mLib

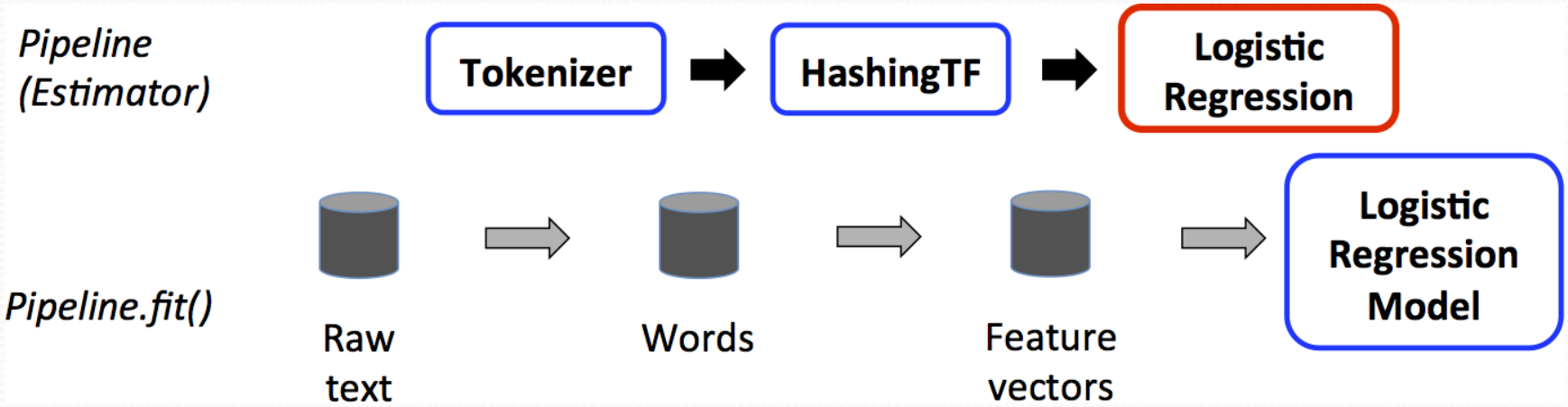
MLlib

- MLlib is a scalable machine learning library consisting of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, and underlying optimization primitives.

Mllib Pipeline

- MLlib's Pipeline helps manage the ML workflow. Components:
- Transformer: Converts one DataFrame to another (e.g., VectorAssembler).
- Estimator: Learns from data and produces a Transformer (e.g., LogisticRegression).
- Pipeline: Chains multiple stages.PipelineModel: A fitted pipeline.

- A Pipeline is specified as a sequence of stages, and each stage is either a Transformer or an Estimator.
- These stages are run in order, and the input DataFrame is transformed as it passes through each stage.
- For Transformer stages, the transform() method is called on the DataFrame.
- For Estimator stages, the fit() method is called to produce a Transformer (which becomes part of the PipelineModel, or fitted Pipeline), and that Transformer's transform() method is called on the DataFrame.

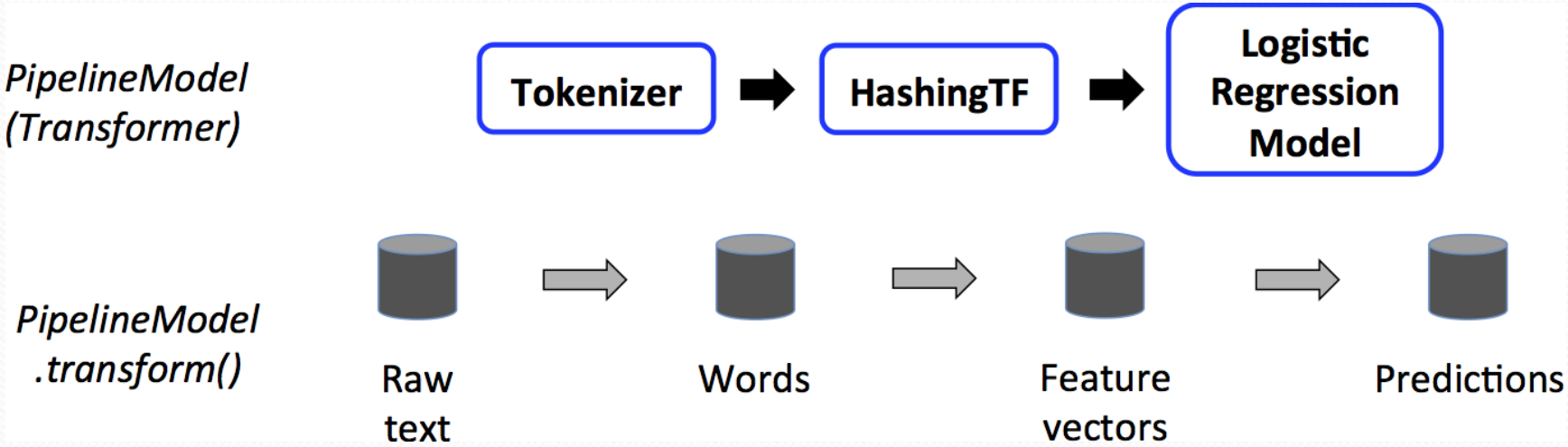


- Above, the top row represents a Pipeline with three stages.

The first two (Tokenizer and HashingTF) are Transformers (blue), and the third (LogisticRegression) is an Estimator (red).

- The bottom row represents data flowing through the pipeline, where cylinders indicate DataFrames.

- The Pipeline.fit() method is called on the original DataFrame, which has raw text documents and labels.
- The Tokenizer.transform() method splits the raw text documents into words, adding a new column with words to the DataFrame.
- The HashingTF.transform() method converts the words column into feature vectors, adding a new column with those vectors to the DataFrame.
- Now, since LogisticRegression is an Estimator, the Pipeline first calls LogisticRegression.fit() to produce a LogisticRegressionModel. If the Pipeline had more Estimators, it would call the LogisticRegressionModel's transform() method on the DataFrame before passing the DataFrame to the next stage.



A Pipeline is an Estimator. Thus, after a Pipeline's fit() method runs, it produces a PipelineModel, which is a Transformer. This PipelineModel is used at test time; above figure illustrates this usage.

- the PipelineModel has the same number of stages as the original Pipeline,
- but all Estimators in the original Pipeline have become Transformers.
- When the PipelineModel's transform() method is called on a test dataset, the data are passed through the fitted pipeline in order.
- Each stage's transform() method updates the dataset and passes it to the next stage.
- Pipelines and PipelineModels help to ensure that training and test data go through identical feature processing steps.

```
import org.apache.spark.ml.feature.{RegexTokenizer, HashingTF}
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.ml.linalg.Vector
import org.apache.spark.sql.{SparkSession, Row}
import org.apache.spark.ml.param.ParamMap
// Create Spark session
val spark = SparkSession.builder.appName("Transformer and Estimator Example").getOrCreate()
// Sample dataset with text data
val documentDF = spark.createDataFrame(Seq(
  (0, "Hello world"),
  (1, "Spark is powerful"),
  (2, "Machine learning is great")
)).toDF("id", "text")
```

```
// Step 1: Tokenizer - This is a **Transformer**
val tokenizer = new RegexTokenizer()
  .setInputCol("text")
  .setOutputCol("words")
  .setPattern("\\W+")  // Tokenizes by non-word characters
val wordsData = tokenizer.transform(documentDF)
// Step 2: HashingTF - This is also a **Transformer**
val hashingTF = new HashingTF()
  .setInputCol("words")
  .setOutputCol("features")
  .setNumFeatures(10)  // Number of features (vector size)
val featurizedData = hashingTF.transform(wordsData)
```

```
// Step 3: Logistic Regression - This is an **Estimator**
val lr = new LogisticRegression()
  .setMaxIter(10)
  .setRegParam(0.01)

val model = lr.fit(featurizedData)  // Estimator that learns from featurized data

// Print out model parameters
println(s"LogisticRegression parameters:\n ${lr.explainParams()}\n")
println(s"Model was fit using parameters: ${model.parent.extractParamMap()}")

// Step 4: Make Predictions using the Logistic Regression Model - Transformer
val predictions = model.transform(featurizedData)
// Show the results: predictions, features, and probabilities
predictions.select("id", "text", "features", "probability", "prediction").show()
```

Binary Classification

- Binary classification aims to divide items into two categories: positive and negative. MLlib supports two linear methods for binary classification:
  - linear support vector machines (SVMs) and
  - logistic regression.
- For both methods, MLlib supports L1 and L2 regularized variants.
- The training data set is represented by an RDD of LabeledPoint in MLlib.
- Note that, in the mathematical formulation in this guide, a training label y is denoted as either +1 (positive) or −1 (negative), which is convenient for the formulation.
- However, the negative label is represented by 0 in MLlib instead of −1, to be consistent with multi-class labeling.
- .

The simplest method to solve optimization problems of the form $\min_{w \in \mathbb{R}} df(w)$ is gradient descent. This methodology is very well-suited for large-scale and distributed computation. Spark MLlib uses stochastic gradient descent (SGD) to solve these optimization problems, which are the core of supervised machine learning, for optimizations and much higher performance

Precision is the fraction of retrieved documents that are relevant to the find:

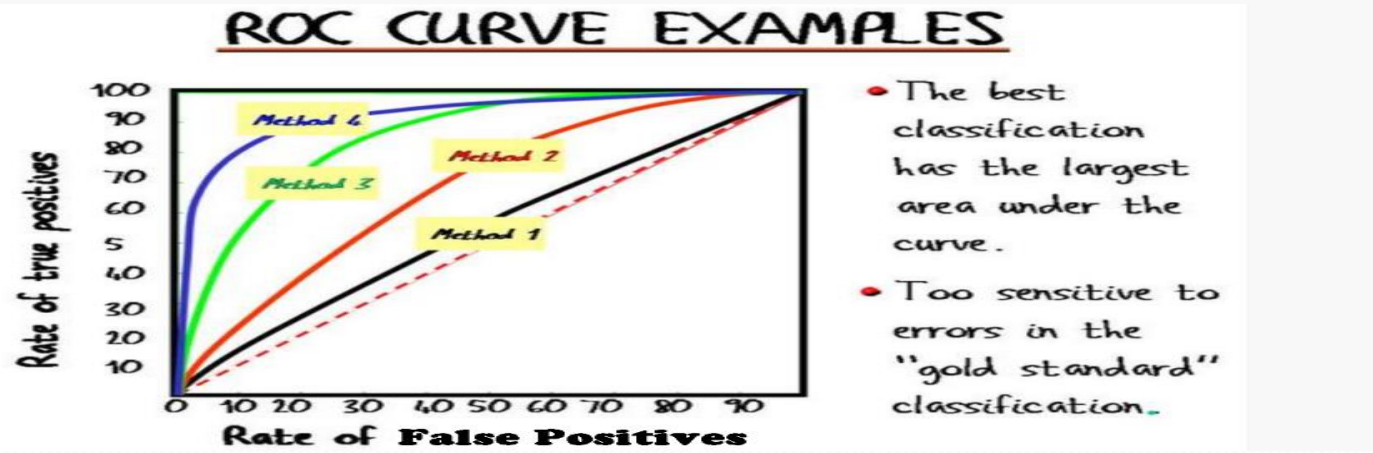
$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

Recall is the fraction of the documents that are relevant to the query that are successfully retrieved:

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

- Receiver operating characteristic (ROC), or ROC curve, is a graphical plot that illustrates the performance of a binary classifier system and created by plotting the true positive rate against the false positive rate.



- A classifier with the Red dashed line is guessing the label randomly.
- Closer the ROC curve gets to top-left part of the chart, better the classifier is.
- Area under the curves (AUC) is the area below these ROC curves. Therefore, in other words, AUC is a great indicator of how well a classifier functions.
- AUC is commonly used to compare the performance of various models while precision/recall/F-measure can help determine the appropriate threshold to use for prediction purposes.

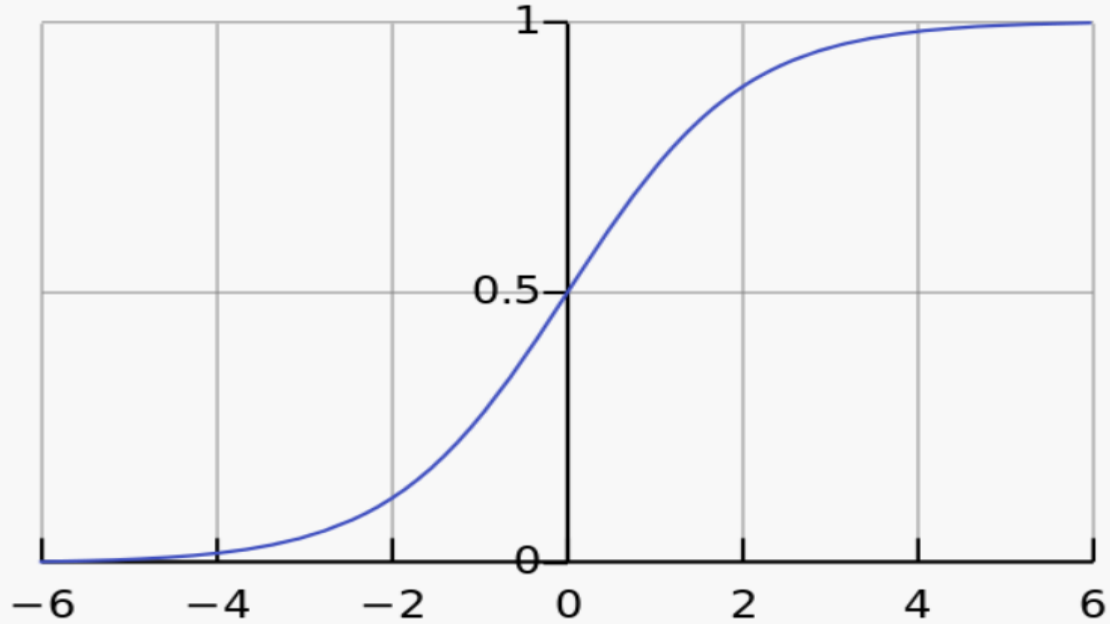
Logistic Regression

- Logistic Regression is a type of probabilistic statistical classification model.
- Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables, which are usually (but not necessarily) continuous, by using probability scores as the predicted values of the dependent variable.

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}},$$

$$t = \beta_0 + \beta_1 x$$

$$F(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

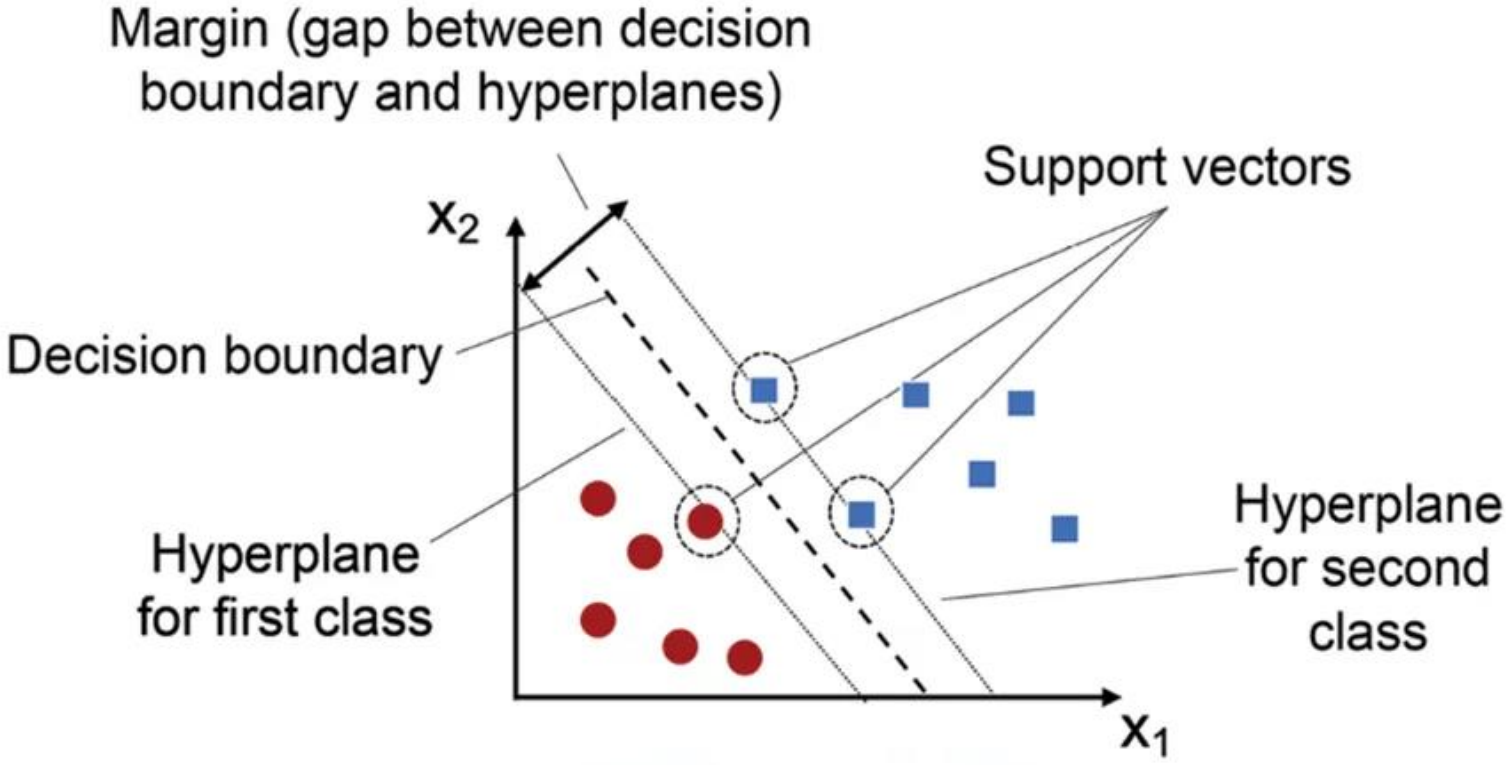


Linear Support Vector Machines (SVMs)

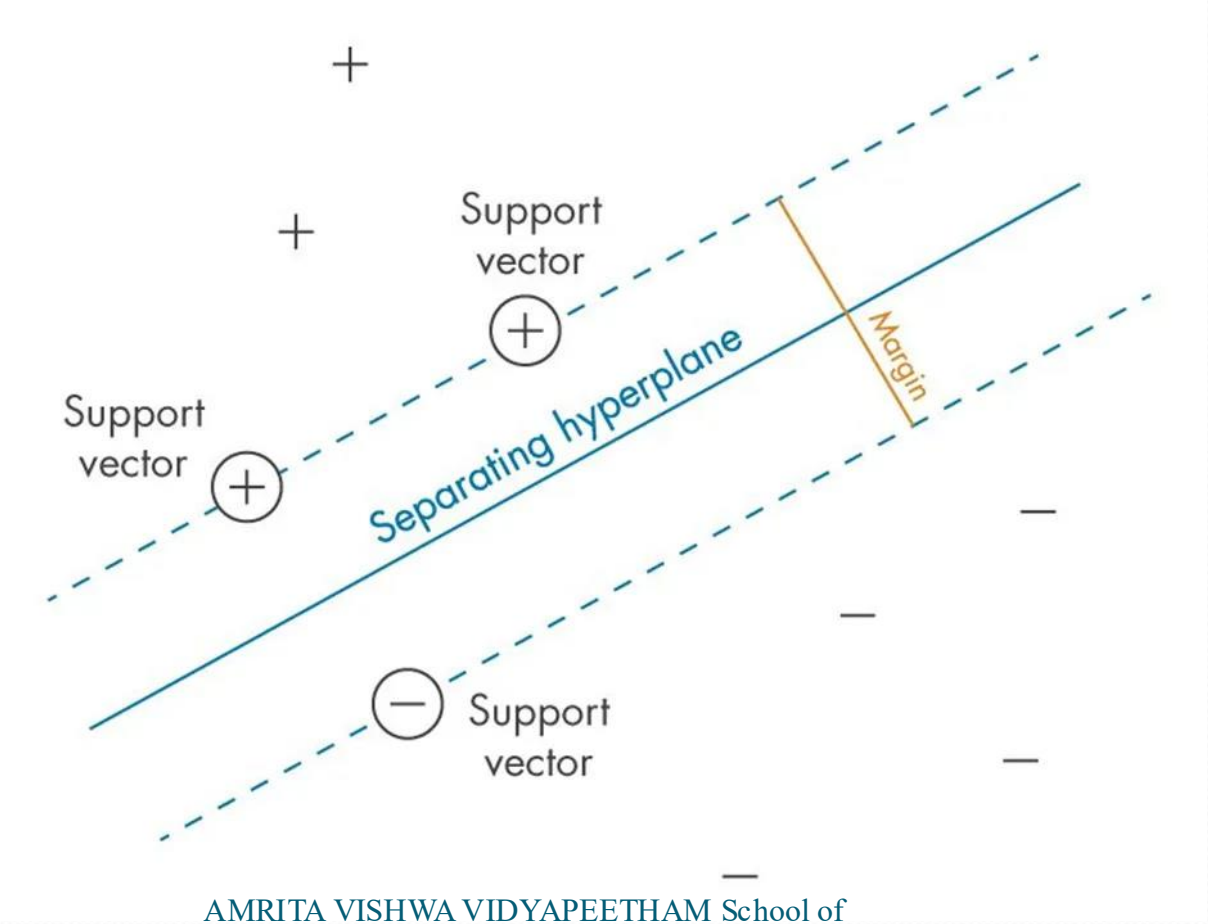
- In machine learning, support vector machines (SVMs) are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis.
- Given a set of training examples, each belonging to one of two class labels, an SVM algorithm builds a model that assigns new examples into one label or another.
- Unlike Logistic Regression, SVM is a non-probabilistic binary linear classifier.
- An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.
- New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

Key Terms

- **Hyperplane**
- A hyperplane is a decision boundary that separates data points into different classes in a high-dimensional space.
- In two-dimensional space, a hyperplane is simply a line that separates the data points into two classes. In three-dimensional space, a hyperplane is a plane that separates the data points into two classes. Similarly, in N-dimensional space, a hyperplane has (N-1)-dimensions.
- It can be used to make predictions on new data points by evaluating which side of the hyperplane they fall on. Data points on one side of the hyperplane are classified as belonging to one class, while data points on the other side of the hyperplane are classified as belonging to another class.
-


Margin (gap between decision boundary and hyperplanes)
Support vectors
x_2
Decision boundary
Hyperplane for first class
Hyperplane for second class
x_1

- **Margin**
- A margin is the distance between the decision boundary (hyperplane) and the closest data points from each class.
- The goal of SVMs is to maximize this margin while minimizing classification errors.
- A larger margin indicates a greater degree of confidence in the classification, as it means that there is a larger gap between the decision boundary and the closest data points from each class.
- The margin is a measure of how well-separated the classes are in feature space.
- SVMs are designed to find the hyperplane that maximizes this margin, which is why they are sometimes referred to as maximum-margin classifiers.


Support
vector
Support
vector
Separating hyperplane
Margin
Support
vector

- Support Vectors
- They are the data points that lie closest to the decision boundary (hyperplane) in a Support Vector Machine (SVM).
- These data points are important because they determine the position and orientation of the hyperplane, and thus have a significant impact on the classification accuracy of the SVM.
- In fact, SVMs are named after these support vectors because they “support” or define the decision boundary.
- The support vectors are used to calculate the margin, which is the distance between the hyperplane and the closest data points from each class.
- The goal of SVMs is to maximize this margin while minimizing classification errors.

Kernal Trick

- The kernel trick is a method used in SVMs to enable them to classify non-linear data using a linear classifier.
- By applying a kernel function, SVMs can implicitly map input data into a higher-dimensional space where a linear separator (hyperplane) can be used to divide the classes.
- This mapping is computationally efficient because it avoids the direct calculation of the coordinates in this higher space.
- The kernel trick relies on the inner products of vectors.

- For SVMs, the decision function is based on the dot products of vectors within the input space.
- Kernel functions replace these dot products with a non-linear function that computes a dot product in a higher-dimensional space.
- Importantly, the computation of this dot product via the kernel function does not require explicit knowledge of the coordinates in the higher space, thus saving computational resources and time.