Classical Algorithms

Marek Jeliński
Gradient Science Club 2022

Plan for Today

- Linear Regression
  - Loss/Cost Function
  - Gradient Descent
- Logistic Regression
- Decision Trees
- K-Nearest Neighbors
- SVMs (Support Vector Machines)

Linear regression

Linear regression

Linear regression learns to model a dependent variable y, as a function of some independent variables (aka "features"), x_i, by finding a line (or surface) that best "fits" the data.

Equation for linear regression

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \varepsilon$$

y: the dependent variable; the thing we are trying to predict.

x_i : the independent variables: the features our model uses to model y

β_i : the coefficients (aka "weights") of our regression model. These are the foundations of our model. They are what our model "learns" during optimization.

ε: the irreducible error in our model. A term that collects together all the unmodeled parts of our data.

Predicting future values

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \ldots + \hat{\beta}_n x_n$$

$\hat{y}$: the future value

$\hat{\beta}_i$: the estimated coefficients

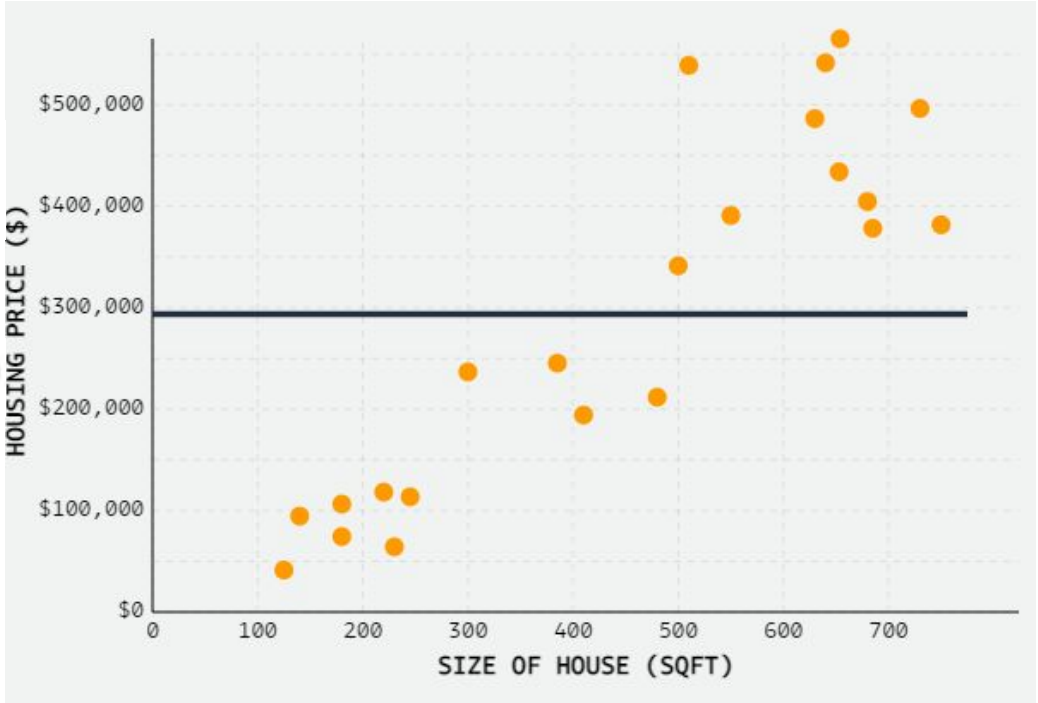
Example

Predict housing price ($) using the size of the house (in square-footage):

$$housePrice = \hat{\beta}_1 * sqft + \hat{\beta}_0$$

Predicting the price of each house to be just the average house price in our dataset ~$290,000

$$housePrice = 0 * sqft + 290000$$



Errors

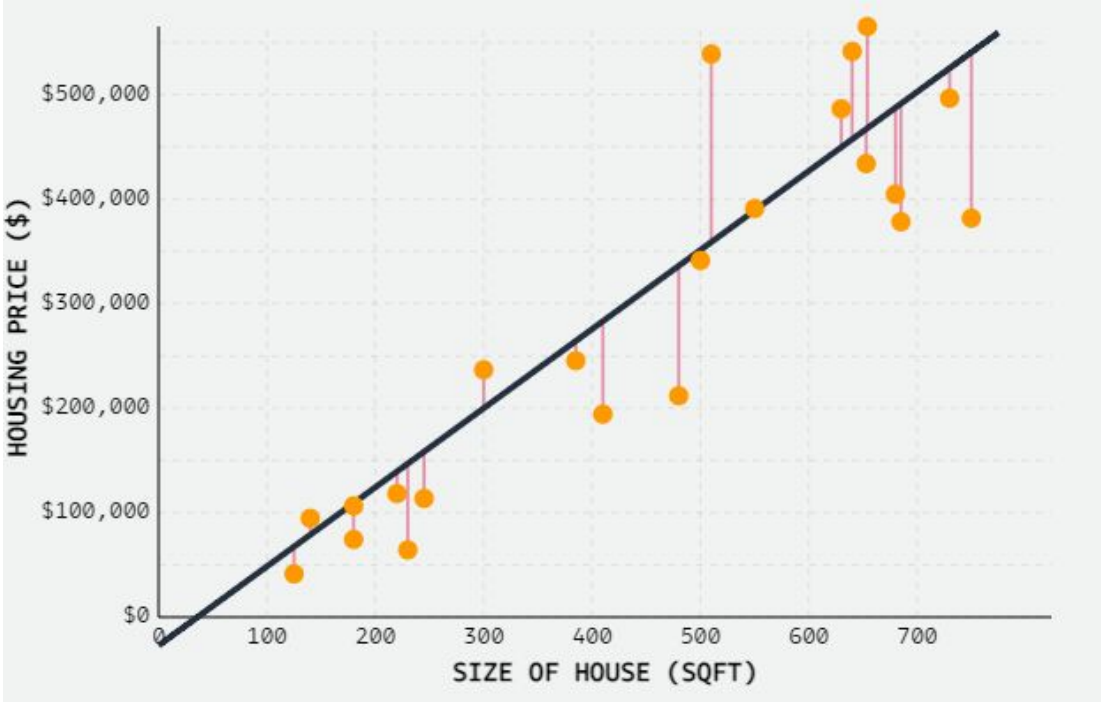
The model doesn't fit the data well at all

Plot the error of each observation directly.

These errors, or residuals, measure the distance between each observation and the predicted value for that observation.

Final goal

The goal of linear regression is reducing this error such that we find a line/surface that 'best' fits our data.



Predicting

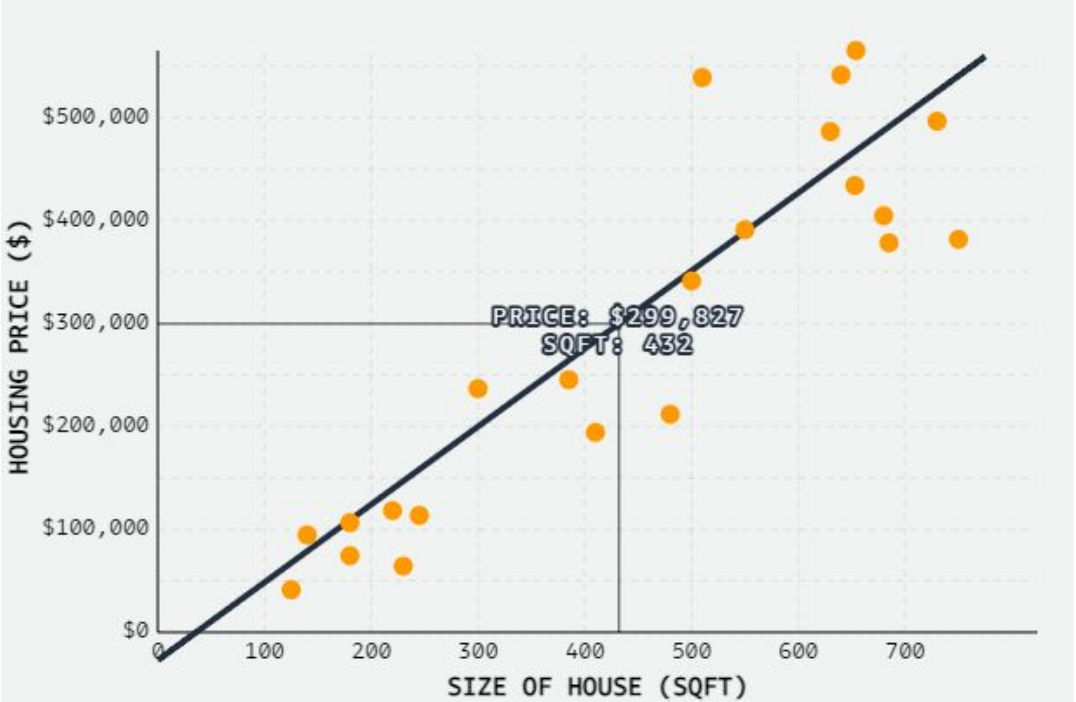
To predict future we just plug in any x_i values into our equation!

sqft Value: 432

$$\hat{y} = 756.9 * 432 - 27153.8$$

$$\hat{y} = 299827$$

Model predicts a house that is 432 square-feet will cost $299,827



Loss functions

To train an accurate linear regression model, we need a way to quantify how good (or bad) our model performs.

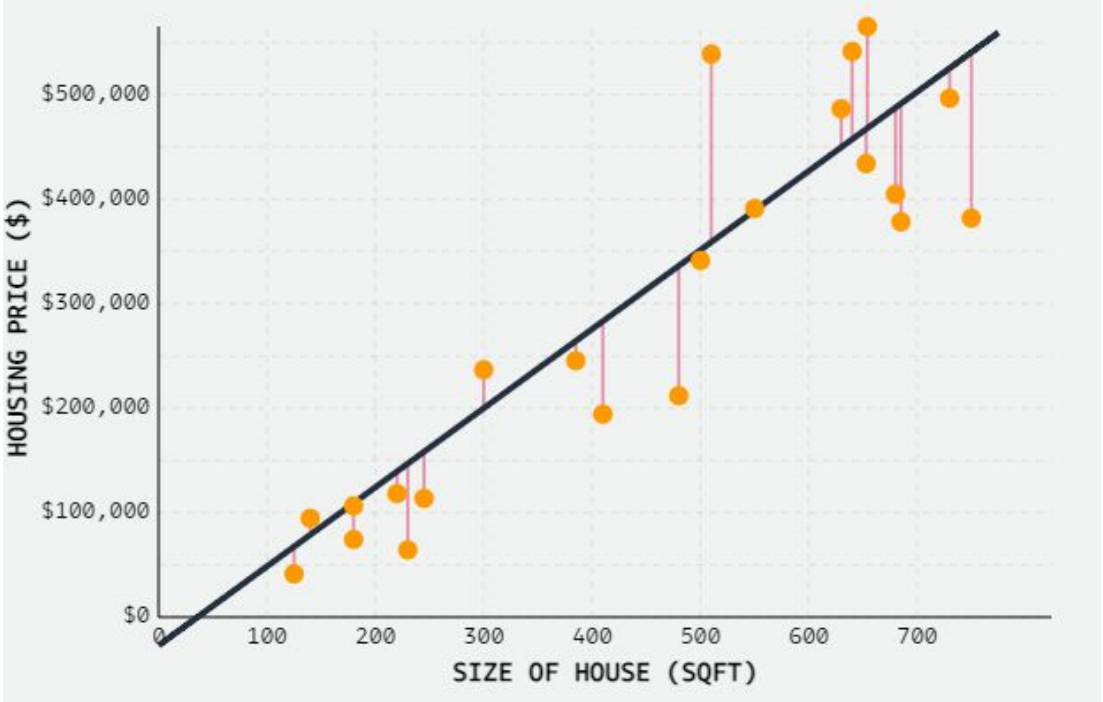
In machine learning, we call such performance-measuring functions loss functions.

Mean-Squared Error (MSE)

MSE quantifies how close a predicted value is to the true value, so we'll use it to quantify how close a regression line is to a set of points.

$$MSE = \frac{1}{n}\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2$$

MSE works by squaring the distance between each data point and the regression line (the red residuals in the graphs), summing the squared values, and then dividing by the number of data points



Learning The Coefficients

Linear regression is all about finding a line (or surface) that fits our data well.

And as we just saw, this involves selecting the coefficients for our model that minimize our evaluation metric.

But how can we best estimate these coefficients?

Iterative Solution

Gradient descent is an iterative optimization algorithm that estimates some set of coefficients to yield the minimum of a convex function.

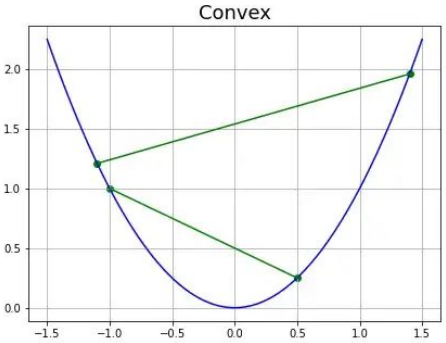
Put simply: it will find suitable coefficients for our regression model that minimize prediction error (remember, lower MSE equals better model).

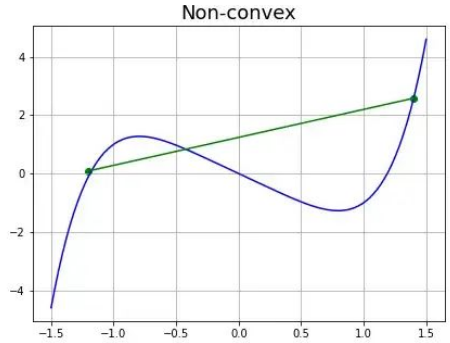
Gradient descent quick look

We assume that we have some convex function representing the error of our machine learning algorithm (in our case, MSE).

Gradient descent will iteratively update our model's coefficients in the direction of our error functions minimum

Gradient descent won't always yield the best coefficients for our model, because it can sometimes get stuck in local minima





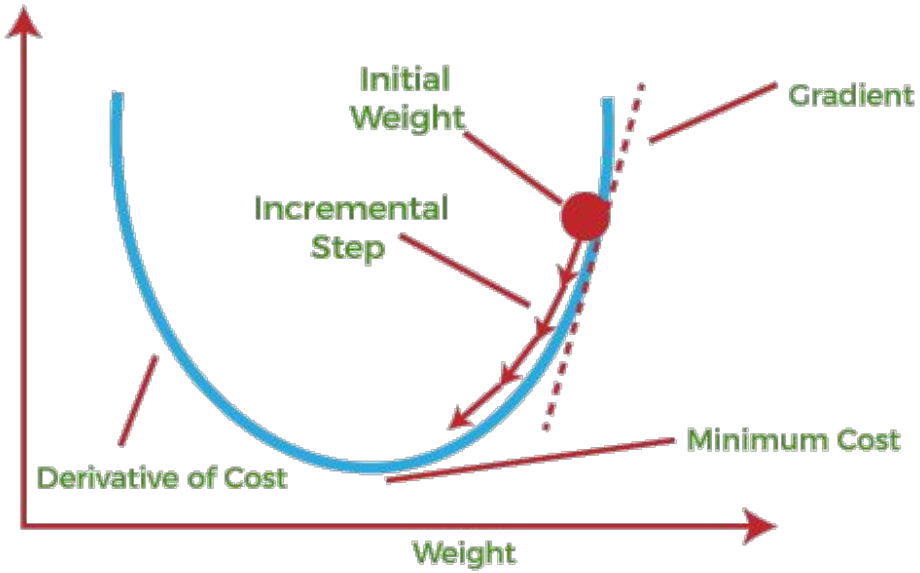
How gradient descent works?

Our goal is to find the coefficients, β_0 and β_1, to minimize the error function.

To do this, we'll use the gradient, which represents the direction that the function is increasing, and the rate at which it is increasing.

Since we want to find the minimum of this function, we can go in the opposite direction of where it's increasing.

This is exactly what Gradient Descent does, it works by taking steps in the direction opposite of where our error function is increasing, proportional to the rate of change.



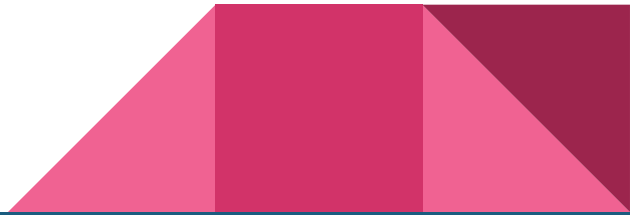
Gradient descent in our case

In our case, our model takes the form:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1$$

and our error function takes the form:

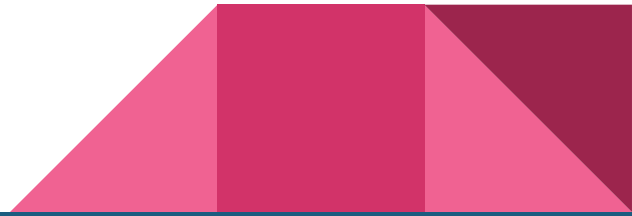
$$MSE(\hat{\beta}_0, \hat{\beta}_1) = \frac{1}{n}\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\Sigma_{i=1}^{n}(y_i - \hat{\beta}_0 + \hat{\beta}_1 x_1)^2$$



Finding coefficients

To find the coefficients that minimize the function, we first calculate the derivatives of our error (loss) function MSE:

$$\frac{\delta}{\delta \beta_i} MSE = \begin{cases} -\frac{2}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i) \ for \ i = 0 \\ -\frac{2}{n} \sum_{i=1}^{n} x_i (y_i - \hat{y}_i) \ for \ i = 1 \end{cases}$$

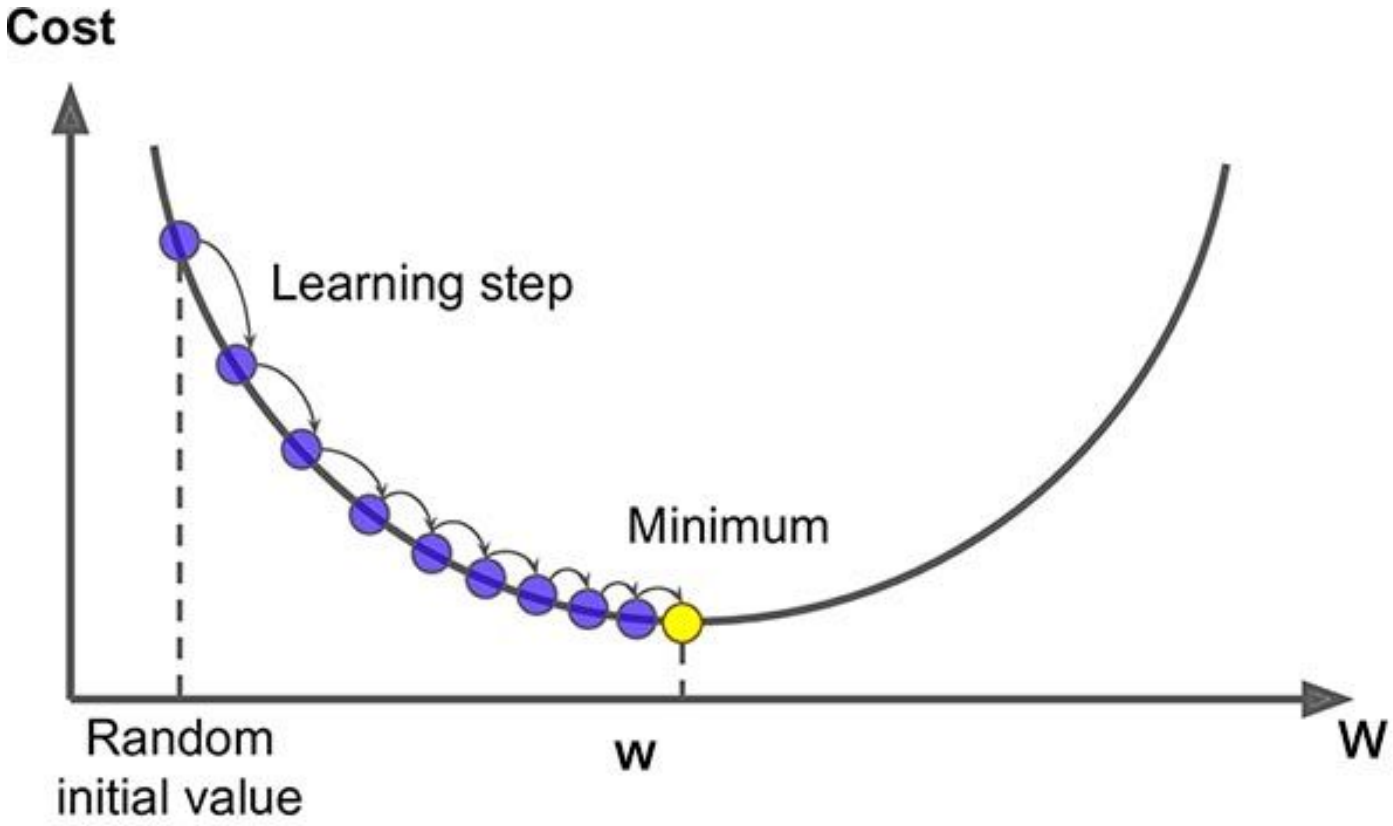


Updating coefficients

Now that we have the gradients for our error function (with respect to each coefficient to be updated), we perform iterative updates:

$$repeat\ until\ converge: = \begin{cases} \beta_0 = \beta_0 - \alpha\left(-\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)\right) \\ \beta_1 = \beta_1 - \alpha\left(-\frac{2}{n}\sum_{i=1}^{n}x_i(y_i - \hat{y}_i)\right) \end{cases}$$

Updating these values iteratively will yield coefficients of our model that minimize error.

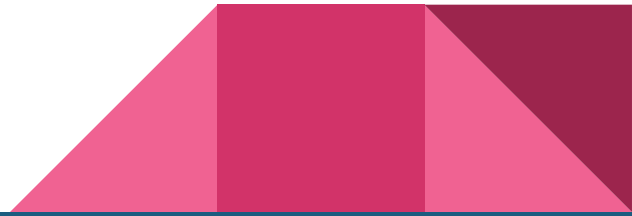

Cost
Learning step
Minimum
Random
initial value
w
W

Logistic regression

Logistic regression

Logistic regression is often used for binary classification, i.e. determining which of two groups a data point belongs to, or whether an event will occur or not.

The typical setup for logistic regression is as follows: there is an outcome y that falls into one of two categories (say 0 or 1), and the following equation is used to estimate the probability that y belongs to a particular category given inputs $X = (x_1, x_2, ..., x_n)$



Equation for logistic regression

$$P(y = 1|X) = sigmoid(z) = \frac{1}{1+e^{-z}}$$

$$z = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \ldots + \hat{\beta}_n x_n$$

z is called a linear predictor, and it is transformed by the sigmoid function so that the values fall between 0 and 1, and can therefore be interpreted as probabilities.

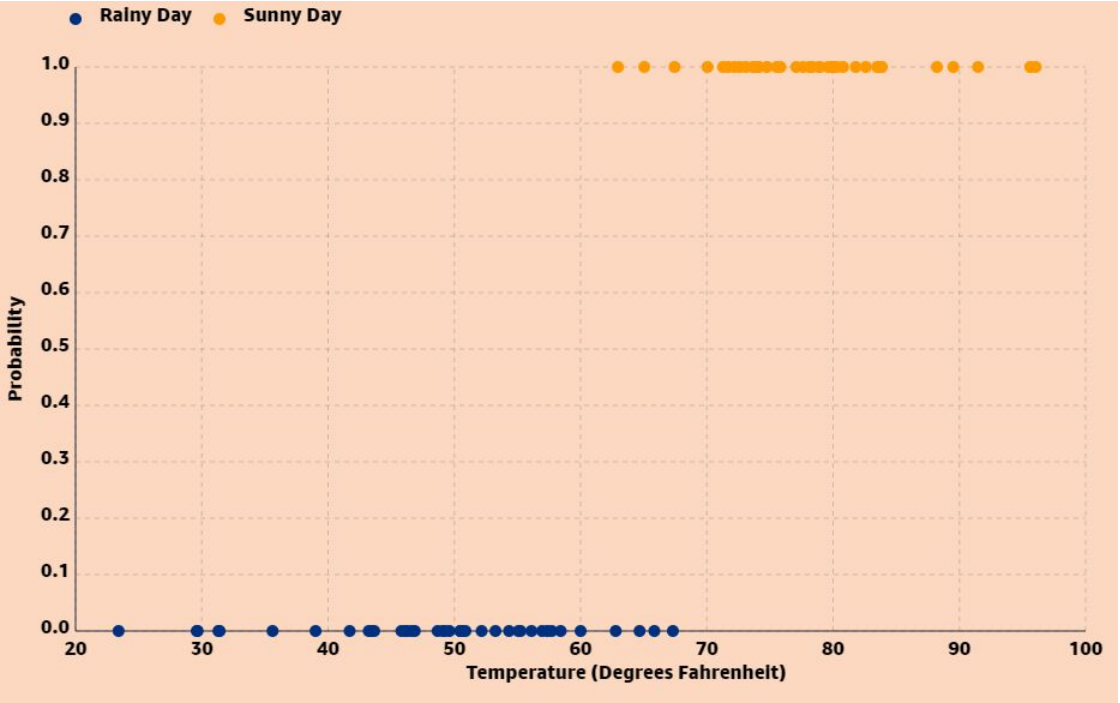
This resulting probability is then compared to a threshold to predict a class for y based on X.

Example

Can we predict the weather, given factors such as the temperature?

Assume there are two classes: Rainy Days and Sunny Days. We can assign a numeric value of 0 and 1 to each class, say 0 to a Rainy Day and 1 to a Sunny Day.

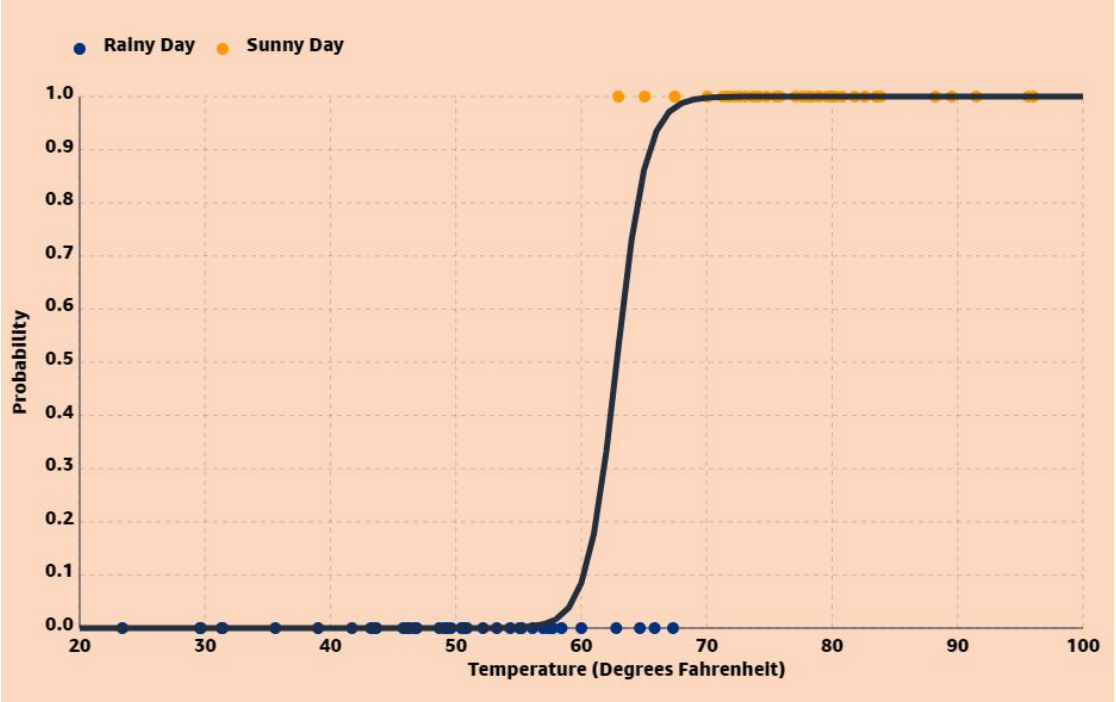
20°F ~ -7°C
60°F ~ 16°C
100°F ~ 38°C



Probability

The values of this function can be interpreted as probabilities, as the values range between 0 and 1.

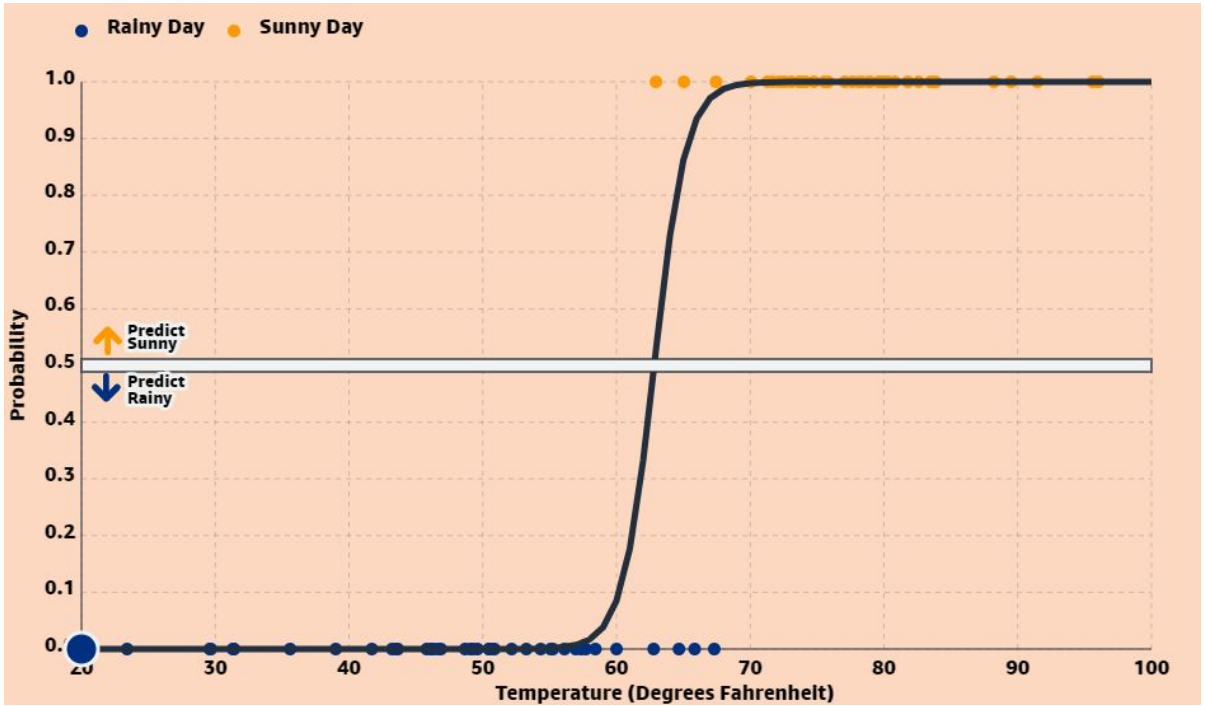
We can interpret the line as the probability of a sunny day given a particular temperature.



Threshold

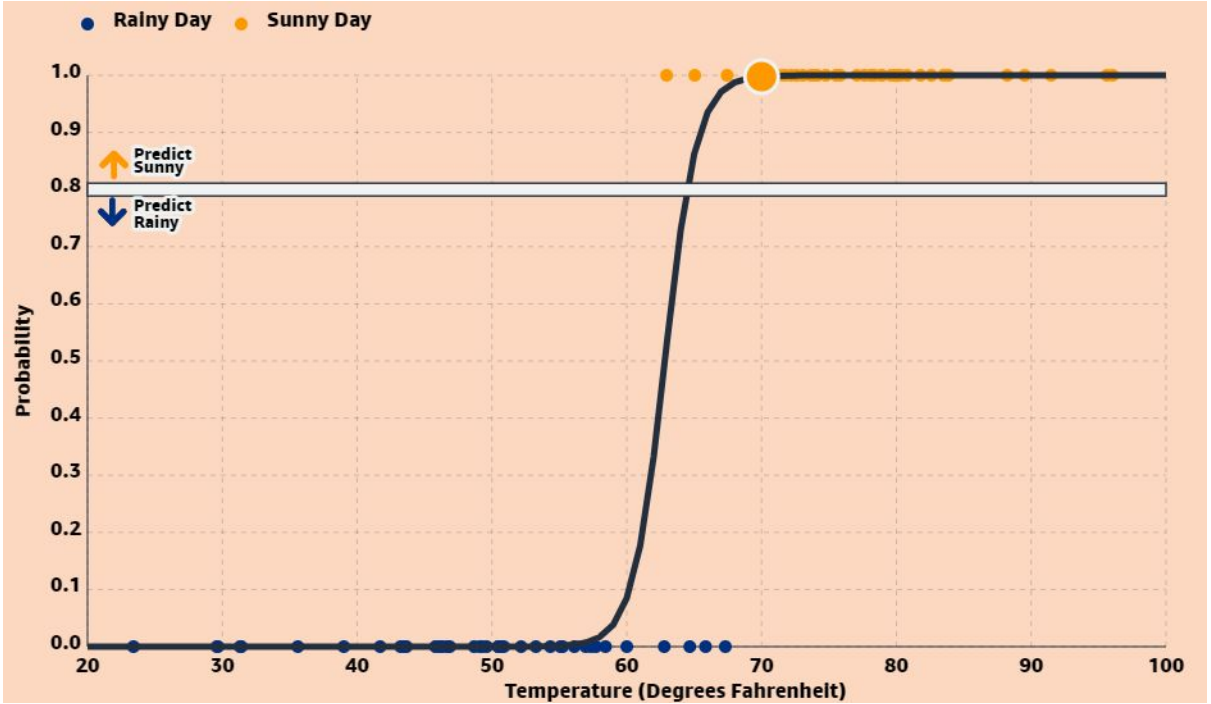
We use a classification threshold, or decision boundary, to decide the predicted class based on the probability of each class given the feature values.

A typical threshold is 0.5



Adjusted threshold

This threshold can be adjusted — for example, if you really dislike the rain, you may want to set the threshold higher to be more cautious



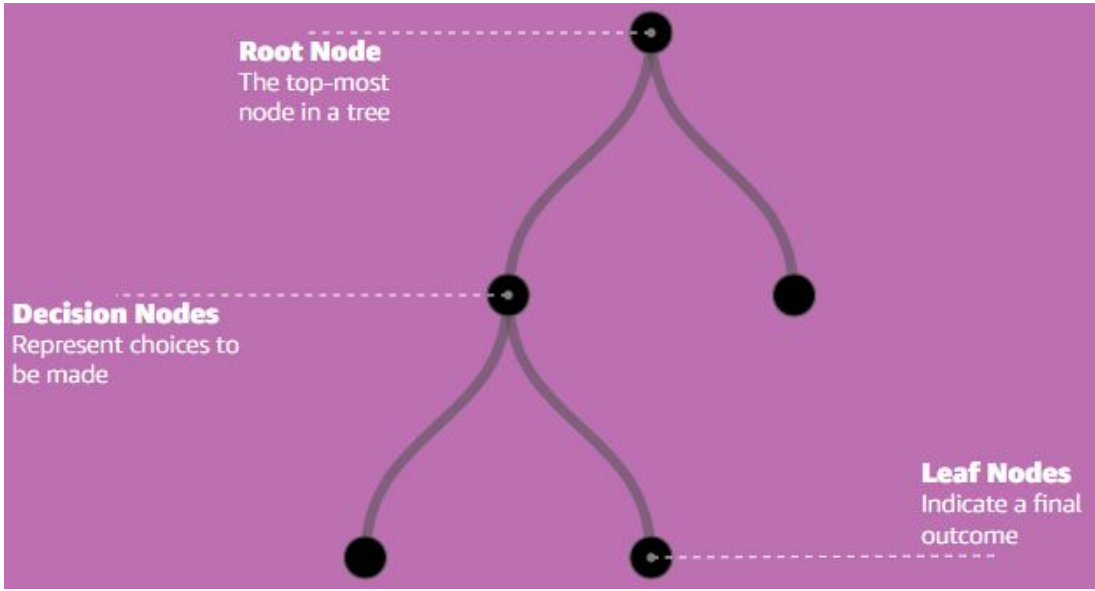
Decision trees

Decision Tree

A Decision Tree consists of a series of sequential decisions, or decision nodes, on some data set's features.

The resulting flow-like structure is navigated via conditional control statements, or if-then rules, which split each decision node into two or more subnodes.

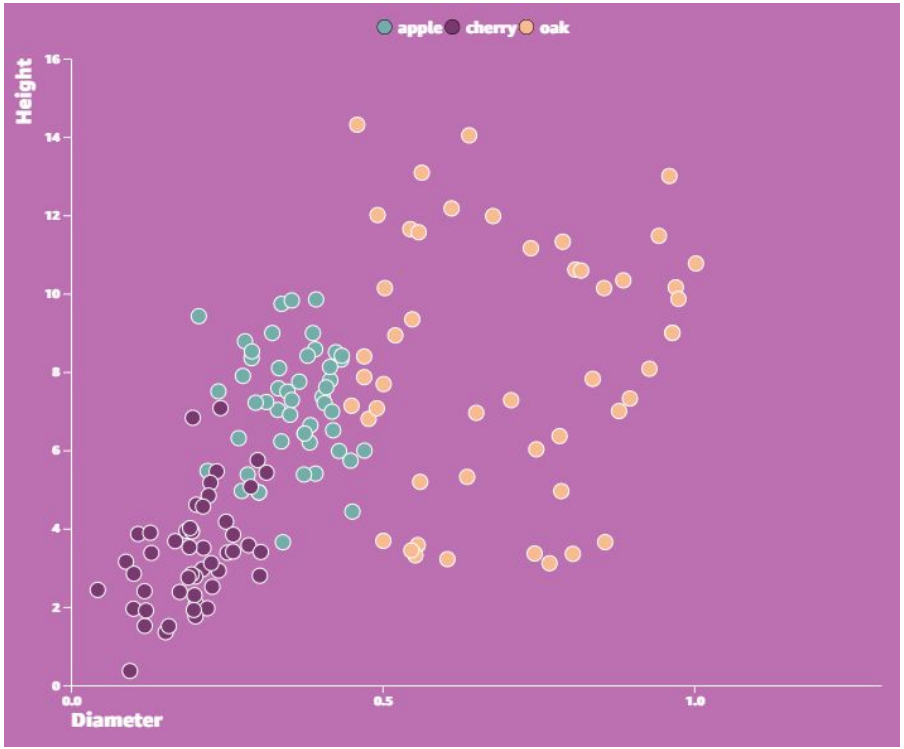
Leaf nodes, also known as terminal nodes, represent prediction outputs for the model.



Example

Let's pretend we're farmers with a new plot of land.

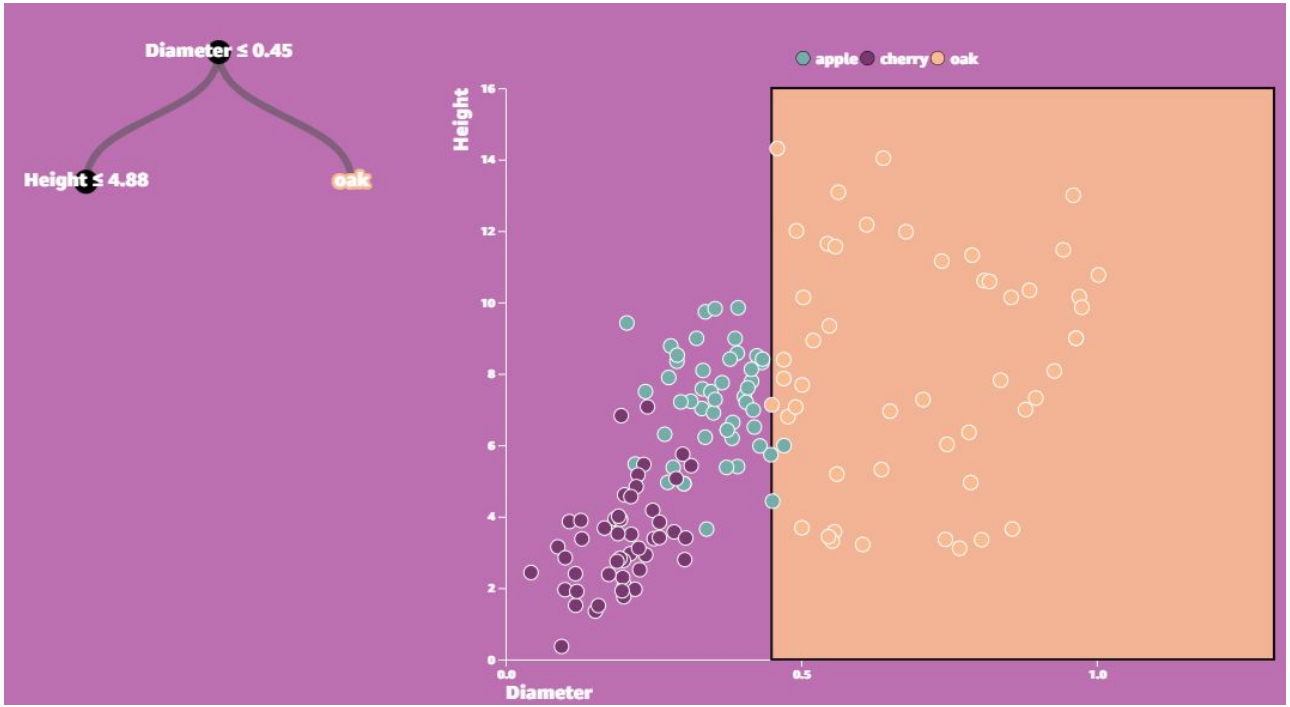
Given only the Diameter and Height of a tree trunk, we must determine if it's an Apple, Cherry, or Oak tree.



Start splitting

Almost every tree with a Diameter ≥ 0.45 is an Oak tree! Thus, we can probably assume that any other trees we find in that region will also be one.

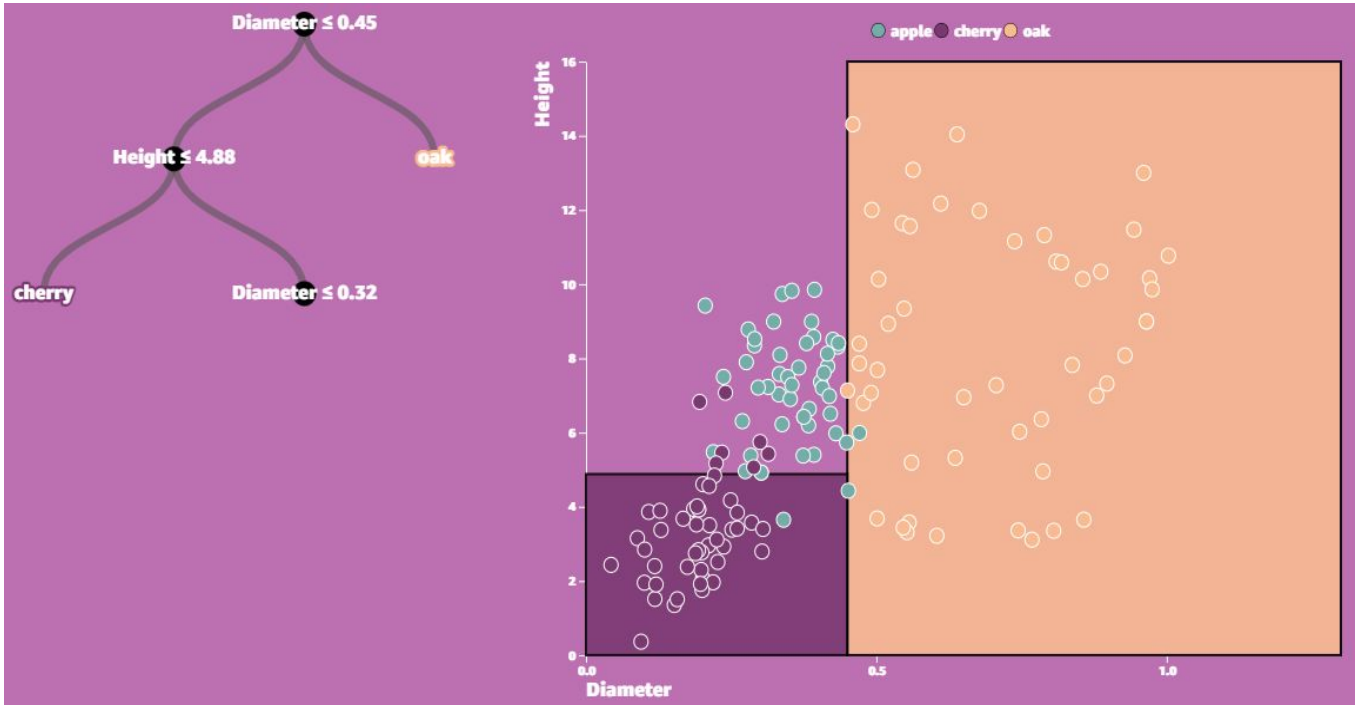
This first decision node will act as our root node. We'll draw a vertical line at this Diameter and classify everything above it as Oak (our first leaf node), and continue to partition our remaining data on the left.



Split Some More

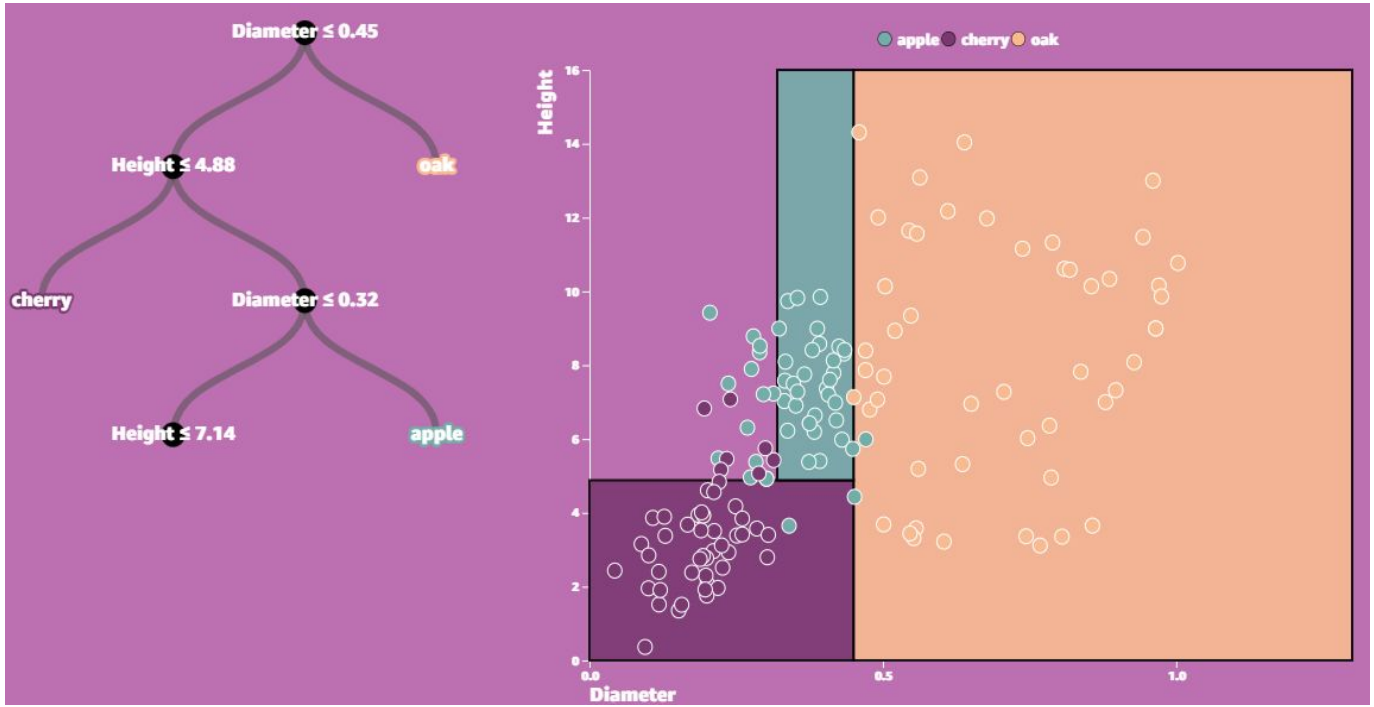
We see that creating a new decision node at Height ≤ 4.88 leads to a nice section of Cherry trees, so we partition our data there.

Our Decision Tree updates accordingly, adding a new leaf node for Cherry.



And Some More

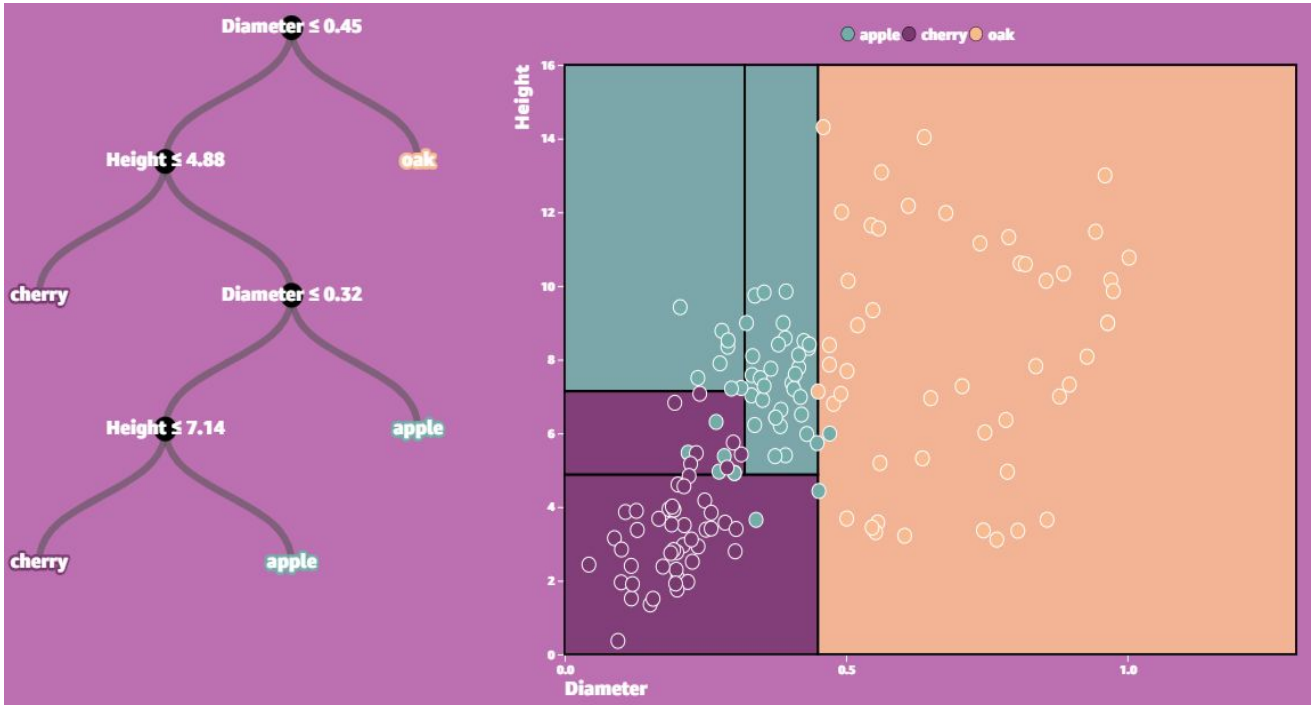
After this second split we're left with an area containing many Apple and some Cherry trees. No problem: a vertical division can be drawn to separate the Apple trees a bit better.



And Yet Some More

The remaining region just needs a further horizontal division and boom – our job is done! We've obtained an optimal set of nested decisions.

That said, some regions still enclose a few misclassified points. Should we continue splitting, partitioning into smaller sections?



Don't Go Too Deep!

If we do, the resulting regions would start becoming increasingly complex, and our tree would become unreasonably deep.

Such a Decision Tree would learn too much from the noise of the training examples and not enough generalizable rules.

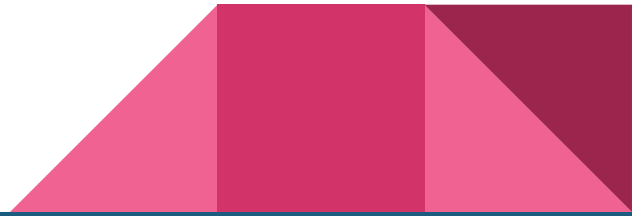
K-Nearest Neighbor

K-Nearest Neighbor

KNN tries to predict the correct class for the test data by calculating the distance between the test data and all the training points.

Then select the K number of points which is closest to the test data.

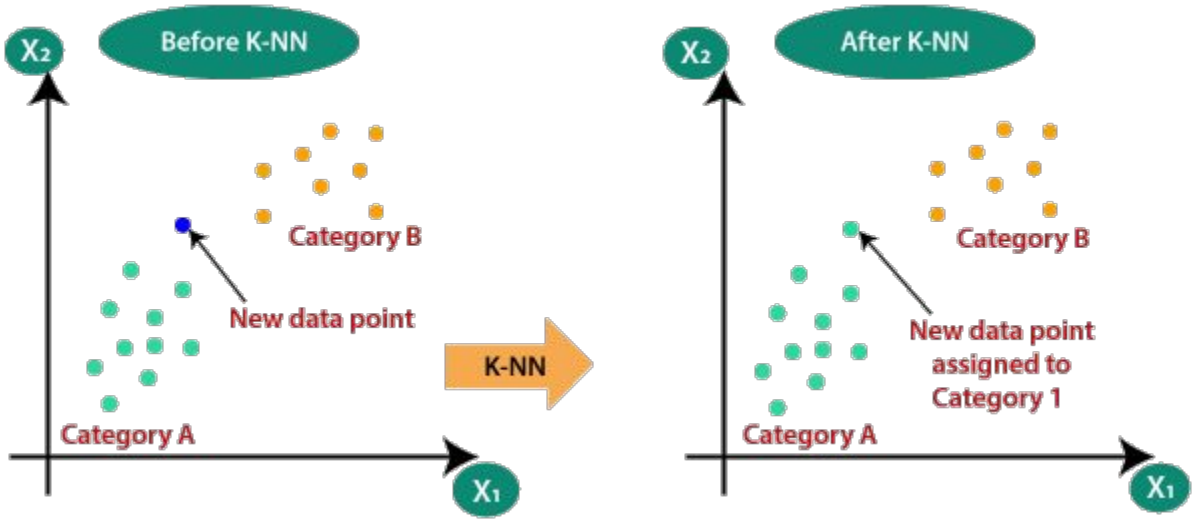
The KNN algorithm calculates the probability of the test data belonging to the classes of ‘K’ training data and class holds the highest probability will be selected.



Quick Overview

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in one of these categories.

With the help of K-NN, we can easily identify the category or class of a particular dataset.



Example

Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog.

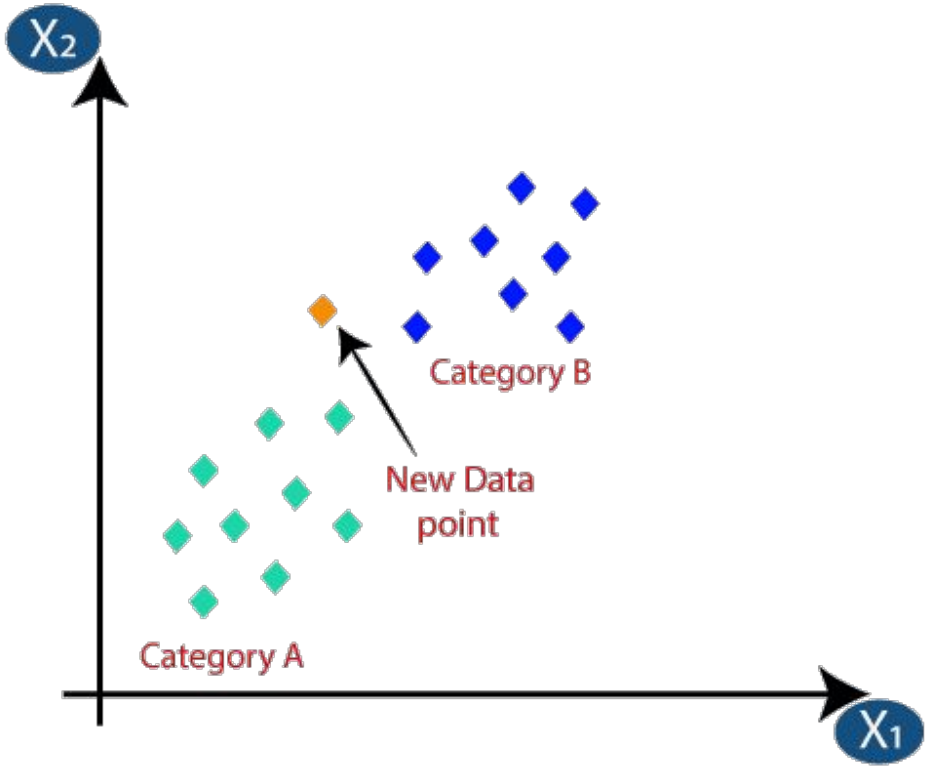
Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

New Data Point

Suppose we have a new data point and we need to put it in the required category.

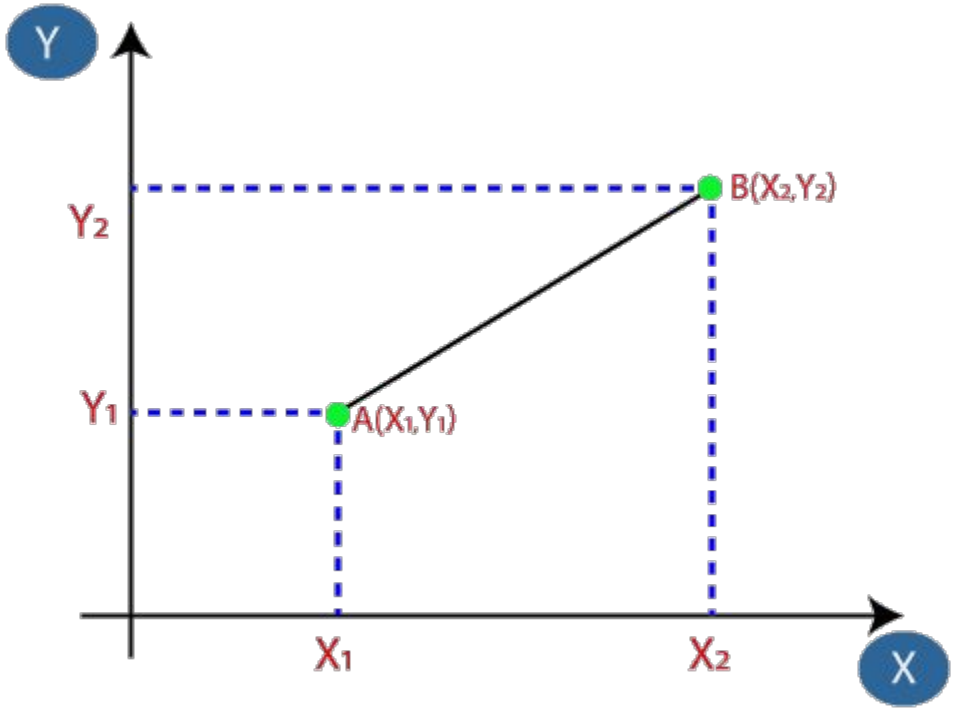
Firstly, we will choose the number of neighbors, so we will choose the k=5

Next, we will calculate the Euclidean distance between the data points



Euclidean distance

The Euclidean distance is the distance between two points

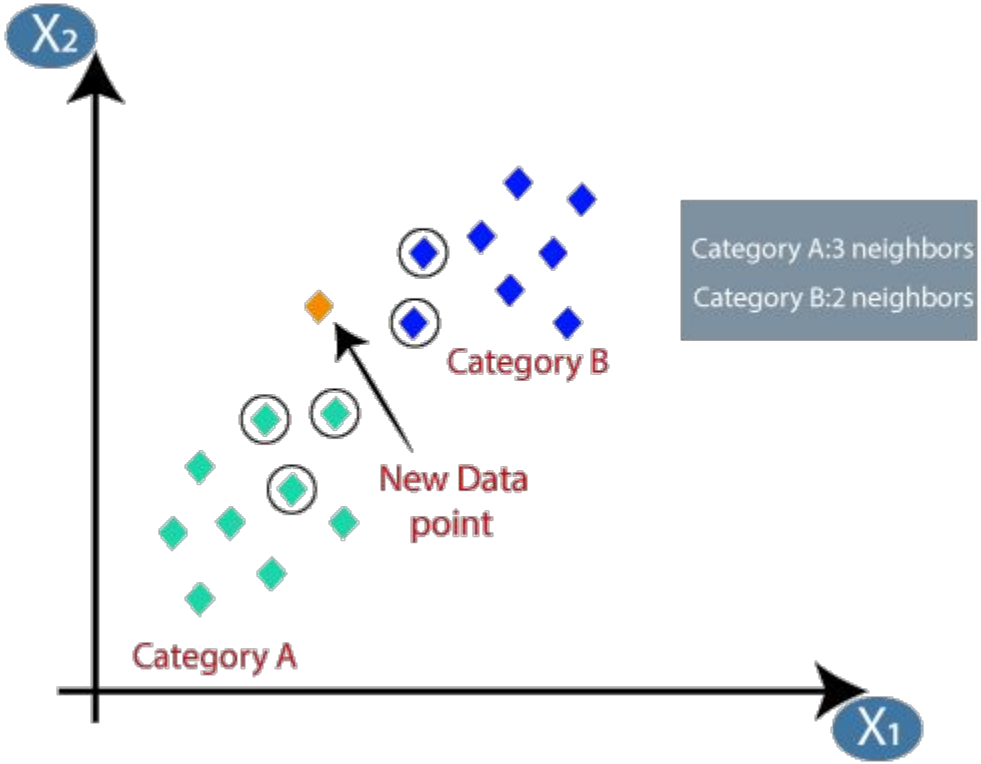


Euclidean Distance between A_1 and $B_2 = \sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

Nearest Neighbors

By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B

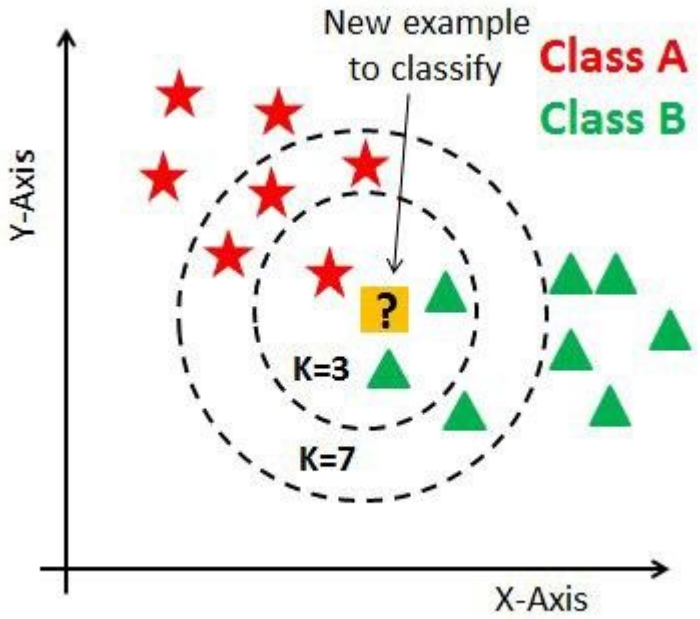
As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A



K value

Kvalue indicates the count of the nearest neighbors

If we proceed with K=3, then we predict that test input belongs to class B, and if we continue with K=7, then we predict that test input belongs to class A



How to choose a K value?

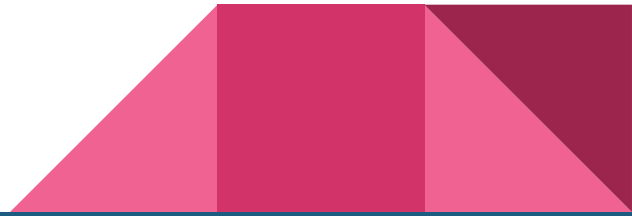
There are no predefined statistical methods to find the most favorable value of K

Initialize a random K value and start computing

Choosing a small value of K leads to unstable decision boundaries

The substantial K value is better for classification as it leads to smoothening the decision boundaries

Derive a plot between error rate and K denoting values in a defined range. Then choose the K value as having a minimum error rate



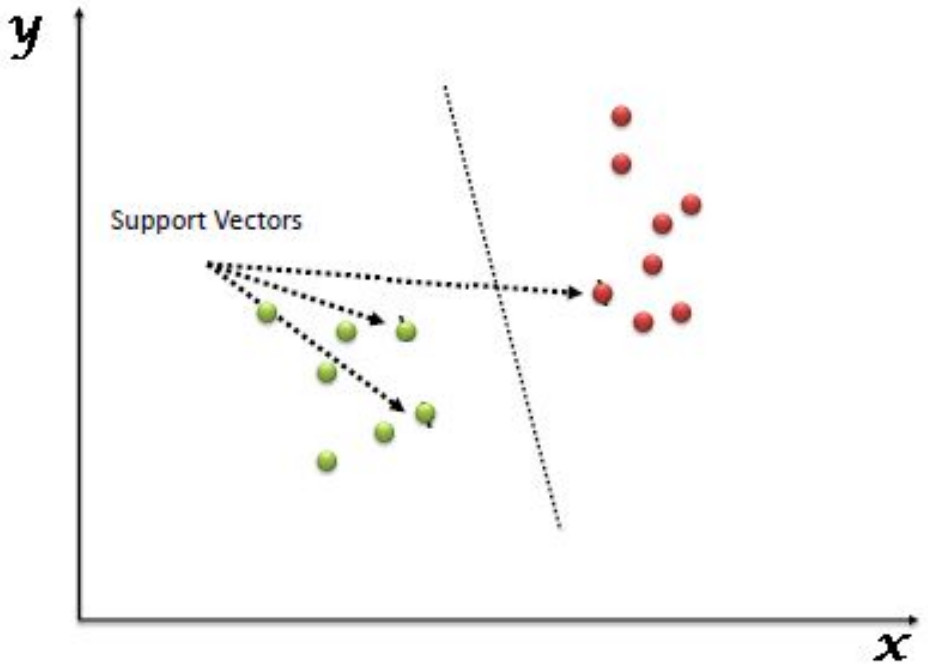
SVMs (Support Vector Machines)

Support Vector Machines

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future.

This best decision boundary is called a hyperplane

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine



Hyperplane

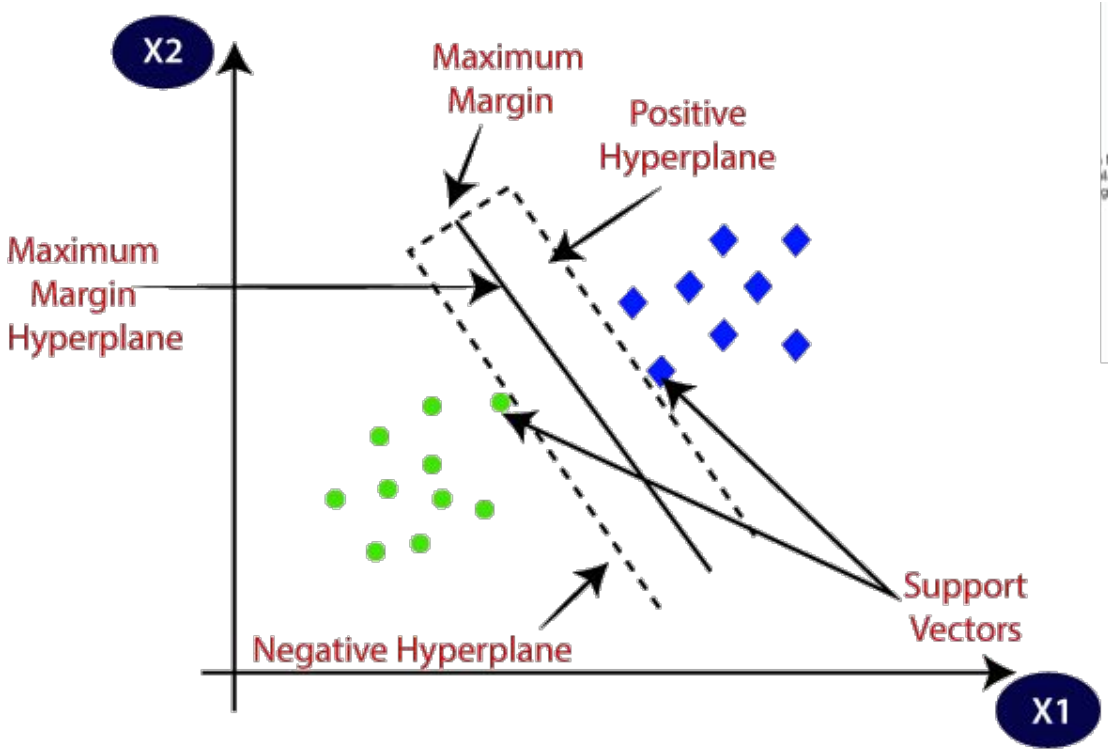
There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features, then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

Support Vectors

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.



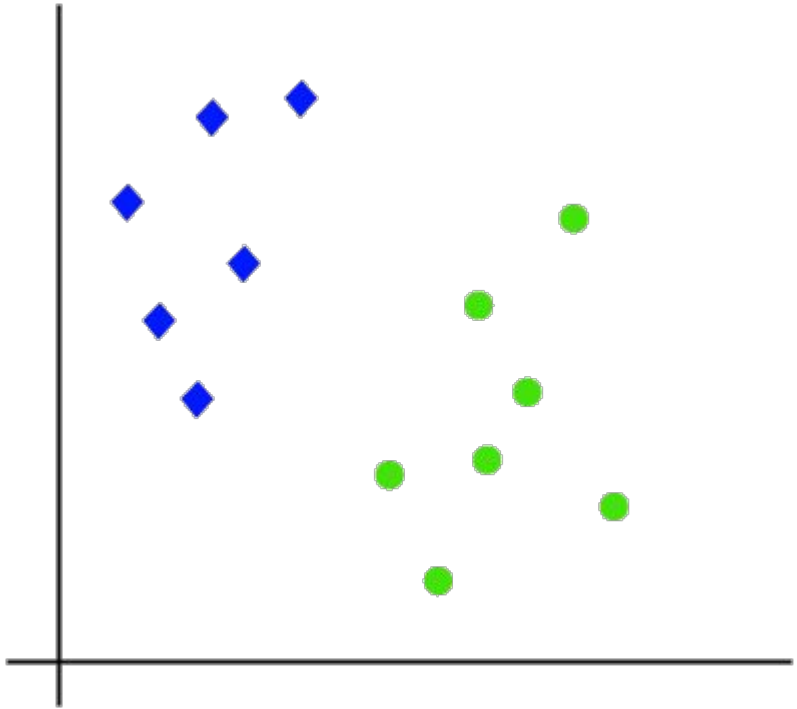
Types of SVM

Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier

Non-linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier

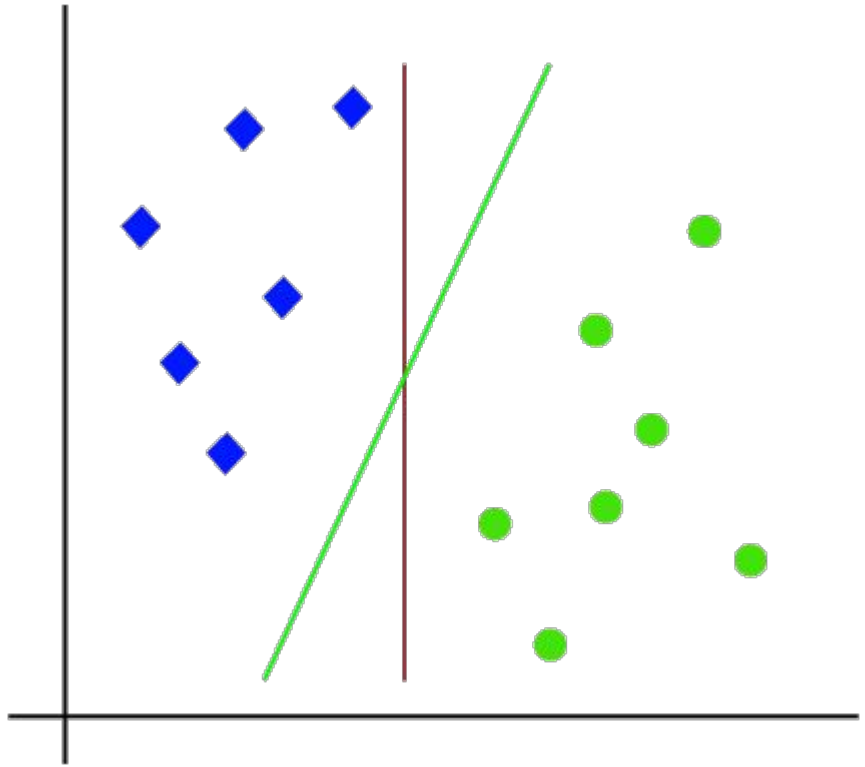
Linear SVM

Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue.



Linear SVM

So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes.



Linear SVM

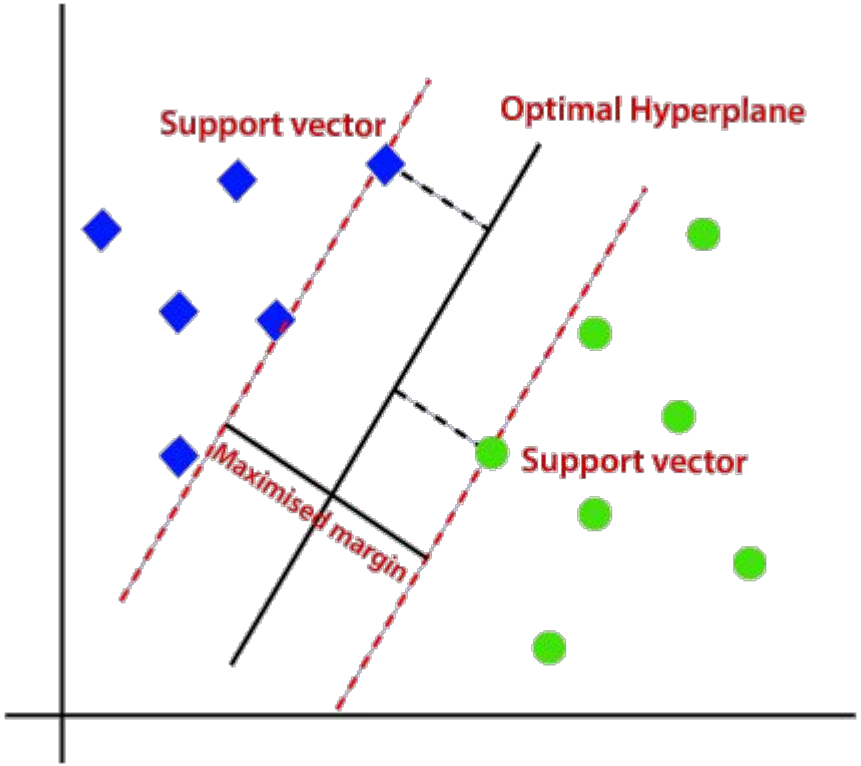
Hence, the SVM algorithm helps to find the best line or decision boundary.

This best boundary or region is called as a **hyperplane**.

SVM algorithm finds the closest point of the lines from both the classes.

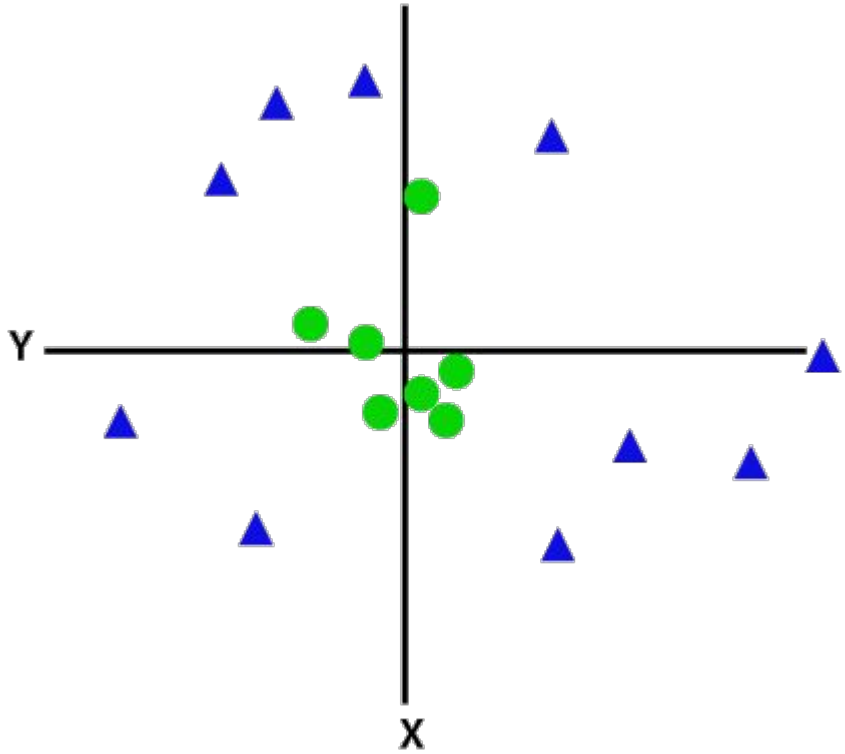
These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin** and the goal of SVM is to maximize this margin.

The hyperplane with maximum margin is called the **optimal hyperplane**.



Non-Linear SVM

If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line.

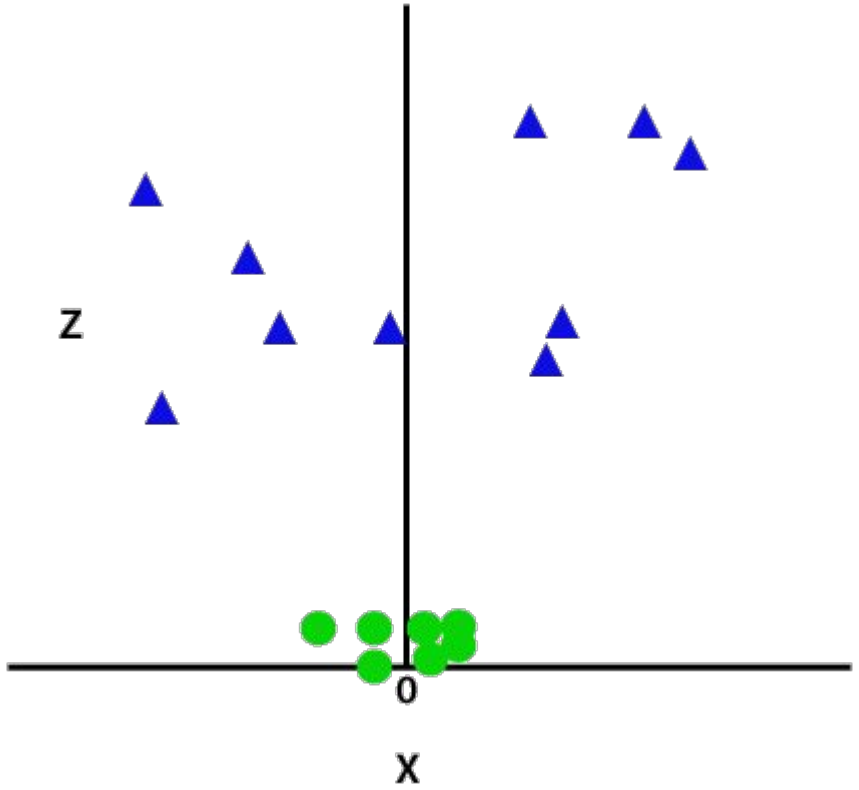


Non-Linear SVM

So to separate these data points, we need to add one more dimension.

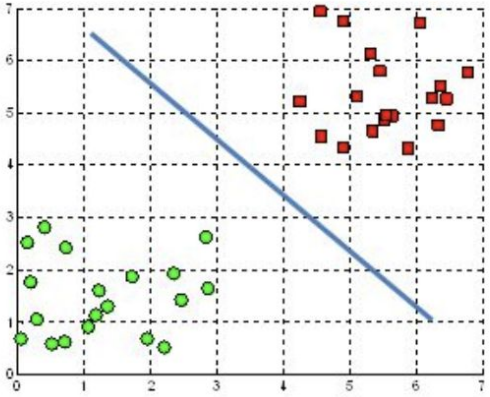
For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z

$$z = x^2 + y^2$$

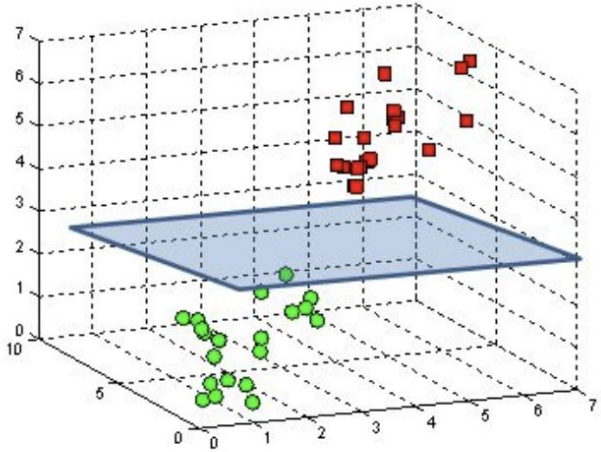


Hyperplane in 3D

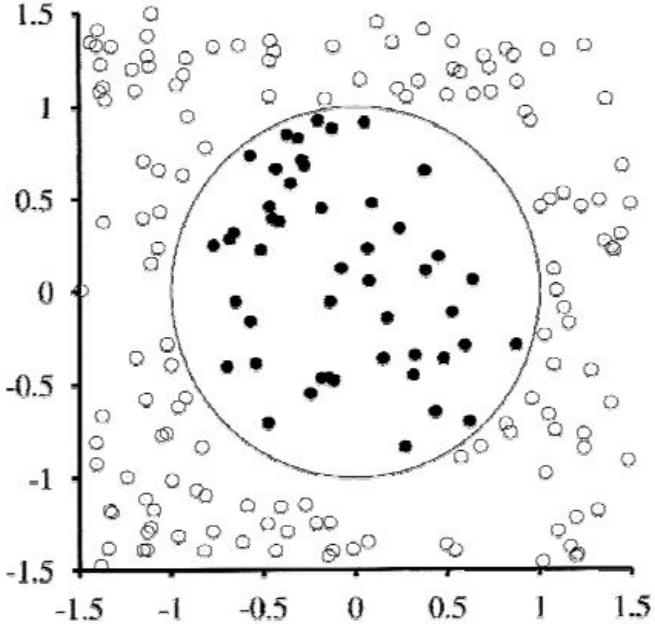
A hyperplane in $\mathbb{R}^2$ is a line

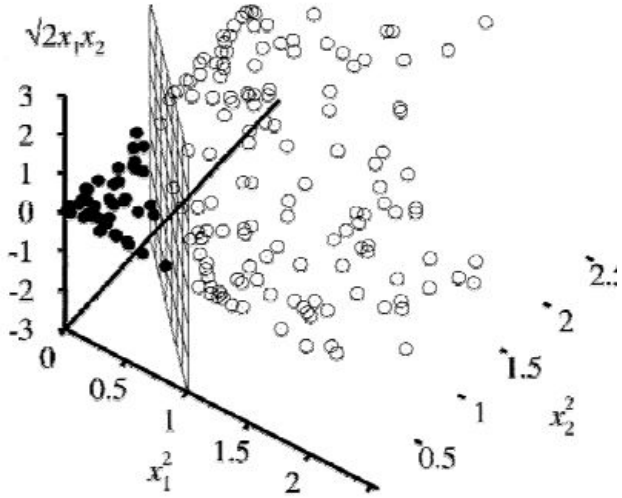


A hyperplane in $\mathbb{R}^3$ is a plane



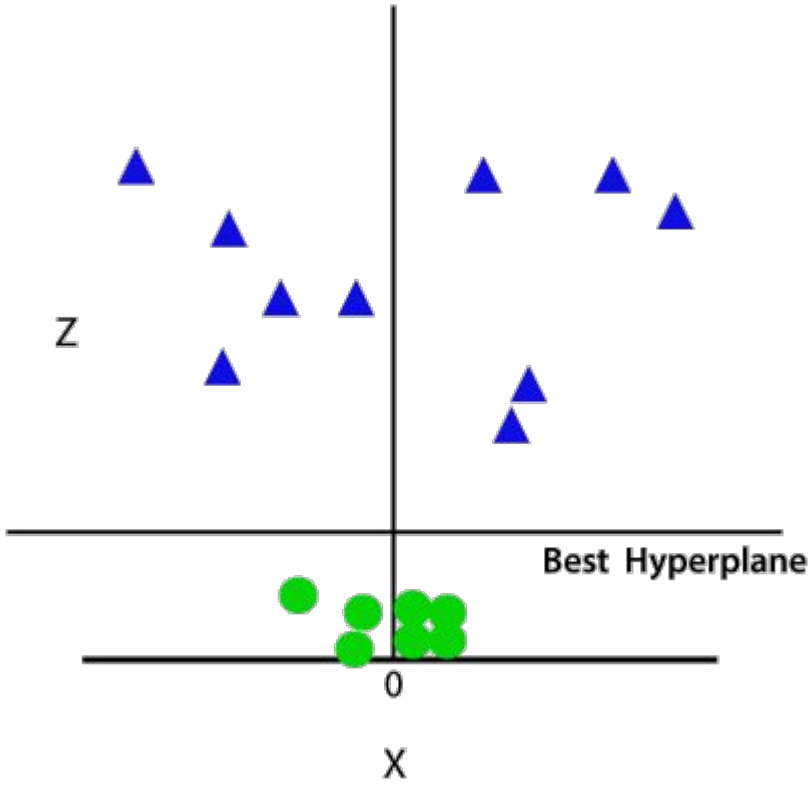
Better visualization





Non-Linear SVM

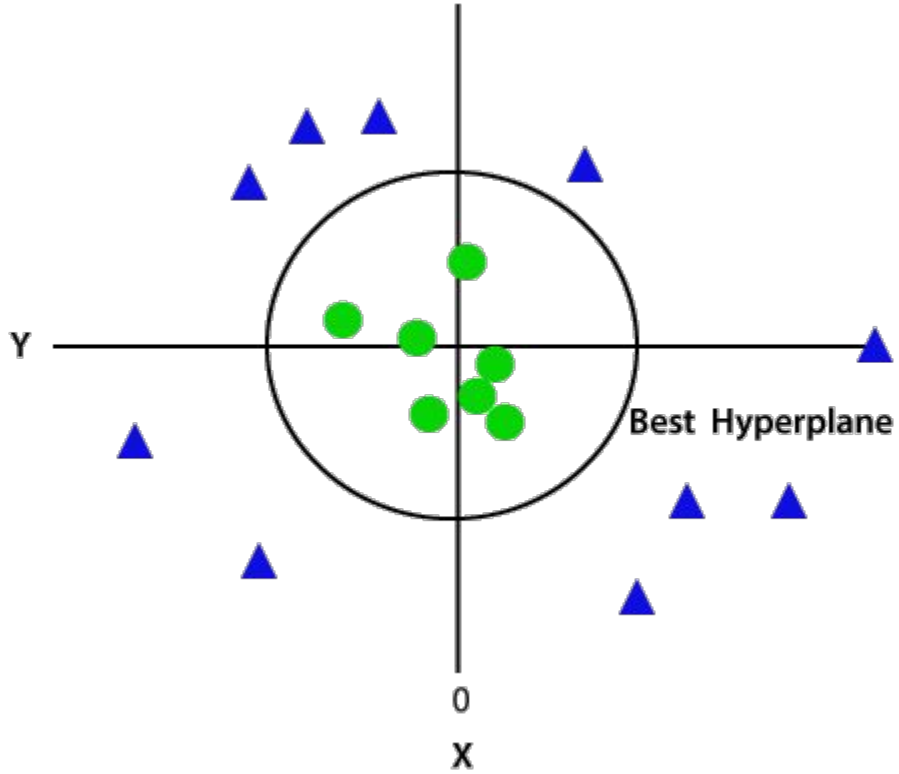
Now SVM will divide the datasets into classes in the following way.



Non-Linear SVM

Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis.

If we convert it in 2d space with z=1, it will become →



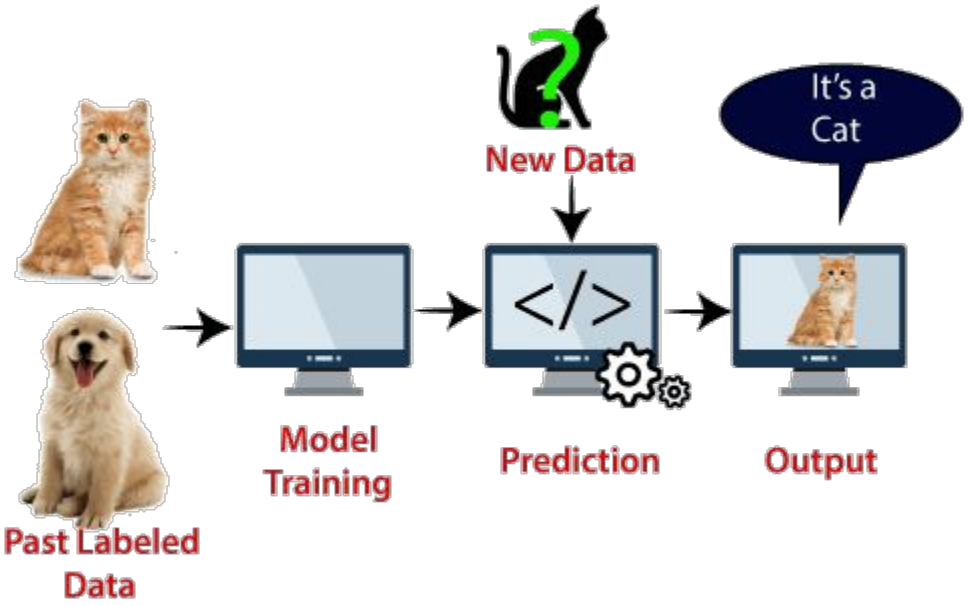
Example

SVM can be understood with the example that we have used in the KNN classifier.

Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm

We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature.

So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog



Resources

- Linear Regression - https://mlu-explain.github.io/linear-regression/
- Logistic Regression - https://mlu-explain.github.io/logistic-regression/
- Decision Trees - https://mlu-explain.github.io/decision-tree/
- K-Nearest Neighbor - https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4
- SVM - https://towardsdatascience.com/demystifying-support-vector-machines-8453b39f7368

Questions
&
Discussion

Hands-on

Hands-on Title

All hands-on materials available at
github.com/Gradient-PG/gradient-live-session

Thank you!
See you next week on
Introduction to Deep Learning.