Machine Learning (ML) with Python

Supervised Learning (SVM)

Dr. Aeshah Alsughayyir

Collage of Computer Science and Engineering

Taibah University

2021-2022

Outline:

- Preliminary Basics
- Support Vector Machine (SVM)
  - What is Support Vector Machine?
  - How does it work?
  - How to implement SVM in Python?
  - How to tune Parameters of SVM?
  - Pros and Cons associated with SVM

Preliminary Basics

- **What is a Norm?**

  In Linear Algebra, a Norm refers to the magnitudes or total length of the vectors in a space.

- **Vector Norm**

  The length of the vector V is referred to as the vector norm or the vector's magnitude and denoted by $||V||$ (it is always a positive number, except for a vector of all zero values).

> "The length of a vector is a nonnegative number that describes the extent of the vector in space, and is sometimes referred to as the vector's magnitude or the norm."
> - *Page 112, No Bullshit Guide To Linear Algebra, 2017*

Preliminary Basics (Cont.)

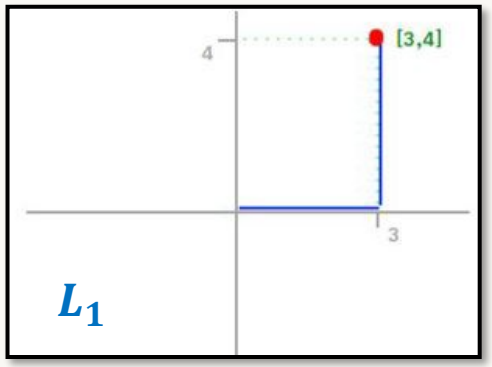
There are different ways to measure the magnitude of vectors, here are the most common:

L_0 Norm:

It is actually not a norm and corresponds to the total number of nonzero elements in a vector. For example, the L0 norm of the vectors (0,0) and (0,2) is 1 because there is only one nonzero element.

L_1 Norm:

It is the sum of the magnitudes of the vectors in a space. For example, having $V = (3,4)$, then $\left|\left|V\right|\right|_1 = |3| + |4| = 7$ as shown in the figure. In other words, the L1 norm is the distance you have to travel between the origin (0,0) to the destination (3,4)

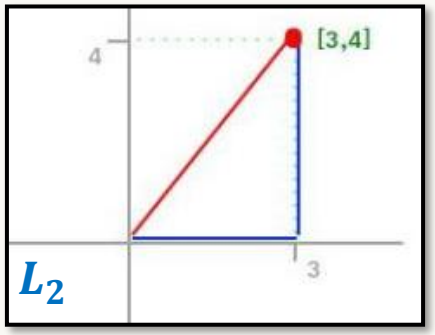


L_2 Norm:

the most popular norm, also known as the Euclidean norm. It is the shortest distance to go from one point to another.

Using the same example, the L2 norm is calculated by

$$\left|\left|V\right|\right|_2 = \sqrt[2]{|3|^2 + |4|^2} = \sqrt[2]{9 + 16} = \sqrt[2]{25} = 5$$



L_∞ Norm:

Gives the largest magnitude among each element of a vector.

Having $V = (-6,4,2)$, then $\left|\left|V\right|\right|_\infty = 6$

Preliminary Basics (Cont.)

- **In short:**

  $\boldsymbol{L_p\ Norm}$ generalizes the three norms. For $p > 0$, it is defined in R by: $\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$

  So, with various values of p norms can be calculated as follows:

- p=1 L^1 norm $\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i|$
- p=2 L^2 norm $\|\boldsymbol{x}\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}$
- p= ∞ L^∞ norm $\|\boldsymbol{x}\|_\infty = \max_i |x_i|$

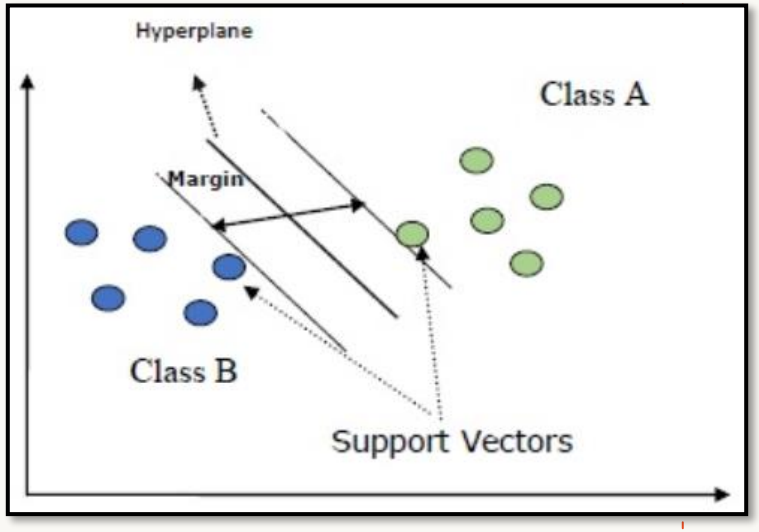
Support Vector Machine

- What is Support Vector Machine?
- How does SVM work?
- How to implement SVM in Python?
- How to tune Parameters of SVM?
- Pros and Cons associated with SVM

What is Support Vector Machine?

- "Support Vector Machine" (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems.
- In the SVM algorithm, we plot each data item as a point in n-dimensional space (where *n* is number of features you have) with the value of each feature being the value of a particular coordinate.
- Then, we perform classification by finding the *hyperplane* that differentiates the two classes very well.

- **Hyperplane**: This is the decision boundary that separates two classes in n-dimensional space. The number of features present in our dataset decides the number of hyperplanes.
  For example, if we have just two features → then the hyperplane will be a straight line,
  in the case of 3 features → we get a 2-D plane.
- **Support Vectors**: These data points affect the positioning of the hyperplane
- **Margin**: Distance between a vector/data point and the hyperplane is called margin.
- **Maximum margin**: Hyperplane with the maximum margin is called an optimal hyperplane.

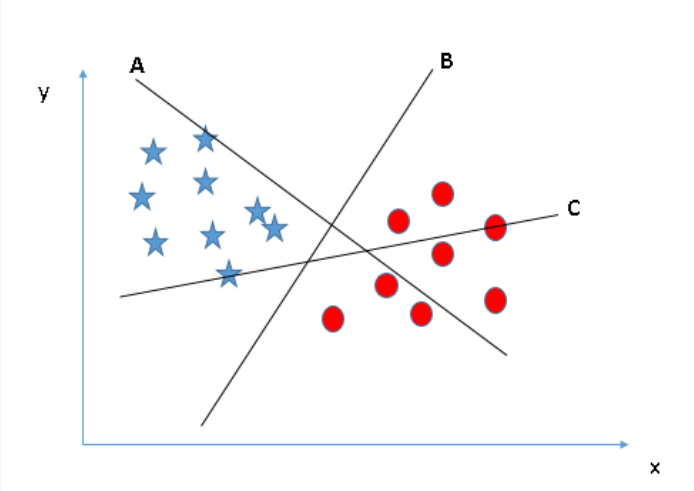


How does SVM work?

- **How can we decide the right (best) hyperplane?**

  Let's look at the following cases, for identifying the right hyperplane to classify stars and circles:

1)



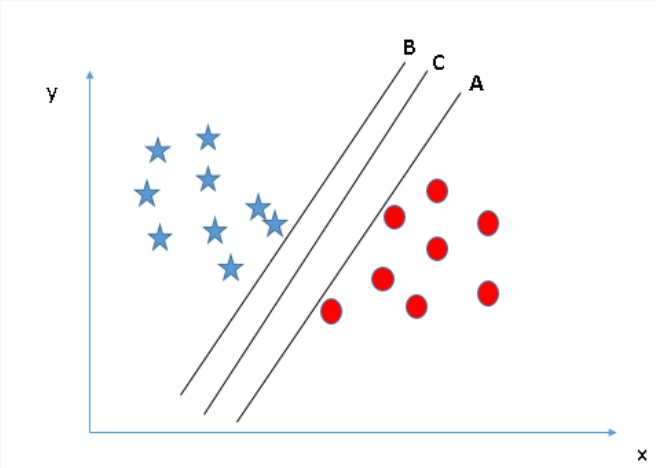
- Here, we have three hyperplanes (A, B, and C).
- In this scenario, hyperplane "B" has excellently performed this job.

How does SVM work? (Cont.)

- **How can we decide the right (best) hyperplane?**

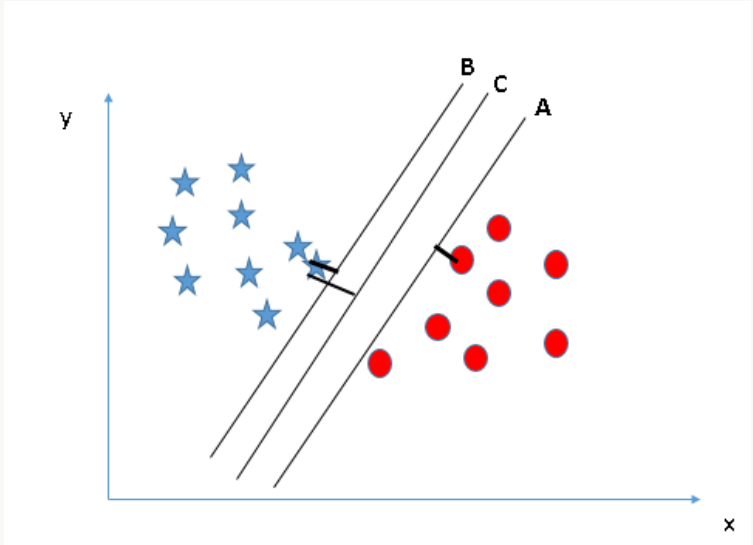
  Let's look at the following cases, for identifying the right hyperplane to classify stars and circles:

2)



- Here, we have another three hyperplanes (A, B, and C).
- All seems like doing the job (if we using linear logistic regression). However, with SVM there are one of them is better than the others.

Looking at the distances (**Margin**) between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane.



- The margin for hyper-plane C is high as compared to both A and B. So, the right hyper-plane is C.

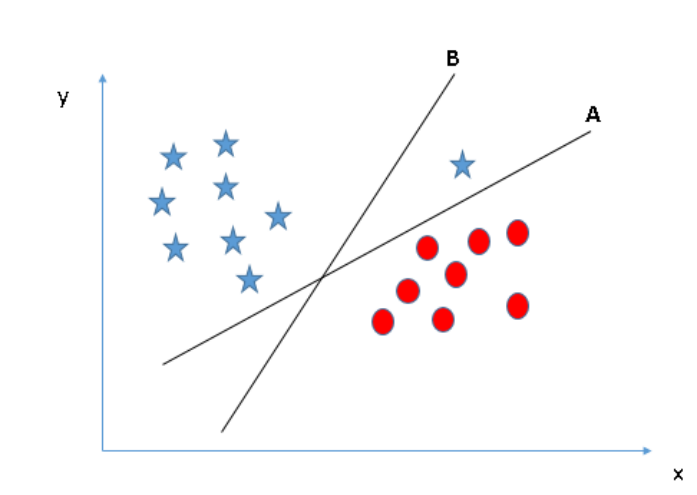
The lightning **reason for selecting the hyper-plane with higher margin** is robustness. If we select a hyper-plane having low margin, then there is high chance of miss-classification.

How does SVM work? (Cont.)

- **How can we decide the right (best) hyperplane?**

  Let's look at the following cases, for identifying the right hyperplane to classify stars and circles:

3)



- Now, you may think that the right one is **B**, as it has higher margin!!!

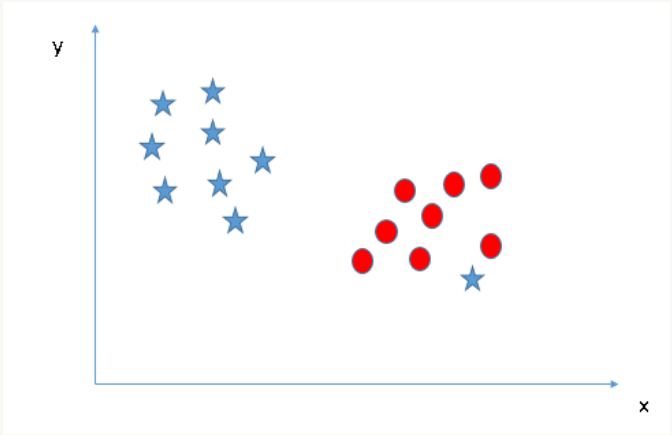
But…

SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin.

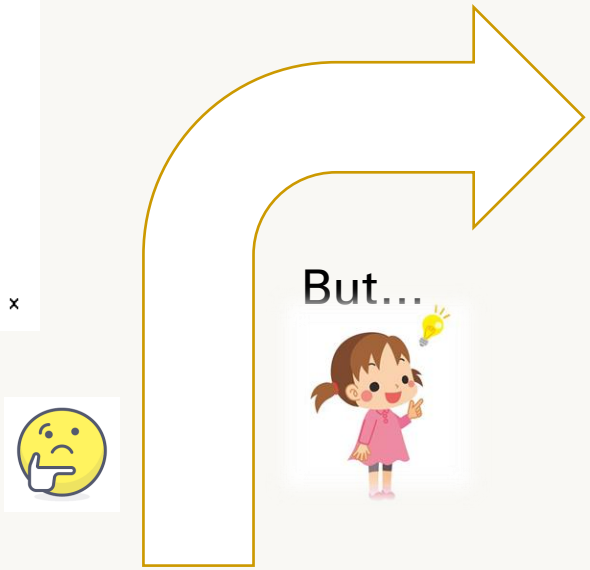
Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is **A.**

How does SVM work? (Cont.)

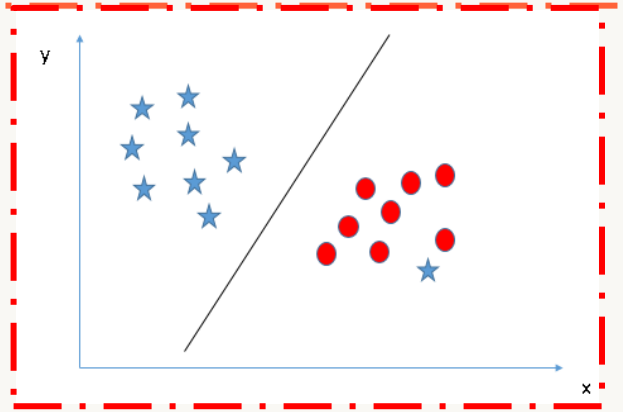
4) **Can we classify the two-classes dataset below:**



- It is NOT possible to separate the two classes using a straight line.
- Note: it is one star at other end is *like an outlier for star class*.

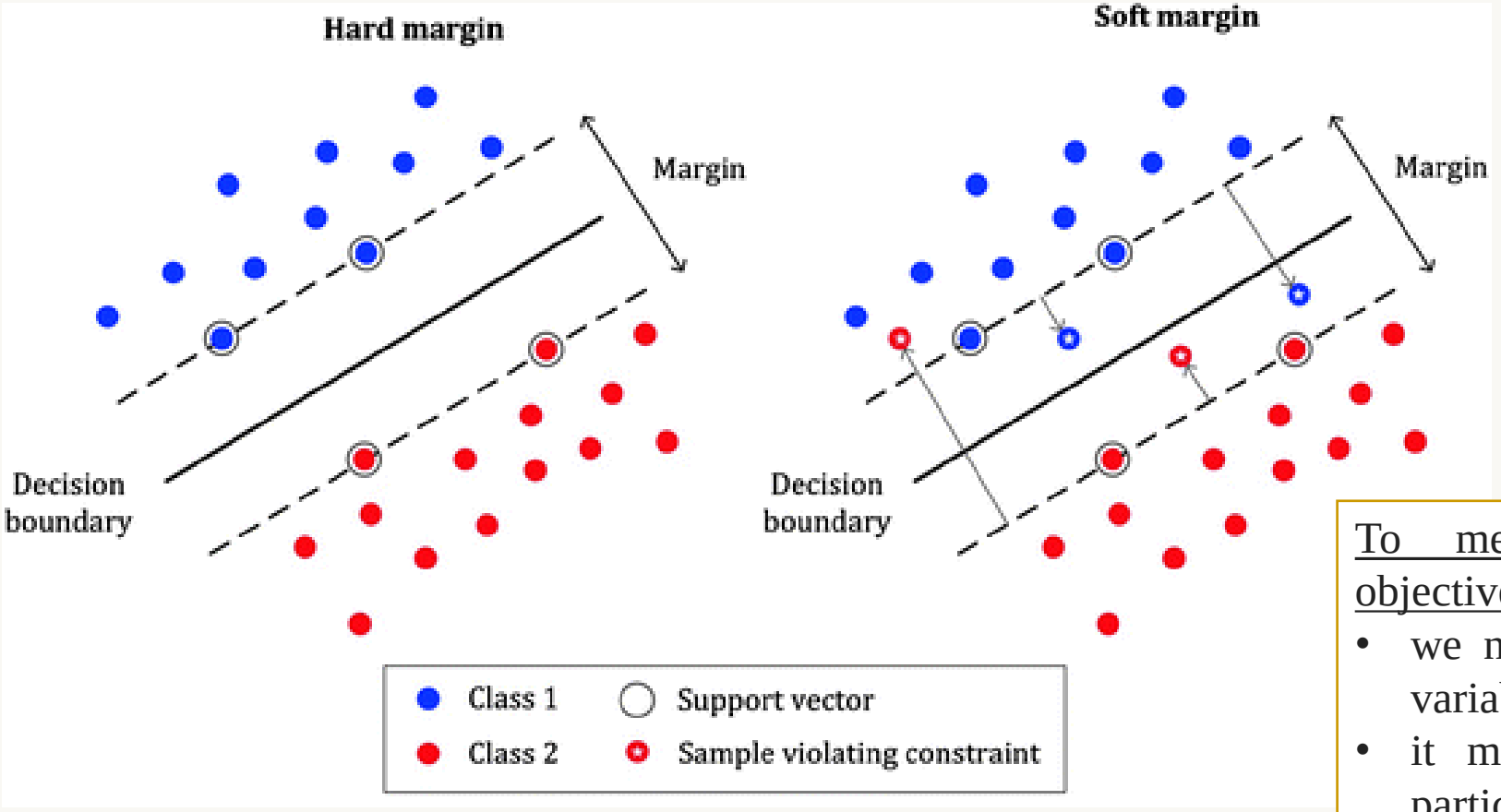


SVM algorithm has a feature to ignore outliers and finds the hyper-plane that has the maximum margin. Hence, we can say, SVM classification is robust to outliers.



How does SVM work? (Cont.)

Note:



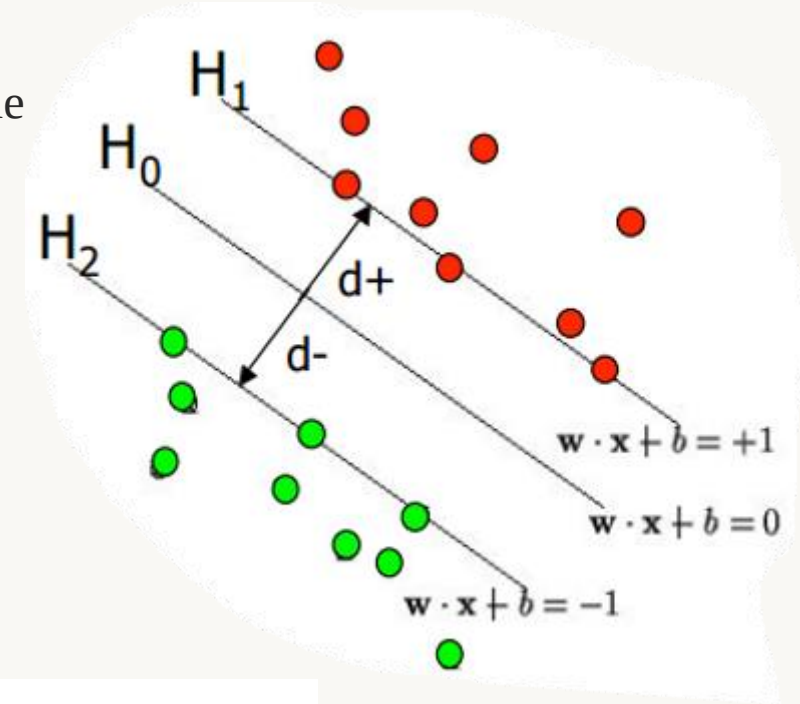
To meet the **soft margin** objective:

- we need to introduce a slack variable $\varepsilon >= 0$ for each sample;
- it measures how much any particular instance is allowed to violate the margin.

How does SVM work? (Cont.)

Note:

- The target is to find a classifier with as big margin (street width) as possible



The distance between H_0 and H_1 is then:
$|w \bullet x + b| / \|w\| = 1 / \|w\|$, so

The total distance between H_1 and H_2 is thus: $2/\|w\|$

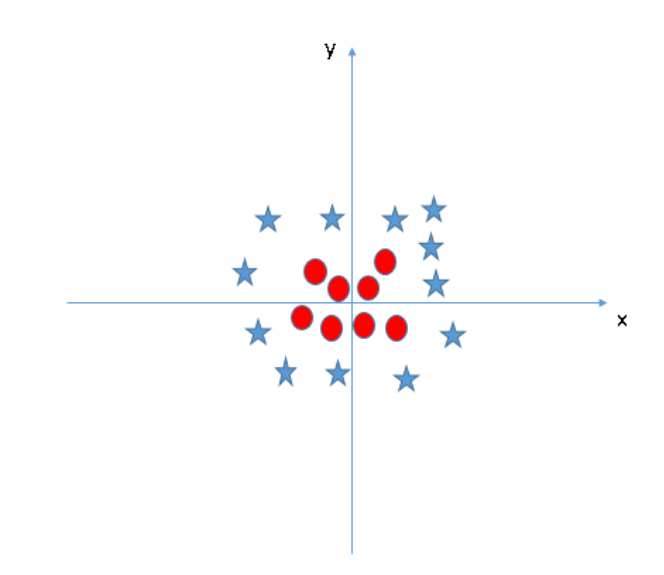
In order to maximize the margin, we thus need to minimize $\|w\|$. With the condition that there are no datapoints between H_1 and H_2:

$\mathbf{x}_i \bullet \mathbf{w} + b \geq +1$ when $y_i = +1$
$\mathbf{x}_i \bullet \mathbf{w} + b \leq -1$ when $y_i = -1$

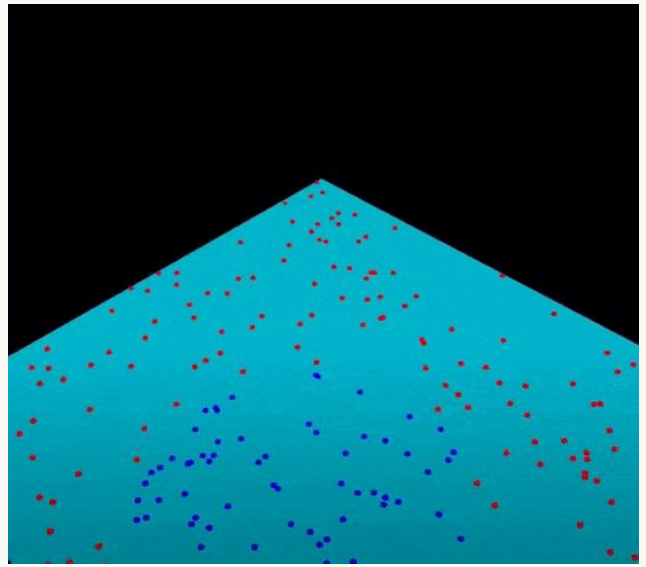
Can be combined into: $y_i(\mathbf{x}_i \bullet \mathbf{w}) \geq 1$

How does SVM work? (Cont.)

5) **Can we classify the two-classes dataset below:**



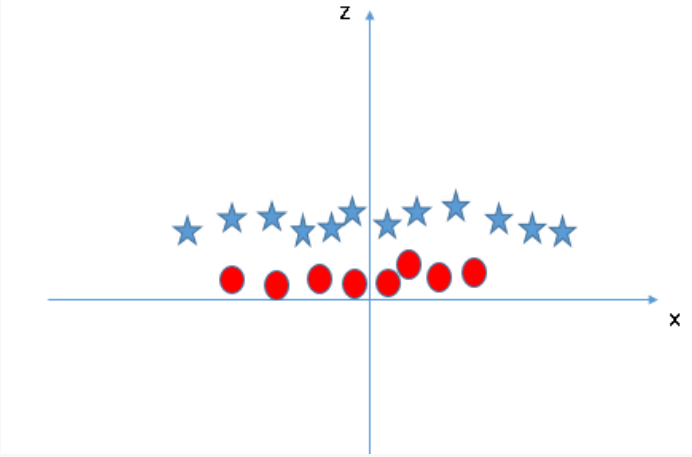
SVM will add a third dimension. Up until now we had two dimensions: x and y. It creates a new z dimension, and we rule that it be calculated a certain way that is convenient for us: $z = x^2 + y^2$ (you'll notice that's the equation for a circle).



- We, also, can't have linear hyper-plane between the two classes, so how does SVM classify these two classes?
- Note: These classes represent nonlinear data.

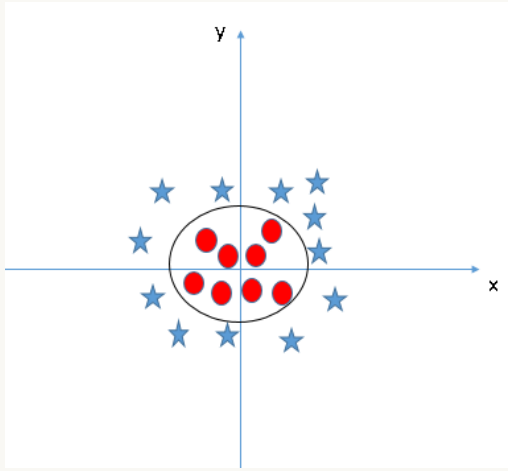
How does SVM work? (Cont.)

5) **Now, let's plot the data points on axis x and z:**



- All values for z would be positive because z is the squared sum of both x and y
- In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and stars relatively away from the origin result to higher value of z.

When we look at the hyper-plane in original input space it looks like a circle:



So, by using this trick **SVM** can be easily have a linear hyper-plane between these two classes.

Thus, the **SVM** algorithm has a technique called the **kernel** trick.

How does SVM work? (Cont.)

- **The kernel**
  - The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e., it converts not separable problem to separable problem.
  - It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then finds out the process to separate the data based on the labels or outputs you've defined.
  - However, it turns out that calculating this transformation can get computationally expensive: there can be a lot of new dimensions, each one of them possibly involving a complicated calculation. Doing this for every vector in the dataset can be a lot of work, so it'd be great if we could find a cheaper solution.

Here's a trick!!!

SVM doesn't need the actual vectors to work its magic, it actually can get by only with the dot products between them.

- That's the **kernel trick**, which allows us to avoid a lot of expensive calculations.
- Normally, the kernel is linear, and we get a linear classifier. However, by using a nonlinear kernel (like previous example) we can get a nonlinear classifier without transforming the data at all: we only change the dot product to that of the space that we want and SVM will happily chug along.

How does SVM work? (Cont.)

- **Dot Product:**

  The inner product between two vectors is the sum of the multiplication of each pair of input values. For example, the inner product of the vectors [2, 3] and [5, 6] is 2*5 + 3*6 or 28.

- **Common Kernel functions for SVM:**

  - linear $k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \cdot \mathbf{x}_2$
  - polynomial $k(\mathbf{x}_1, \mathbf{x}_2) = (\gamma\, \mathbf{x}_1 \cdot \mathbf{x}_2 + c)^d$
  - Gaussian or radial basis $k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\gamma \left\|\mathbf{x}_1 - \mathbf{x}_2\right\|^2\right)$
  - sigmoid $k(\mathbf{x}_1, \mathbf{x}_2) = \tanh(\gamma\, \mathbf{x}_1 \cdot \mathbf{x}_2 + c)$

How to implement SVM in Python?

- **Implementation Example:**

See scikit_learn_tutorial *page 64*

Following Python script uses sklearn.svm.SVC class:

```
import numpy as np
X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
y = np.array([1, 1, 2, 2])
from sklearn.svm import SVC
SVCClf = SVC(kernel='linear',gamma='scale', shrinking=False,)
SVCClf.fit(X, y)
```

Output

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=False,
  tol=0.001, verbose=False)
```

Penalty parameter C of the error term

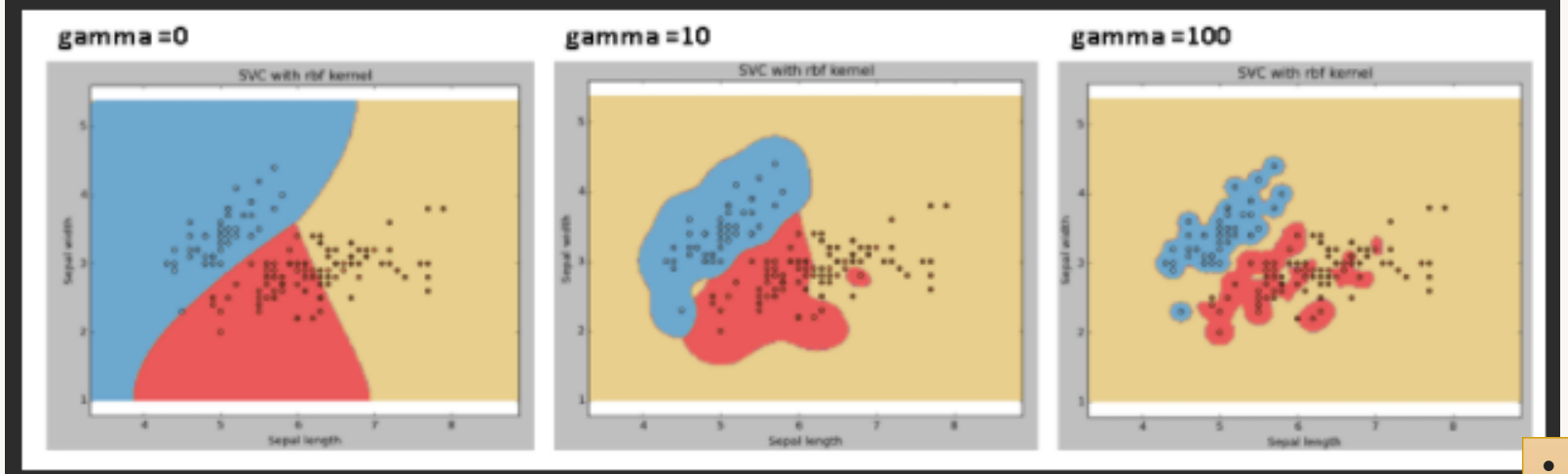
Kernel coefficient mostly for ‘rbf’, ‘poly’ and ‘sigmoid’.

various options available with kernel like, “linear”, “rbf”,”poly”

How to tune Parameters of SVM? (Cont.)

- **Some of the important parameters having higher impact on model performance, “kernel”, “gamma” and “C”:**
  - Let’s difference if we have different **gamma** values like 0, 10 or 100.:

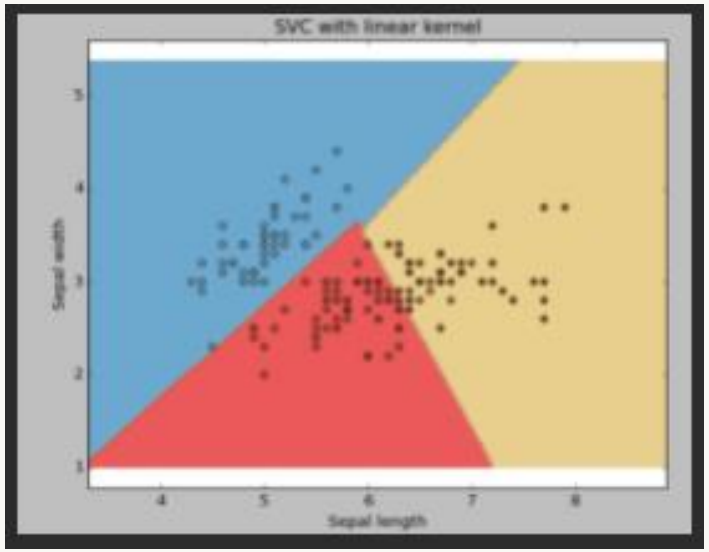
```
svc = svm.SVC(kernel='rbf', C=1,gamma=0).fit(X, y)
```



- Higher the value of gamma, will try to exact fit the as per training data set i.e., generalization error and cause over-fitting problem.

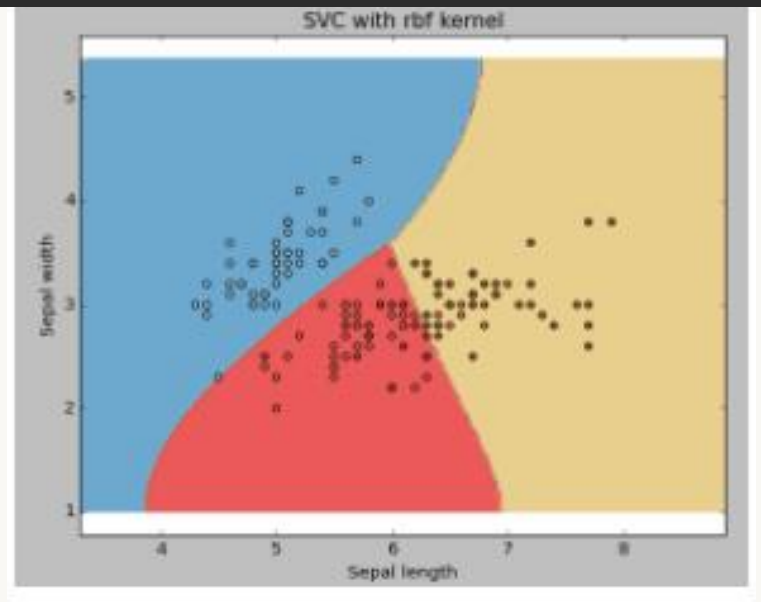
How to tune Parameters of SVM?

- If we have the following model:



Tuning the **kernel** and use ‘rbf’

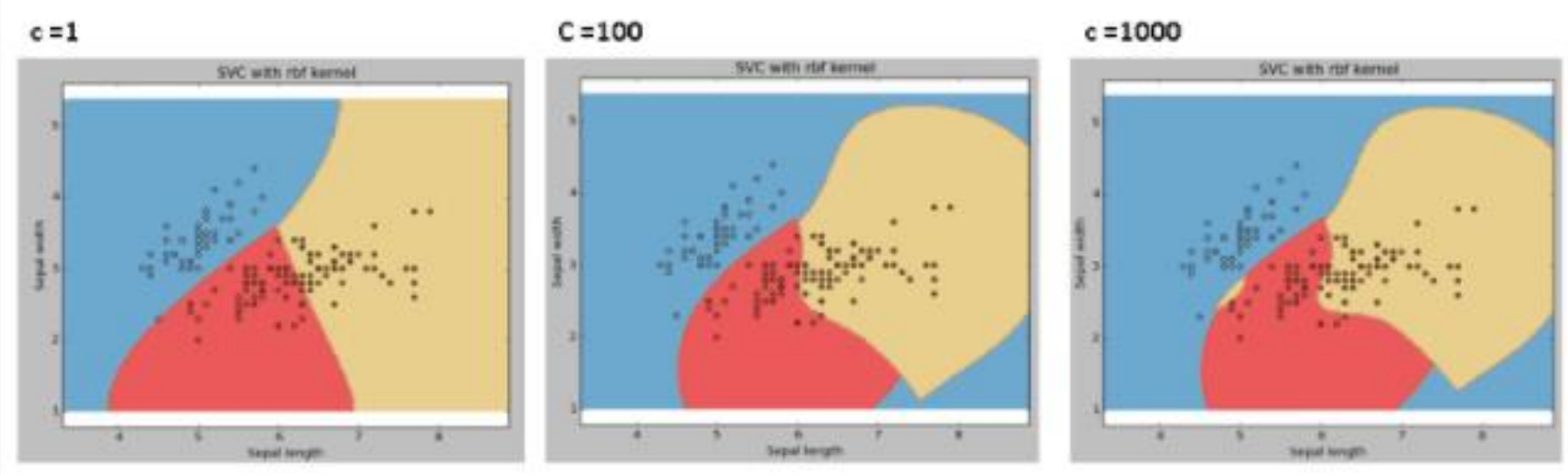
```
svc = svm.SVC(kernel='rbf', C=1,gamma=0).fit(X, y)
```



- Change the kernel type to rbf in below line and look at the impact.

How to tune Parameters of SVM?

- Penalty parameter **C** of the error term.
- It also controls the trade-off between smooth decision boundaries and classifying the training points correctly.



- In other words, the balance between keeping the margins as large as possible and limiting the margin violation is controlled by the C parameter: **a small value leads to a wider street** but more margin violation and **a higher value of C makes fewer margin violations** but ends up with a smaller margin and overfitting.

We should always look at the cross-validation score to have effective combination of these parameters and avoid over-fitting.

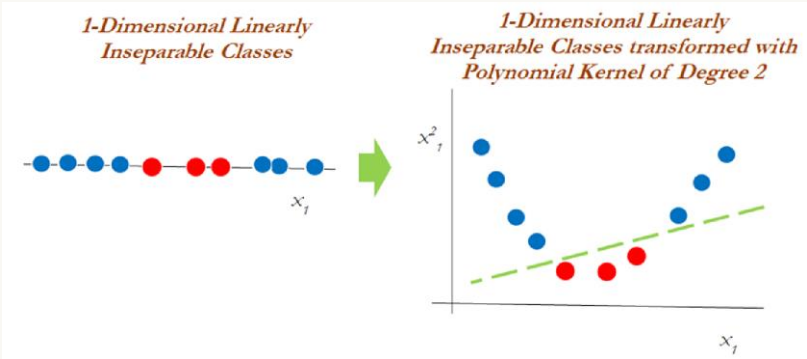
Pros and Cons associated with SVM

- **Pros**:
  - It works really well with a clear margin of separation.
  - It is effective in high dimensional spaces.
  - It is effective in cases where the number of dimensions is greater than the number of samples.
  - It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient (during prediction phase NOT training phase).

- **Cons**:
  - It doesn’t perform well when we have large data set because the required training time is higher.
  - It also doesn’t perform very well, when the data set has more noise i.e., target classes are overlapping
  - SVM doesn’t directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is included in the related SVC method of Python scikit-learn library.

Summary

- Support vector machines (SVM) use a mechanism called kernels, which essentially calculate distance between two observations.
- The SVM algorithm then finds a decision boundary that maximizes the distance between the closest members of separate classes.
- SVM with a linear kernel is similar to logistic regression. Therefore, in practice, the benefit of SVM's typically comes from using non-linear kernels to model non-linear decision boundaries.
- SVM's are memory intensive, trickier to tune due to the importance of picking the right kernel, and don't scale well to larger datasets. Currently in the industry, random forests are usually preferred over SVM's.



Any questions?