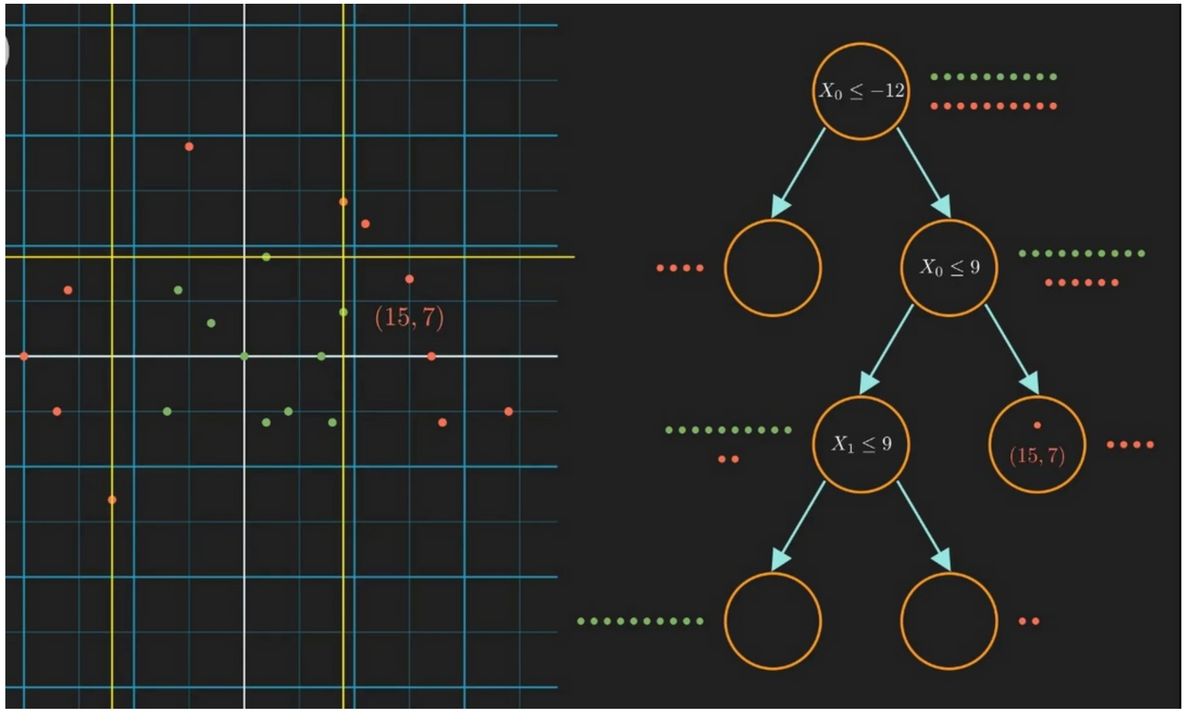
Decision Tree

A **decision tree** is a **supervised machine learning** algorithm that uses a **tree-like structure** to model decisions and predict outcomes. It is a flowchart-like structure where **internal nodes represent features or attributes**, **branches represent decision rules**, and **leaf nodes represent the final predictions or outcomes**.

The construction of a decision tree involves recursively partitioning the dataset based on different features and splitting criteria. The algorithm selects the best feature to split the data, based on metrics such as **information gain** or **Gini impurity**, which measure the purity or homogeneity of the subsets resulting from the split.

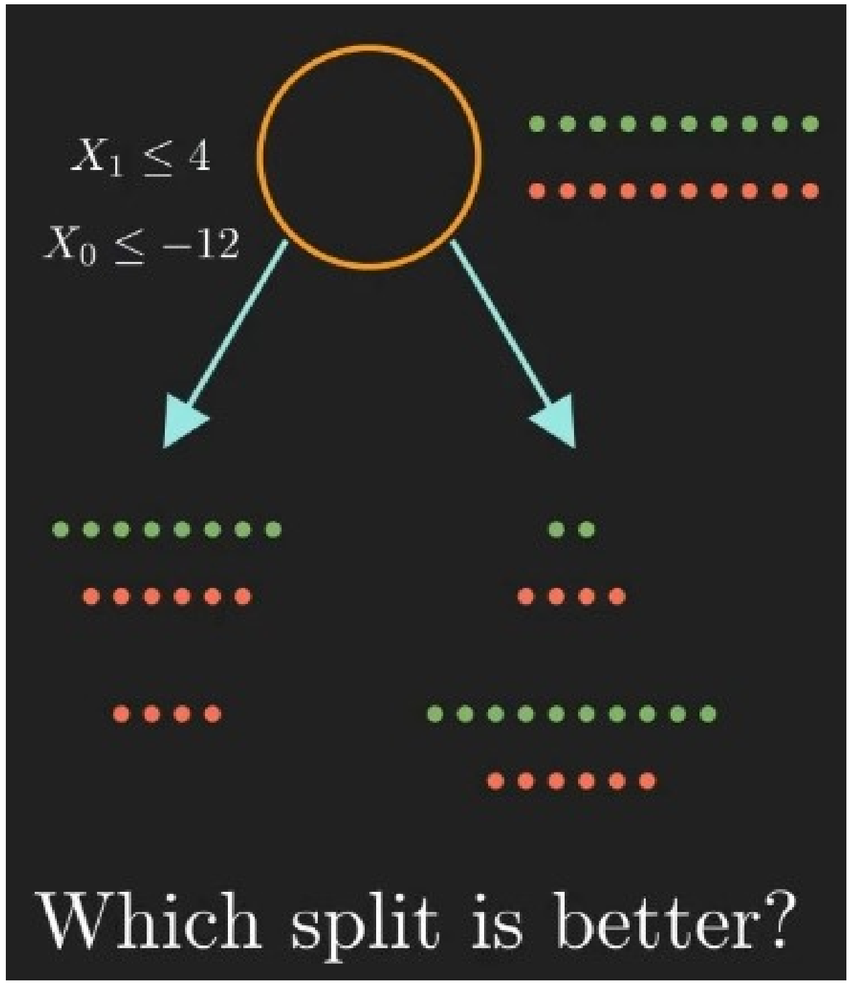


Algorithm 1 : Entropy and Information Gain

$$Entropy = \sum - p_i \log(p_i)$$

p_i = probability of class i

Information Gain = Entropy(S) - (Weight * Entropy(each feature)

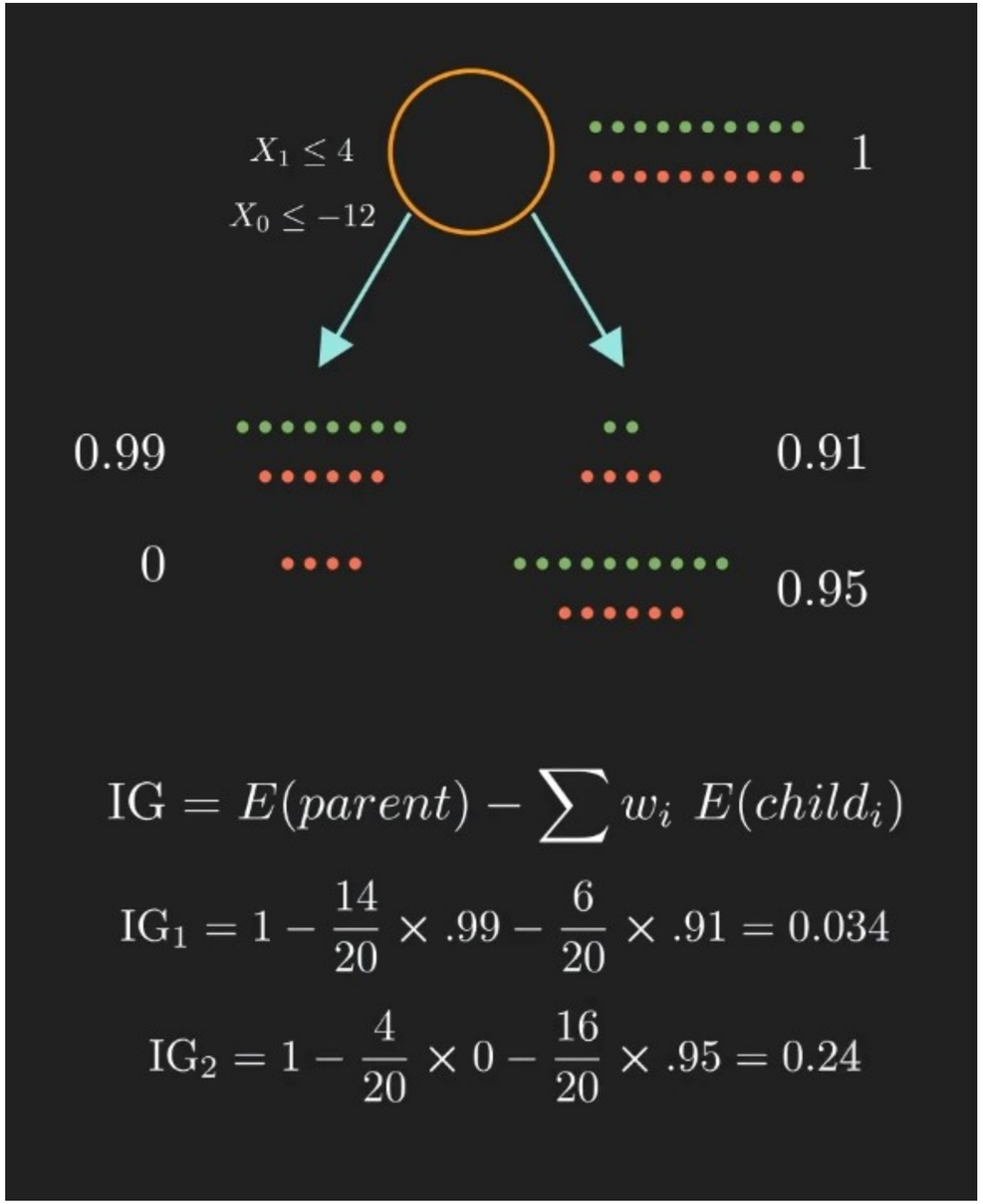


Algorithm 2 : Gini Impurity Index and Gini Gain

Gini Impurity Index= 1- ∑P(i)^2

Gini Gain = Gini impurity before split - sum (Weight*Gini impurity of each subset)

Weight of a feature = Number of samples in the feature/ Total samples before split



Let's consider thsis data present here. We will be trying a fit a decision tree on this data using Gini Impurity as the decision criteria. Same can be done for entropy and Information gain.

Email	Sender	Subject	Body Length	Attachment	Label
1	John	Free Offer	80	No	Spam
2	John	Urgent	120	Yes	Spam
3	Mary	Important	50	No	Not Spam
4	Mary	Free Offer	70	Yes	Spam
5	Peter	Urgent	90	Yes	Spam
6	Peter	Important	60	No	Not Spam
7	John	Urgent	100	No	Not Spam
8	Peter	Free Offer	40	Yes	Not Spam
9	Mary	Urgent	75	No	Not Spam
10	John	Important	55	Yes	Not Spam

We will be trying to first the column and best criteria on which the first split should be made. We have a jupyter notebook <> where the decision tree is plotted to help us verify.

How the decision tree would have processed?

① Initial gini index of column "Label".
4 spam, 6 non-spam.

Gini: $1 - (4/10)^2 - (6/10)^2 = 0.48$

② Based on Column "Sender".
Sender = 0. and ≠ 0.

Labels: 1, 1, 0, 0 Labels: 0, 1, 1, 0, 0, 0

1st Gini = $1 - 0.25 - 0.25 = 0.5$
2nd Gini = $1 - 0.111 - 0.444 = 0.444$
Gini Gain = $0.48 - \frac{4}{10} \times 0.5 - \frac{6}{10} \times 0.444$
$= 0.0136$

Sender = 1 and ≠ 0
Labels: 0, 1, 0 Labels: 1, 1, 1, 0, 0, 0, 0
1st Gini = $1 - (\frac{1}{3})^2 - (\frac{2}{3})^2 = 0.444$
2nd Gini = $1 - (\frac{3}{7})^2 - (\frac{4}{7})^2 = 0.489$
Gini Gain = $0.48 - \frac{3}{10} \times 0.444 - \frac{7}{10} \times 0.489$
$= -0.0045$

Sender = 2 and ≠ 0

③ Based on Column "Subject".
Subject = 0, 1, 2.

⑤ Based on "Attachment" = 0 or 1.

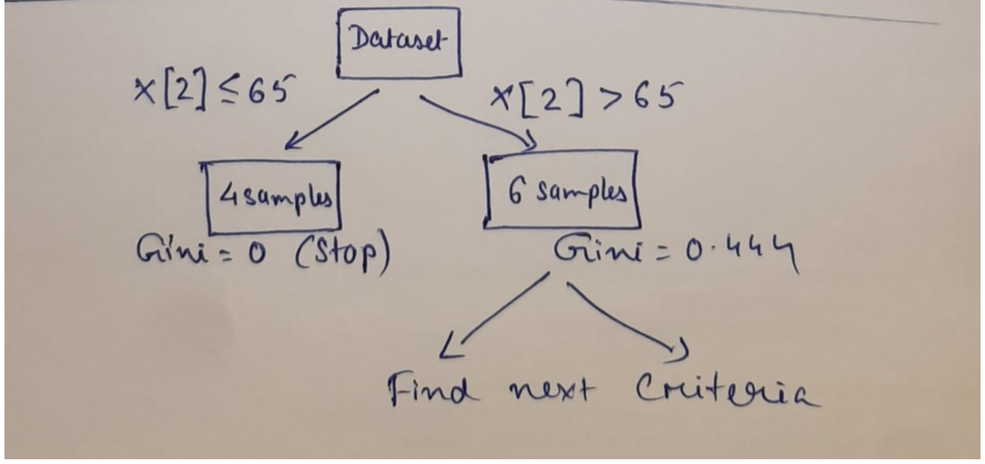
④ Based on Column Body Length
Sort it: 40, 50, 55, 60, 70, 75, 80, 90, 100, 120.
Splits at mid points.
Body length ≤ 45 & > 45

Labels: 0 Labels: (4)(1s), (5)0s.
Gini = 1 - 0 - 1 = 0 Gini = $1 - (\frac{4}{9})^2 - (\frac{5}{9})^2$
$= 0.493$

Gini Gain = $0.48 - 0 \times \frac{1}{10} - 0.493 \times \frac{9}{10}$
$= 0.035$

B.L ≤ 52.5
BL ≤ 57.5

BL ≤ 65 BL > 65
Labels: 0, 0, 0, 0 1 1 1 1 0 0
Gini = 0 Gini = 0.444
Gini Gain = $0.48 - 0.444 \times 0.6 = 0.214$



The decision tree algorithm

- Select the best feature: The first step is to select the feature that best splits the data into classes or reduces the variance of the target variable for regression tasks. The most common measure for selecting the best feature is the information gain or the Gini impurity.

- Split the data: Once the best feature is selected, the data is split into subsets based on the values of the selected feature. Each subset corresponds to a child node of the current node in the tree.
- Recursively repeat steps 1 and 2: The above steps are repeated recursively for each child node until a stopping criterion is met. This stopping criterion can be a predefined maximum depth of the tree, a minimum number of samples required to split an internal node or a minimum number of samples required to be at a leaf node.

- Predict the target variable: Once the decision tree is built, the target variable can be predicted by traversing the tree from the root node to a leaf node based on the decision rules at each node.

- Prune the tree: To avoid overfitting, the tree can be pruned by removing unnecessary nodes or subtrees that do not improve the performance on a validation set.

Code:

Notebook Link

Bayes

The **Naive Bayes theorem** is a statistical technique used for classification and prediction tasks. It is based on Bayes' theorem, which describes the probability of an event occurring given that another related event has already occurred.

The Naive Bayes algorithm assumes that the features (or attributes) used to predict the class of an instance are independent of each other. This assumption is often simplified as "naive" because features are often correlated to some extent in real-world scenarios.

The Naive Bayes algorithm works by calculating the probability of a given class, given a set of input features. It does this by multiplying the conditional probabilities of each feature given the class. The class with the highest probability is then assigned as the prediction for that input.

Here's the formula for Naive Bayes:

P(class | features) = (P(class) * P(features | class)) / P(features)

- P(class | features) is the probability of the class given the input features
- P(class) is the prior probability of the class (before observing the input features)
- P(features | class) is the conditional probability of the features given the class
- P(features) is the prior probability of the input features (before knowing the class)

To use Naive Bayes for classification, we need to train the model first. This involves estimating the prior probabilities and conditional probabilities from a labeled dataset. Then, when new instances are presented, we can use these probabilities to calculate the probability of each class and make predictions.

Naive Bayes has proven to be an effective algorithm for text classification, spam filtering, and other tasks where the input features are discrete and independent. However, it may not work well when the features are highly correlated, or when the distribution of the features is complex and not well understood.

Estimate the value of Y given that $X = (0, 2)$

X_1	X_2	Y
0	0	0
0	1	1
1	2	1
0	0	1
2	2	0
1	1	0
0	2	1
2	0	0
2	1	0
1	0	0

Let's compute $P(Y=0|X=(0,2))$ and $P(Y=1|X=(0,2))$ ...

$$P(Y=0) = \frac{\#Y=0}{\#Y=0+\#Y=1} = \frac{6}{10}$$

$$P(Y=1) = \frac{\#Y=1}{\#Y=0+\#Y=1} = \frac{4}{10}$$

$$P(X=(0,2)|Y=1) = \frac{1}{4}$$

$$P(X=(0,2)|Y=0) = 0$$

Estimated value of Y is 1 given $X=(0,2)$

P(Y|X) = P(Y)*P(X|Y)/P(X)

X_1	X_2	Y
0	0	0
0	1	1
1	2	1
0	0	1
2	2	0
1	1	0
0	2	1
2	0	0
2	1	0
1	0	0

What if the features do not match our required test data exactly?

We *Naively* assume that our features are independent.

Let's compute $P(Y=0|X=(0,2))$ and $P(Y=1|X=(0,2))$ ...

$$P(Y=0) = \frac{\#Y=0}{\#Y=0+\#Y=1} = \frac{6}{10}$$

$$P(Y=1) = \frac{\#Y=1}{\#Y=0+\#Y=1} = \frac{4}{10}$$

$$P(X=(0,2)|Y=1) = P(X_1=0|Y=1) * P(X_2=2|Y=1) = \frac{3}{4}\ \frac{2}{4}$$

$$P(X=(0,2)|Y=0) = P(X_1=0|Y=0) * P(X_2=2|Y=0) = \frac{1}{6}\ \frac{1}{6}$$

$$\frac{3}{4} * \frac{2}{4} > \frac{1}{6} * \frac{1}{6}$$

So, the estimated value of Y is 1

X_1	X_2	Y
0	0	0
0	1	1
1	2	1
0	0	1
2	2	0
1	1	0
0	2	1
2	0	0
2	1	0
1	0	0

Dealing with Continuous features

1. Discretization

$$age \in (11, 50) \longrightarrow age \in \{1, 2, 3, 4\}$$

Code:

Notebook Link

SUPPORT VECTOR MACHINE

Support Vector Machines (SVM) is a powerful and widely used algorithm in machine learning, particularly for classification tasks. SVM is known for its ability to handle high-dimensional data and can be effective even with limited training samples.

Imagine you have a dataset with two classes of points, and your goal is to find a line that can separate these two classes as best as possible. SVM is an algorithm that helps you do just that.

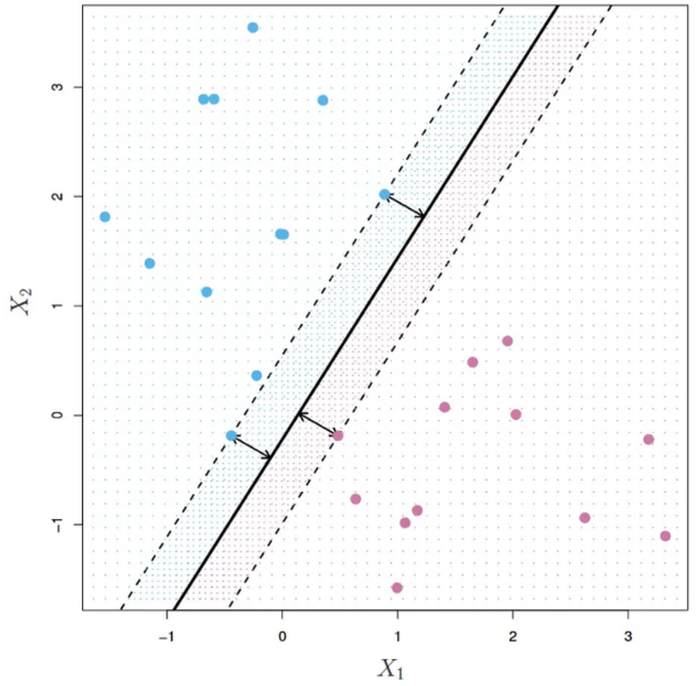
Data Preparation:

You start with a set of data points, each with features (attributes) and a class label indicating their class (e.g., red or blue).

SVM works best when the data is linearly separable, meaning you can draw a straight line to separate the classes. However, SVM can handle some degree of overlapping as well.

Finding the Optimal Hyperplane:

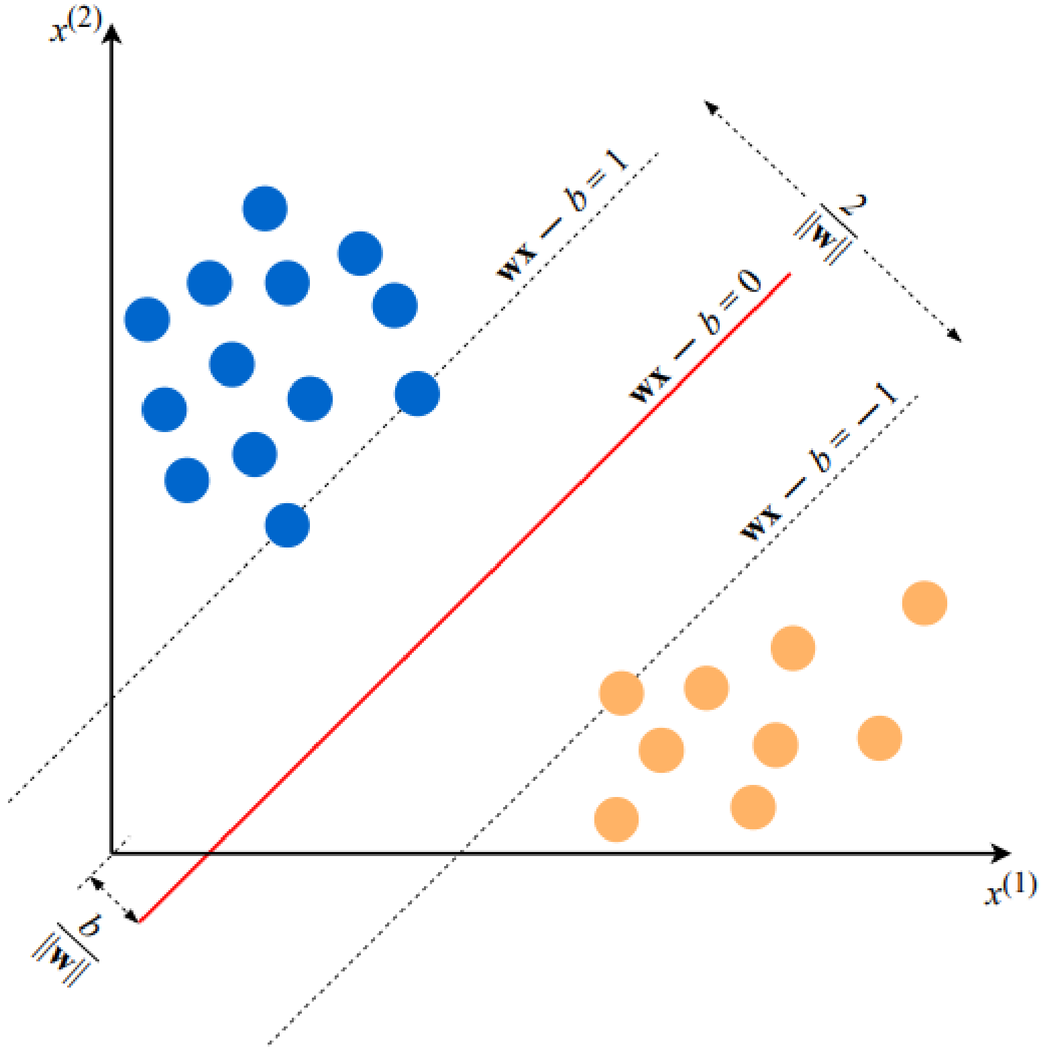
SVM aims to find the best line, called the hyperplane, that separates the classes while maximizing the margin. The margin is the distance between the hyperplane and the closest data points from each class. The larger the margin, the better the separation. SVM selects the hyperplane that has the maximum distance to the closest data points from both classes. These data points are called support vectors.



The equation of the hyperplane is given by two parameters, a real-valued vector w of the same dimensionality as our input feature vector x, and a real number b like this:

$$wx - b = 0,$$

where the expression wx means w(1)x(1) + w(2)x(2) + ... + w(D)x(D), and D is the number of dimensions of the feature vector x.



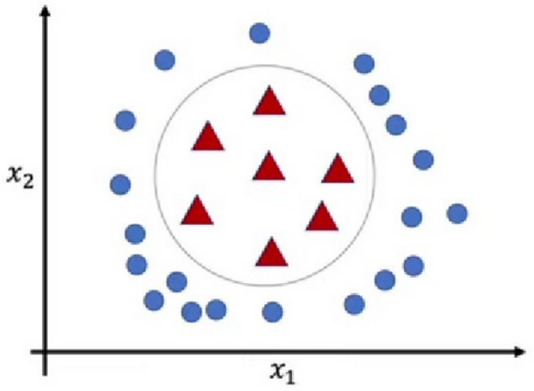
- wxi - b >= 1 if yi = +1, and
- wxi - b <=-1 if yi = -1

yi(wxi+b)>=1

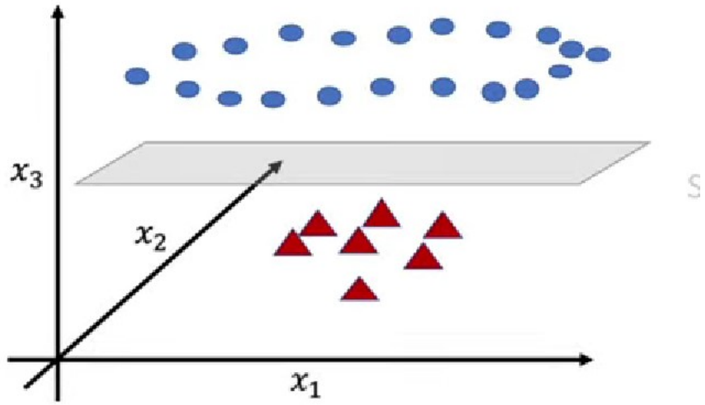
SVM-Kernels:

Sometimes, the original data cannot be separated by a straight line. In such cases, SVM uses kernel tricks to transform the data into a higher-dimensional space.

By transforming the data, SVM can find a hyperplane that separates the classes effectively.

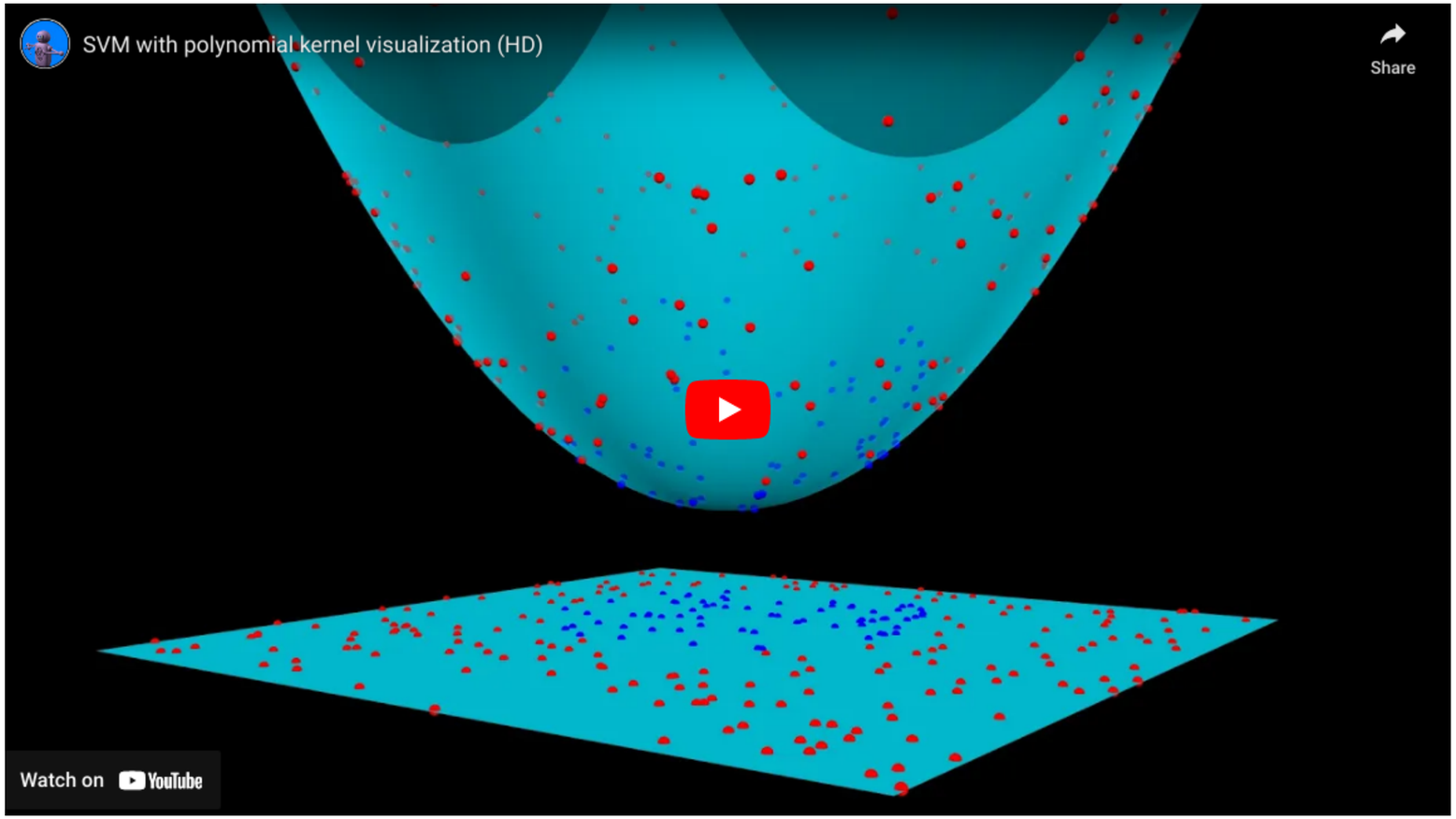


SVM in 2 dimensions



SVM in 3 dimensions

Make from this whatever you want to, you will understand this better once you go through the following slides.

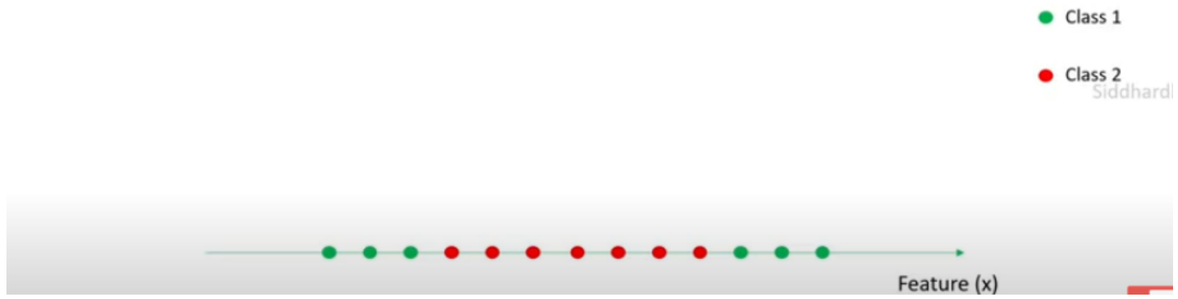


The kernel function generally transforms the training set of data so that a non-linear decision surface can be transformed to a linear equation in higher no. of dimension spaces. It return the inner product between two points in a standard feature dimension.

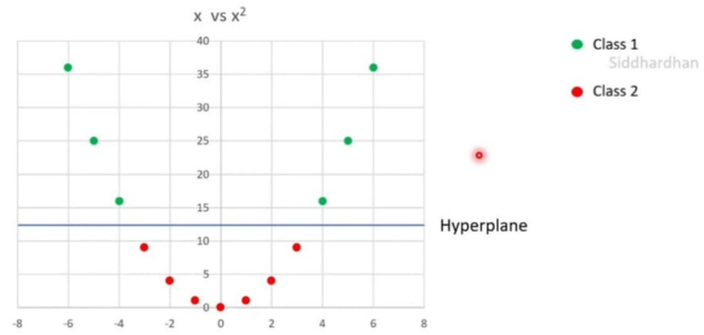
Examples:

1)Linear Kernel

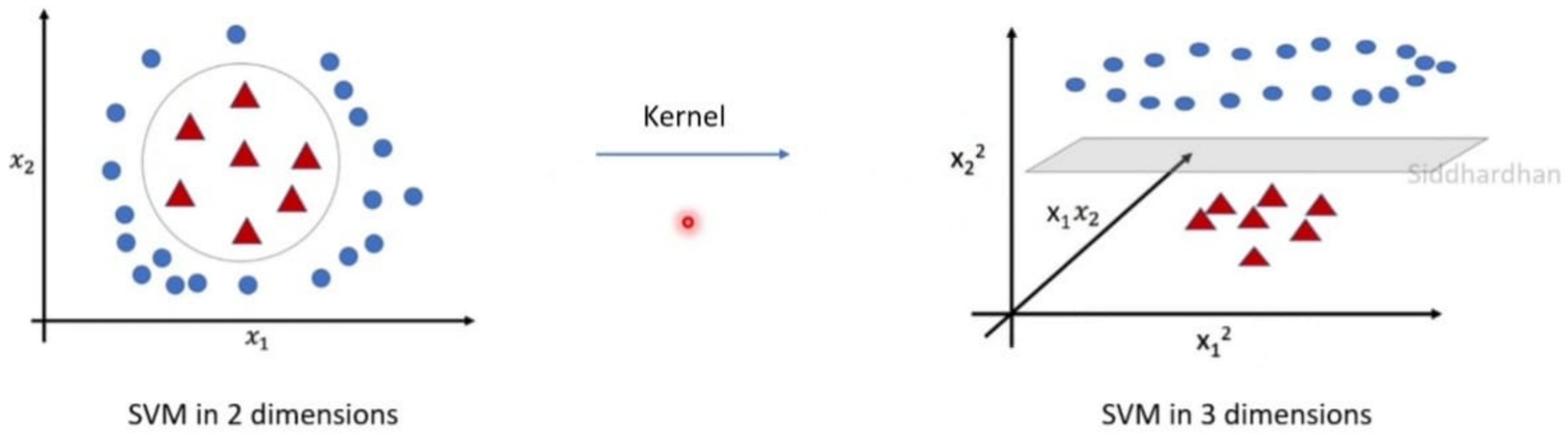
Feature (x)	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6



Feature (x)	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
x^2	36	25	16	9	4	1	0	1	4	9	16	25	36

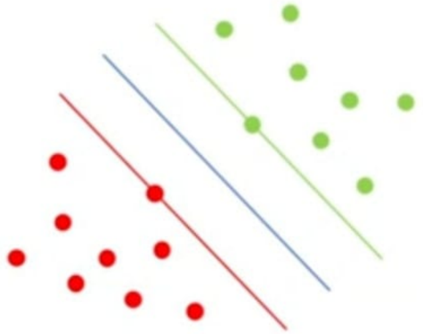


2)Polynomial Kernel

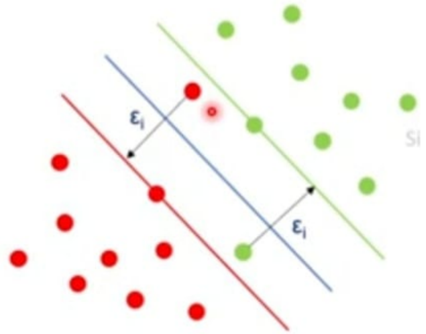


Soft Margin Classification:

In real-world scenarios, the data might not be perfectly separable. There could be outliers or noise that makes it difficult to find a hyperplane that separates the classes perfectly.
SVM introduces a concept of a soft margin that allows for some misclassifications. It balances between maximizing the margin and minimizing the misclassifications.
The algorithm finds the optimal hyperplane that minimizes the errors while maximizing the margin. The points that fall within the margin or are misclassified contribute to the error.



Hard Margin



Soft Margin

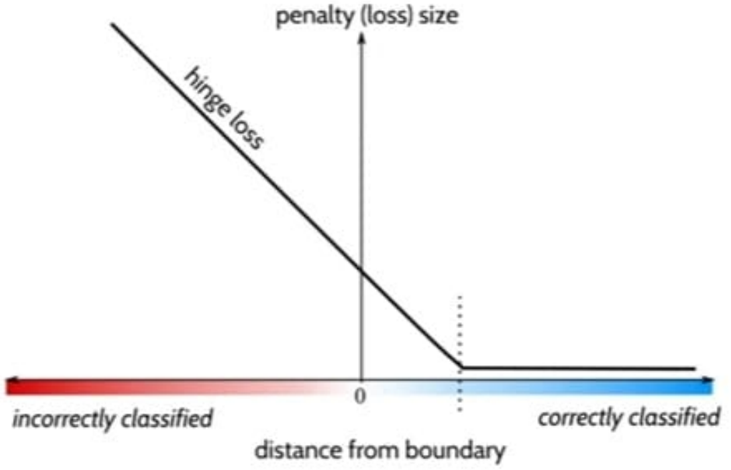
Hinge Loss

Hinge Loss is one of the types of Loss Function, mainly used for maximum margin classification models.

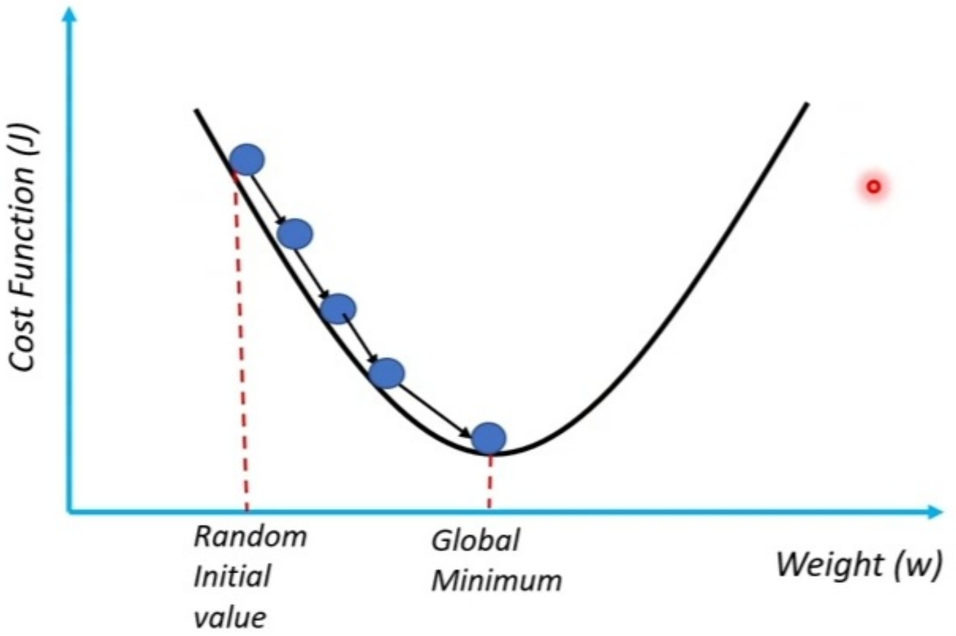
Hinge Loss incorporates a margin or distance from the classification boundary into the loss calculation. Even if new observations are classified correctly, they can incur a penalty if the margin from the decision boundary is not large enough.

$$L = \max(0, 1 - y_i (w^T x_i + b))$$

0 - for correct classification
1 - for wrong classification



Gradient Descent



Gradient Descent

Gradient Descent is an optimization algorithm used for minimizing the cost function in various machine learning algorithms. It is used for updating the parameters of the learning model.

$$w_2 = w_1 - L * \frac{dJ}{dw}$$

$$b_2 = b_1 - L * \frac{dJ}{db}$$

w --> weight
b --> bias
L --> Learning Rate

$\frac{dJ}{dw}$ --> Partial Derivative of cost function with respect to w

$\frac{dJ}{db}$ --> Partial Derivative of cost function with respect to b

Objective Function:

The objective function of soft margin SVM combines the hinge loss with a regularization term to control the complexity of the model

$$J(w, b) = C * \Sigma[\max(0, 1 - y_i * (w^T * x_i + b))] + 0.5 * ||w||^2$$

where C is the regularization parameter that balances the trade-off between margin maximization and hinge loss minimization. The ||w||^2 term is the L2 regularization term.

The gradient descent algorithm is used to iteratively update the model's parameters based on the gradients of the objective function with respect to w and b.

The gradients of the hinge loss function with respect to w and b are:

$$\partial L(y_i, f(x_i)) / \partial w = -y_i * x_i$$

$$\partial L(y_i, f(x_i)) / \partial b = -y_i$$

The gradients of the regularization term with respect to w and b are:

$$\partial(0.5 * ||w||^2) / \partial w = w$$

$$\partial(0.5 * ||w||^2) / \partial b = 0$$

The gradients of the objective function can be obtained by summing the gradients of the hinge loss and the regularization term over all samples:

$$\partial J(w, b) / \partial w = C * \Sigma[-y_i * x_i] + w$$

$$\partial J(w, b) / \partial b = C * \Sigma[-y_i]$$

Making Predictions:

Once the optimal hyperplane is determined, you can use it to make predictions on new, unseen data points.
Given a new data point, SVM classifies it based on which side of the hyperplane it falls on. If it's on one side, it belongs to one class, and if it's on the other side, it belongs to the other class.

Code:

Notebook Link