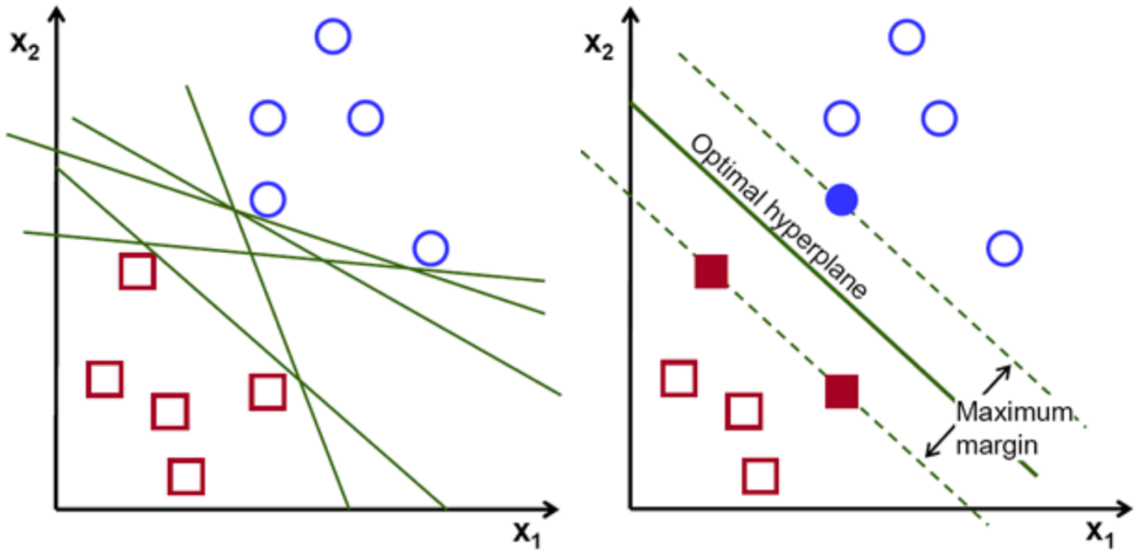
1.Q1: What is the margin and support vectors? Q2: How does SVM deal with non-separable data? Q3: What is a kernel? Q4: How does a kernel relate to feature vectors?
Ans:
Q1. Support Vectors:
Definition: Support-vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers' detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier.

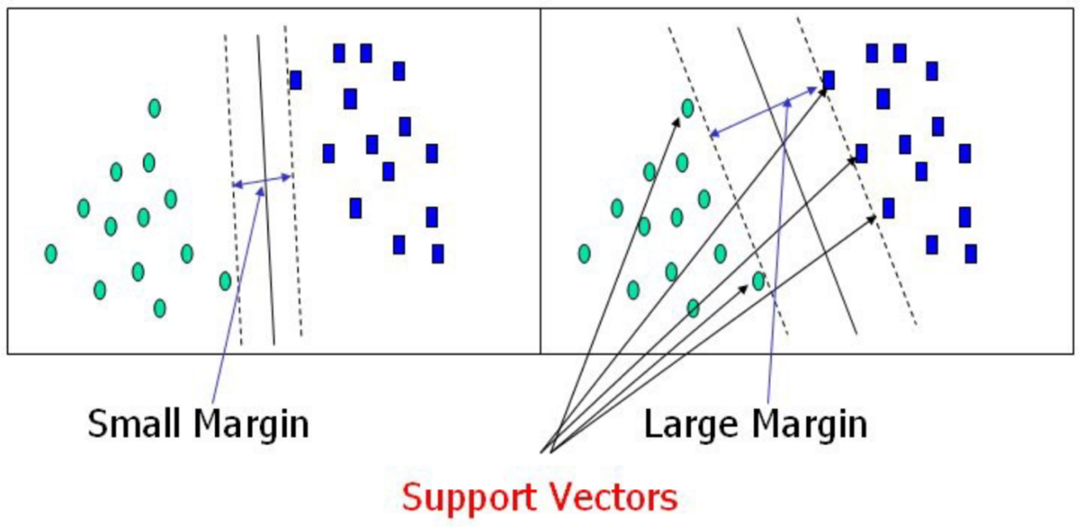
Objective:
The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points.



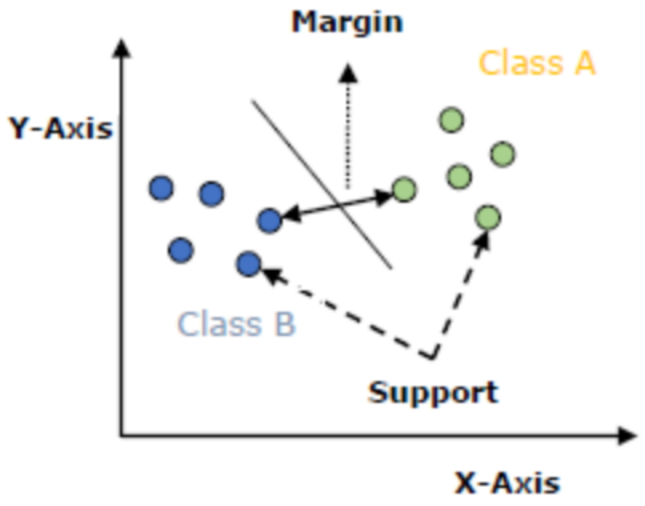
Possible hyperplanes

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.



Margins:
It may be defined as the gap between two lines on the closet data points of different classes. It can be calculated as the perpendicular distance from the line to the support vectors. Large margin is considered as a good margin and small margin is considered as a bad margin.



Hard Margin: If the training data is linearly separable, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the "margin", and the maximum-margin hyperplane is the hyperplane that lies halfway between them.
Generalized Form:

"Minimize $\|\mathbf{w}\|$ subject to $y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1$ for $i = 1, \ldots, n$."

The $\mathbf{w}$ and b that solve this problem determine our classifier, $\mathbf{x} \mapsto \operatorname{sgn}(\mathbf{w}^T\mathbf{x} - b)$ where $\operatorname{sgn}(\cdot)$ is the sign function.

wTx−b=1

Soft Margin:

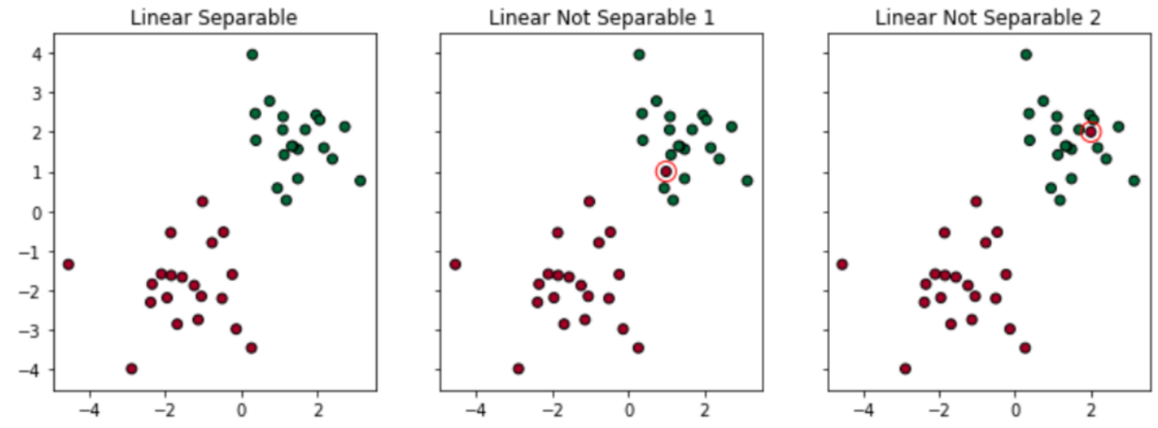
To extend SVM to cases in which the data are not linearly separable, the hinge loss function is helpful.
Max (0,1-yi(Wt Xi -b))
N

ote that yiyi is the i-th target (i.e., in this case, 1 or −1), and wTxi−bWt Xi is the i-th output. This function is zero if the constraint in (1) is satisfied, in other words, if Xi xi lies on the correct side of the margin. For data on the wrong side of the margin, the function's value is proportional to the distance from the margin.
The goal of the optimization then is to minimize

$$\lambda\|\mathbf{w}\|^2 + \left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i - b)\right)\right]$$

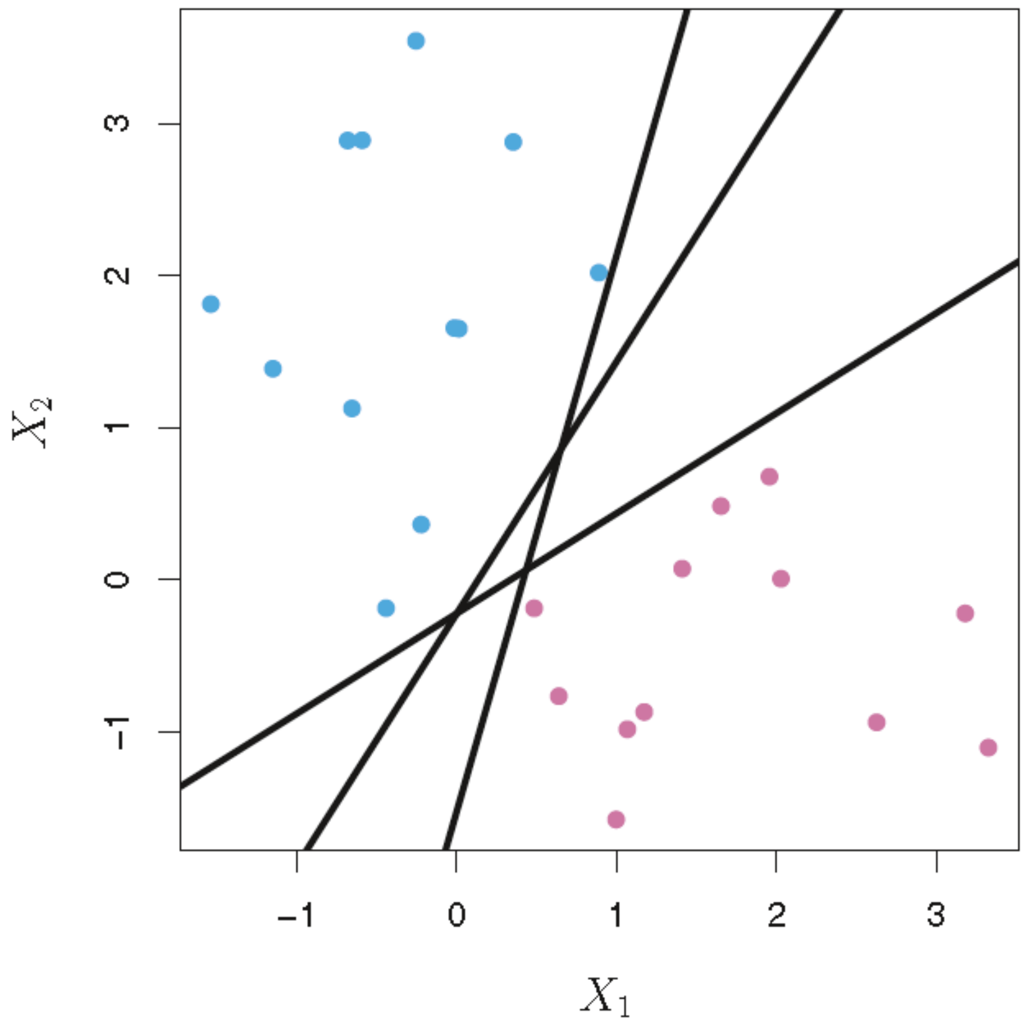
where the parameter λ >0 ddetermines the trade-off between increasing the margin size and ensuring that the Xi xi lie on the correct side of the margin. Thus, for sufficiently small λ values of λ

it will behave similar to the hard-margin SVM, if the input data are linearly classifiable, but will still learn if a classification rule is viable or not.
Q2. In the linearly separable case, SVM is trying to find the hyperplane that maximizes the margin, with the condition that both classes are classified correctly. But in reality, datasets are probably never linearly separable, so the condition of 100% correctly classified by a hyperplane will never be met.
SVM address non-linearly separable cases by introducing two concepts: Soft Margin and Kernel Tricks.
Let’s use an example. If I add one red dot in the green cluster, the dataset becomes linear non separable anymore. Two solutions to this problem:



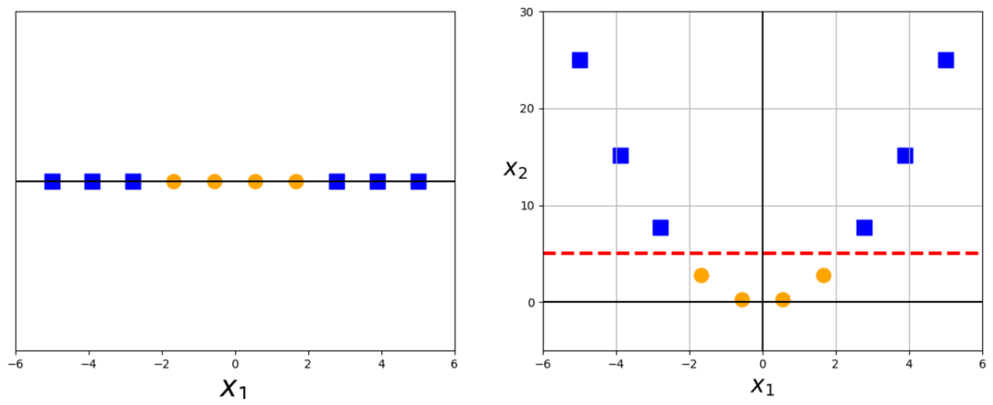
Soft Margin: try to find a line to separate, but tolerate one or few misclassified dots (e.g. the dots circled in red)
Kernel Trick: try to find a non-linear decision boundary
By combining the soft margin (tolerance of misclassifications) and kernel trick together, Support Vector Machine is able to structure the decision boundary for linear non-separable cases.
Hyper-parameters like C or Gamma control how wiggling the SVM decision boundary could be.
The higher the C, the more penalty SVM was given when it misclassified, and therefore the less wiggling the decision boundary will be
The higher the gamma, the more influence the feature data points will have on the decision boundary, thereby the more wiggling the boundary will be.
Q3. Kernel:
In practice, SVM algorithm is implemented with kernel that transforms an input data space into the required form. SVM uses a technique called the kernel trick in which kernel takes a low dimensional input space and transforms it into a higher dimensional space. In simple words, kernel converts non-separable problems into separable problems by adding more dimensions to it. It makes SVM more powerful, flexible and accurate. The following are some of the types of kernels used by SVM.
Linear Kernel: It can be used as a dot product between any two observations. The formula of linear kernel is as below −
K(x,xi)=sum(x∗xi)K(x,xi)=sum(x∗xi)
From the above formula, we can see that the product between two vectors say x & xi is the sum of the multiplication of each pair of input values.
Polynomial Kernel: It is more generalized form of linear kernel and distinguish curved or nonlinear input space. Following is the formula for polynomial kernel −
k(X,Xi)=1+sum(X∗Xi)^dk(X,Xi)=1+sum(X∗Xi)^d
Here d is the degree of polynomial, which we need to specify manually in the learning algorithm.
Radial Basis Function (RBF) Kernel: RBF kernel, mostly used in SVM classification, maps input space in indefinite dimensional space. Following formula explains it mathematically −
K(x,xi)=exp(−gamma∗sum(x−xi^2))K(x,xi)=exp(−gamma∗sum(x−xi^2)).
Here, gamma ranges from 0 to 1. We need to manually specify it in the learning algorithm. A good default value of gamma is 0.1.
Q4.
Feature Vectors:
Definition:
In machine learning, feature vectors are used to represent numeric or symbolic characteristics, called features, of an object in a mathematical, easily analyzable way. They are important for many different areas of machine learning and pattern processing. Machine
learning algorithms typically require a numerical representation of objects in order for the algorithms to do processing and statistical analysis. Feature vectors are the equivalent of vectors of explanatory variables that are used in statistical procedures such as linear regression.
Kernel Trick:
Support vector classification: It is based on a very natural way that one might attempt to classify data points into various target classes. If the classes in our training data can be separated by a line or some boundary, then we can just classify the data depending on what side of this decision boundary the data lies on.

In the following 2-d example, we can separate the data with any of the three lines, and then just assign classes based on whether the observation lies above or below the line. The data are 2-dimensional vectors specified by the features X1 and X2 with class labels as either y =1 (blue) or y = 0 (red).



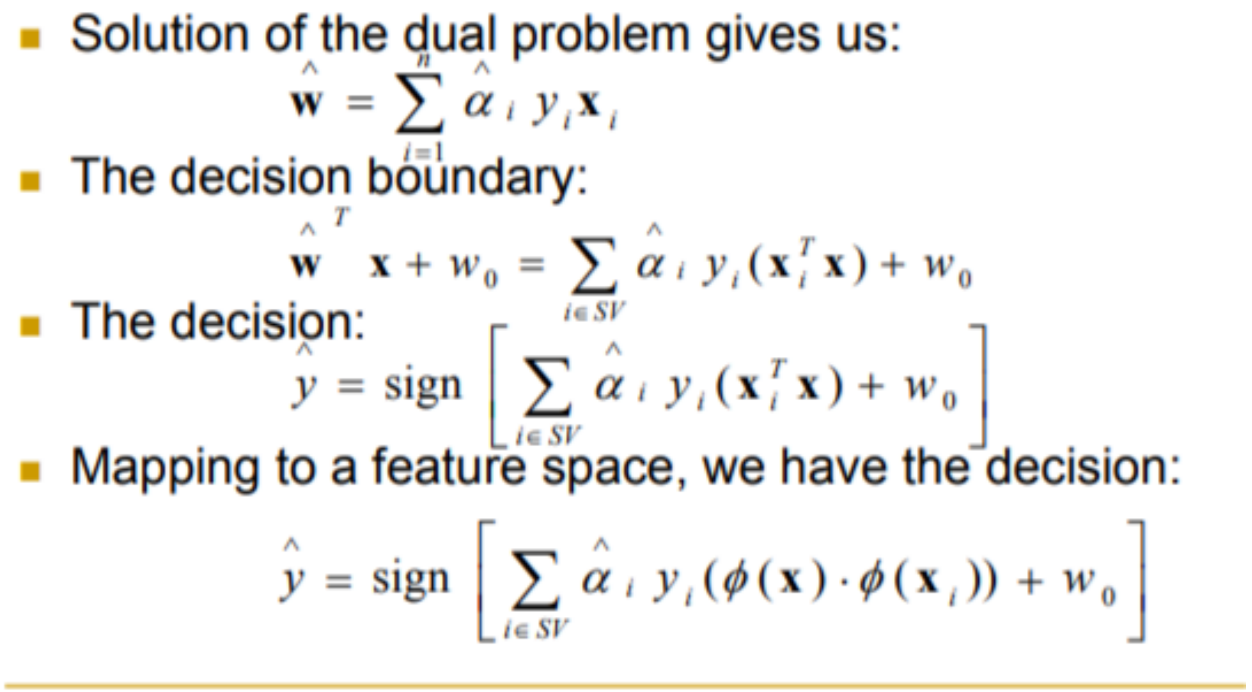
An example dataset showing classes that can be linearly separated.

Non-Linear Transformations: If the data is not linearly separable in the original, or input, space then we apply transformations to the data, which map the data from the original space into a higher dimensional feature space. The goal is that after the transformation to the higher dimensional space, the classes are now linearly separable in this higher dimensional feature space. We can then fit a decision boundary to separate the classes and make predictions. The decision boundary will be a hyperplane in this higher dimensional space.
It is obviously hard to visualize higher dimensional data, and so we first focus on some transformations applied to 1-dimensional data. In this example, the picture on the left shows our original data points. In 1-dimension, this data is not linearly separable, but after applying the transformation $\phi(x) = x^2$ and adding this second dimension to our feature space, the classes become linearly separable.



This data becomes linearly separable after a quadratic transformation to 2-dimensions.

Kernel Trick:
The kernel trick provides a solution to this problem. The "trick" is that kernel methods represent the data only through a set of pairwise similarity comparisons between the original data observations x (with the original coordinates in the lower dimensional space), instead of explicitly applying the transformations $\phi(x)$ and representing the data by these transformed coordinates in the higher dimensional feature space.
In kernel methods, the data set X is represented by an n x n kernel matrix of pairwise similarity comparisons where the entries (i, j) are defined by the kernel function: k(xi, xj). This kernel function has a special mathematical property. The kernel function acts as a modified dot product.
The kernel function here is the polynomial kernel $k(a,b) = (a^T * b)^2$
The ultimate benefit of the kernel trick is that the objective function we are optimizing to fit the higher dimensional decision boundary only includes the dot product of the transformed feature vectors. Therefore, we can just substitute these dot product terms with the kernel function, and we don't even use $\phi(x)$.



- Solution of the dual problem gives us:

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \hat{\alpha}_i y_i \mathbf{x}_i$$

- The decision boundary:

$$\hat{\mathbf{w}}^T \mathbf{x} + w_0 = \sum_{i \in SV} \hat{\alpha}_i y_i (\mathbf{x}_i^T \mathbf{x}) + w_0$$

- The decision:

$$\hat{y} = \text{sign}\left[\sum_{i \in SV} \hat{\alpha}_i y_i (\mathbf{x}_i^T \mathbf{x}) + w_0\right]$$

- Mapping to a feature space, we have the decision:

$$\hat{y} = \text{sign}\left[\sum_{i \in SV} \hat{\alpha}_i y_i (\phi(\mathbf{x}) \cdot \phi(\mathbf{x}_i)) + w_0\right]$$

In the bottom equation, we replace the dot product of the transformed vectors with the kernel function.

3. The circle equation expands into five terms

   $0=x^2{}_1 + x^2 - 2ax1 - 2bx2 + (a^2 + b^2 - r^2)$

   corresponding to weights w = (2a, 2b, 1, 1) and intercept $a^2 + b^2 - r^2$. This shows that a circular boundary is linear in this feature space, allowing linear separability.
   In fact, the three features x1, x2, $x^2{}_1$,x^2 suffice.

4. The (axis-aligned) ellipse equation expands into six terms
   $0=cx^2{}_1 + dx^2 - 2acx1 - 2bdx2 + (a^2c+b^2d-1)$

   corresponding to weights w = (2ac, 2bd, c, d, 0) and intercept $a^2 + b^2 - r^2$. This shows that an elliptical boundary is linear in this feature space, allowing linear separability.
   In fact, the four features x1, x2, $x^2{}_1$, x^2 suffice for any axis-aligned ellipse.

Link for remaining answers:
https://colab.research.google.com/drive/1y-JorjjQB155yhWQ4BGWRa6nxc-F_DuJ?usp=sharing