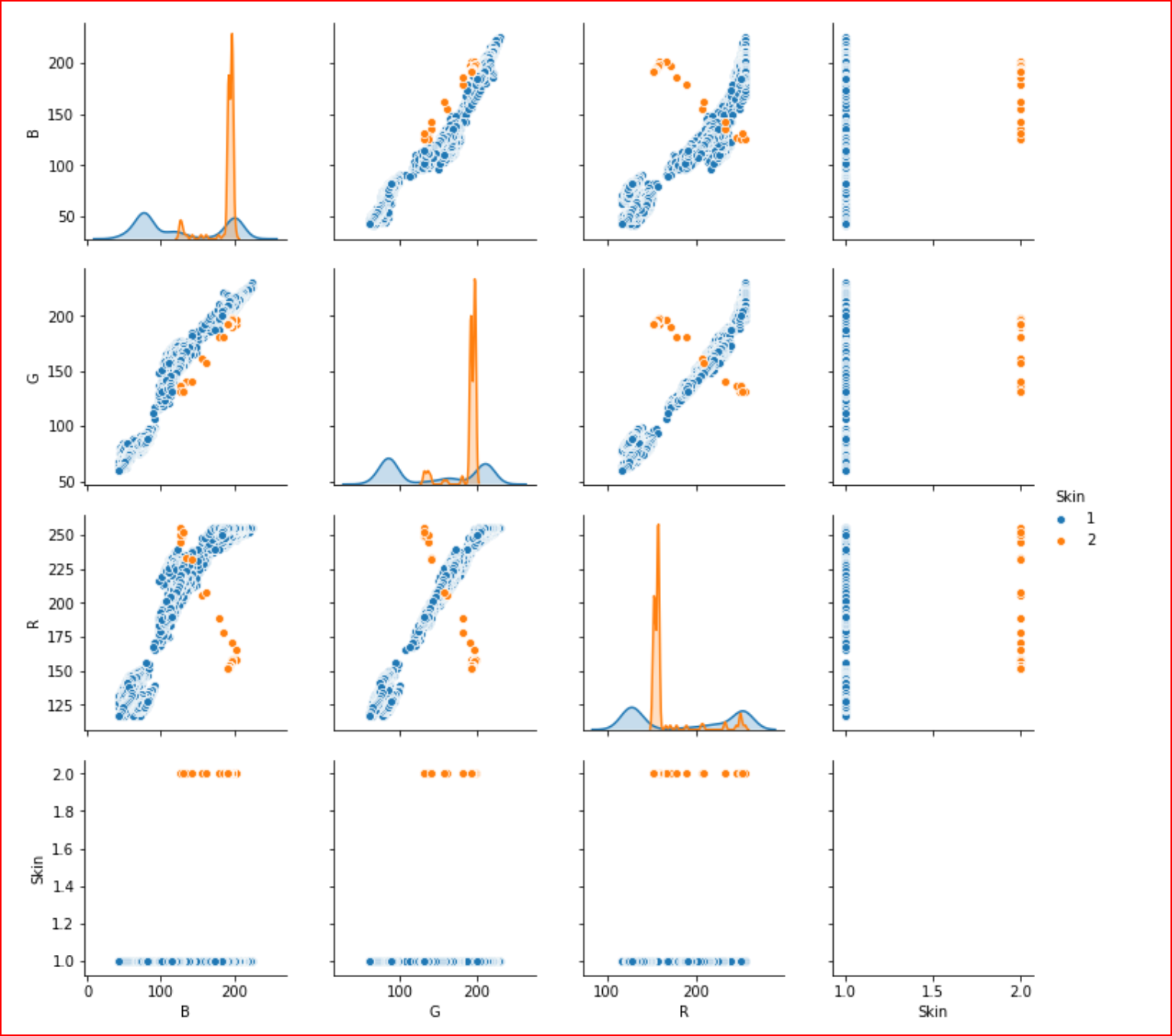
Exploring the data in the dataset and describing all input variables

dataset\_1.info()



Visualize the data using seaborn

sns.pairplot(dataset\_1, hue='Skin')



Creating input feature (X) and output feature(Y)

X= dataset\_1.iloc[:,0:3]  
Y= dataset\_1.iloc[:,-1]

Splitting the dataset\_1 to training and test sets. Test set will be 40% and training set will 60% of the dataset\_1

from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test,Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.4)

we use **StandardScaler** to normally distribute the input features, both train and test data. This way the data is distributed around 0, with a standard deviation of 1.

from sklearn.preprocessing import StandardScaler  
sc= StandardScaler()  
X\_train = sc.fit\_transform(X\_train)  
X\_test = sc.transform(X\_test)

Now creating an instance of svm and then applying the classifier SVC

from sklearn.svm import SVC  
classifier=SVC(kernel ='linear', C=1, gamma=1)  
classifier.fit(X\_train, Y\_train)

we now predict the data.

y\_pred= classifier.predict(X\_test)

calculating the accuracy

from sklearn.metrics import accuracy\_score  
print(accuracy\_score(Y\_test, y\_pred))

getting an accuracy of 0.997598463016330.

I have tried to put the intuition behind SVM. Hope I made it simple to understand

Source Link: <https://medium.com/datadriveninvestor/support-vector-machines-ae0ff2375479>

Further study:

<https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72>

<https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496>

<https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989>