

Machine Learning

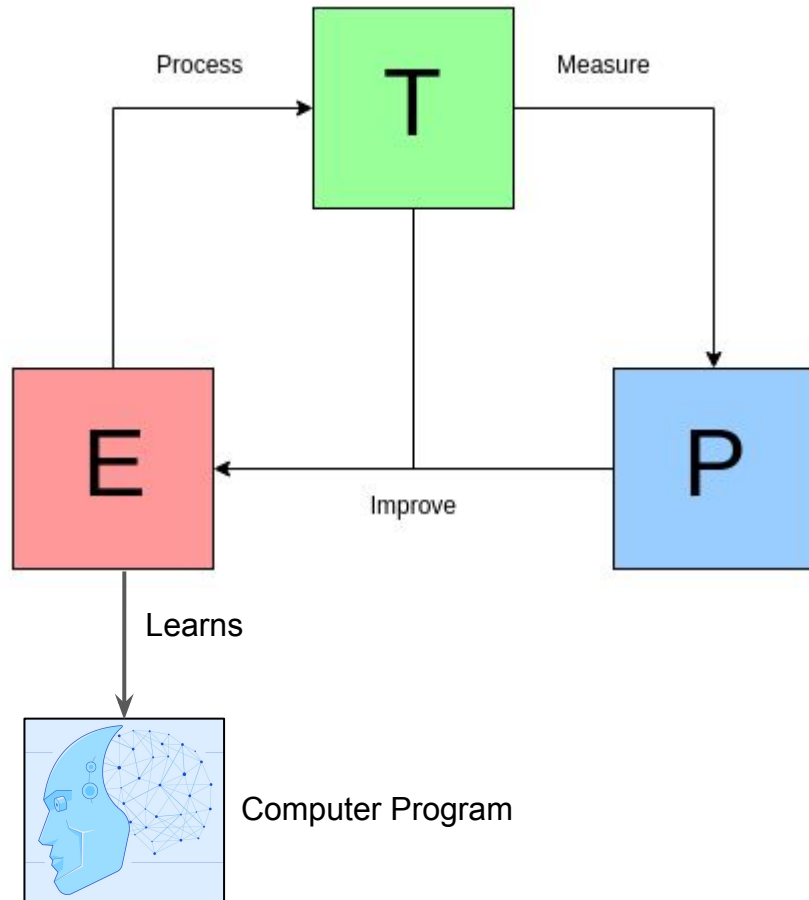
Subha Ilamathy

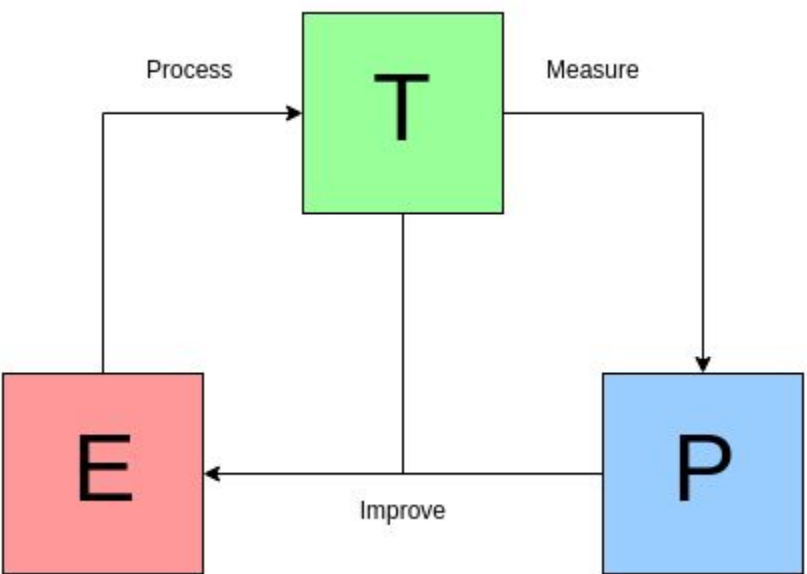
Machine Learning ?

A computer program is said to learn from **Experience E** with respect to some class of **Tasks T** and **Performance measure P** ,

If its performance P at tasks in T , improves with experience E .

- Tom Mitchell





E	T	P
EXPERIENCE	TASK	PERFORMANCE MEASURE
having labeled data	processing an example	measuring performance
supervised learning	classification	accuracy

Example for E -T -P

Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam. What is the task T, Experience and Performance in this setting?

1. The number of emails correctly classified as spam/not spam.
2. Classifying emails as spam or not spam
3. Watching you label emails as spam or not spam

E	T	P
EXPERIENCE	TASK	PERFORMANCE MEASURE
?	?	?

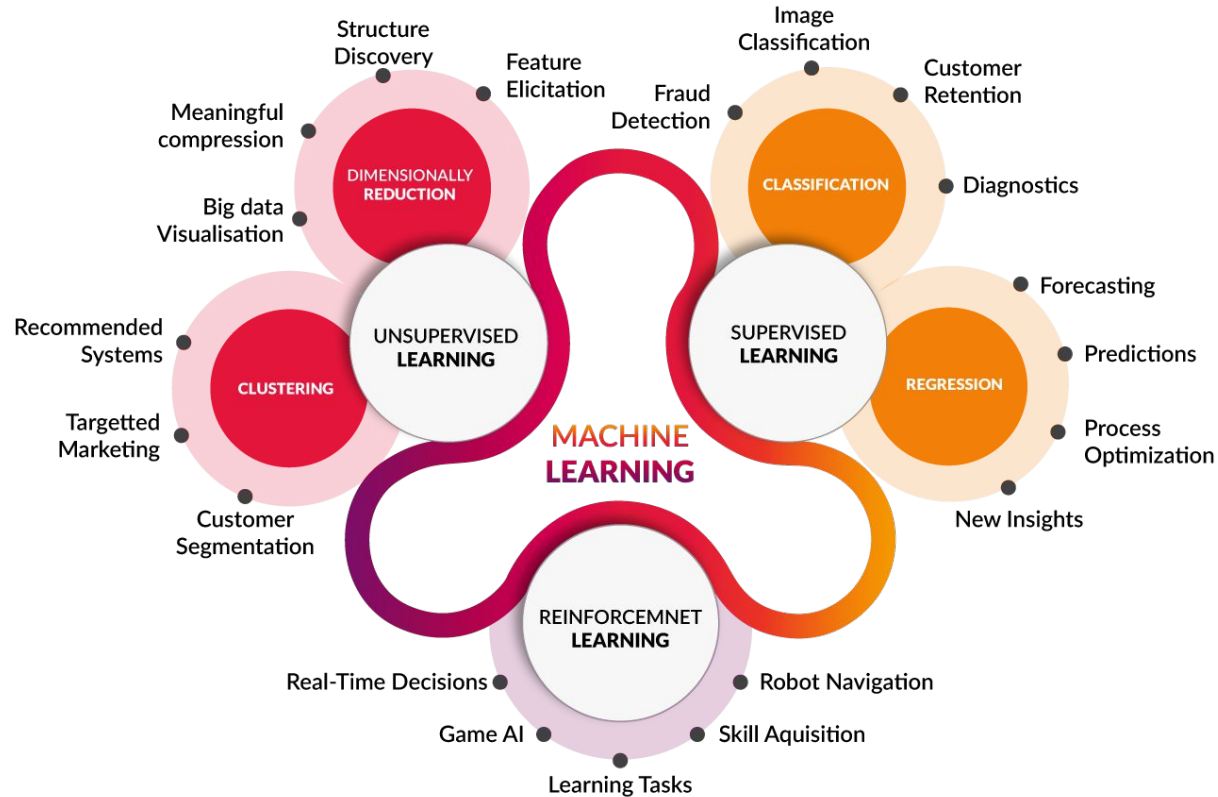
Example for E -T -P

Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam. What is the task T, Experience and Performance in this setting?

1. The number of emails correctly classified as spam/not spam.
2. Classifying emails as spam or not spam
3. Watching you label emails as spam or not spam

E	T	P
EXPERIENCE	TASK	PERFORMANCE MEASURE
3	2	1

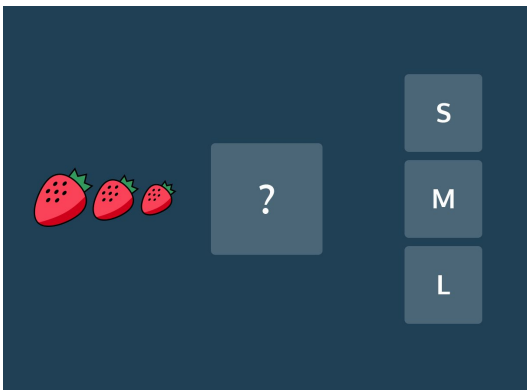
Machine Learning Types



Supervised Learning Types

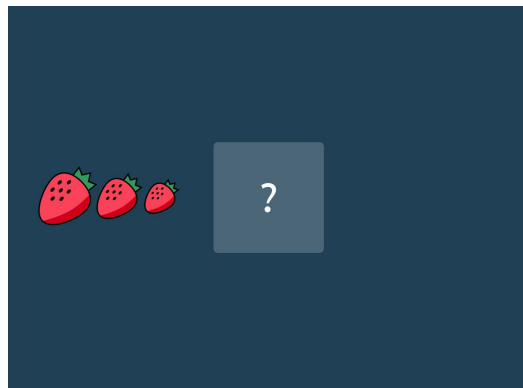
Classification is used to predict a **discrete** label.

- Outputs fall under a finite set of possible outcomes.
- confined to 2 class or multi class labels



Regression is used to predict outputs that are **continuous**.

- Outputs Flexibly determined based on the inputs
- Not Confined to a set of output labels.



Linear Regression

Suppose you are thinking of selling your home. Different sized homes around you have sold for different amounts.



Your home is 3000 square feet. How much should you sell it for?

You have to look at the existing data and predict a price for your home. This is called **linear regression**.

E -T -P of Housing Price Prediction

Suppose you are thinking of selling your home. Different sized homes around you have sold for different amounts.

1. Predicting the price for your home.
2. Watching the pattern in the sample labelled data
3. Validating the prediction price and the actual price.

E	T	P
EXPERIENCE	TASK	PERFORMANCE MEASURE
?	?	?

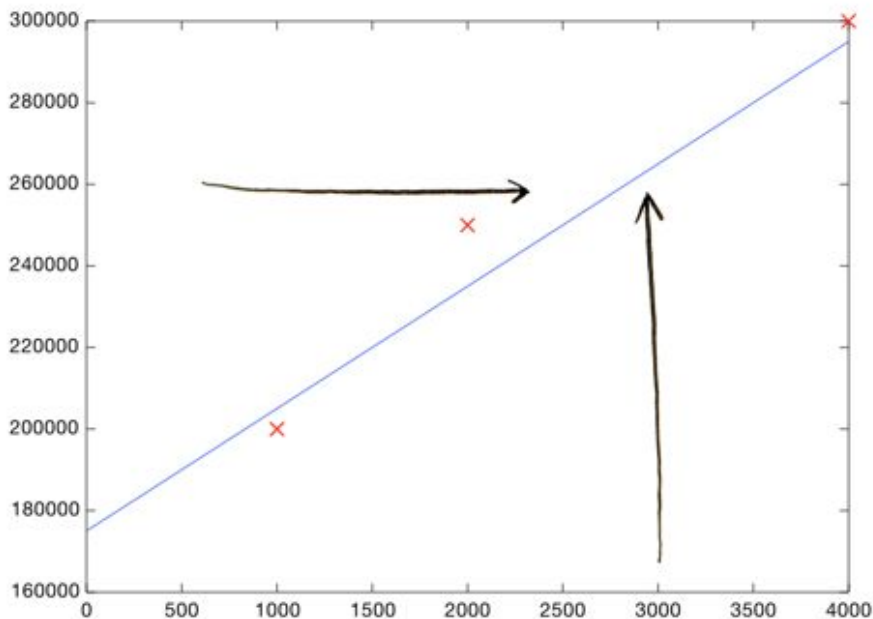
E -T -P of Housing Price Prediction - Answer

Suppose you are thinking of selling your home. Different sized homes around you have sold for different amounts.

1. Predicting the price for your home.
2. Watching the pattern in the sample labelled data
3. Validating the prediction price and the actual price.

E	T	P
EXPERIENCE	TASK	PERFORMANCE MEASURE
2	1	3

Linear Regression



GOOD FIT



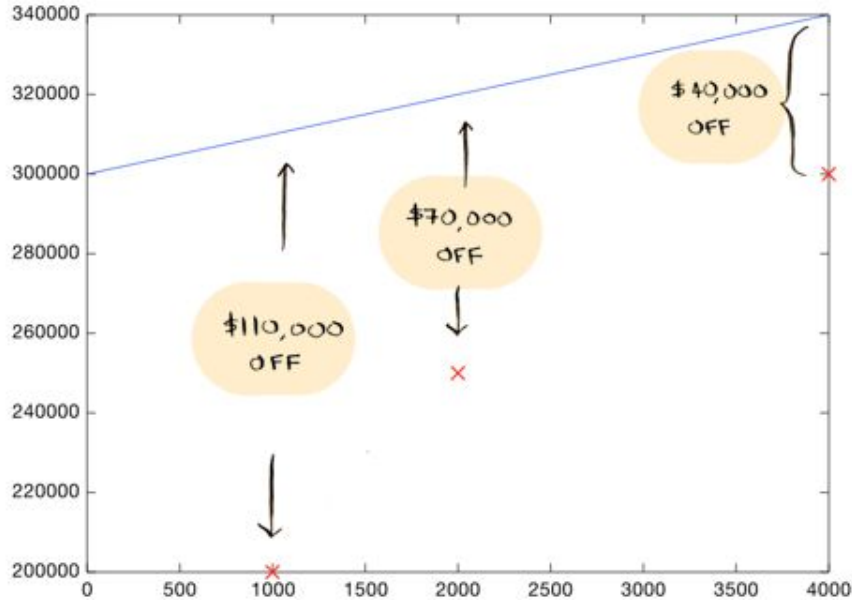
Boom! Your home should sell for \$260,000.

That's all there is to it.

You plot your data, eyeball a line, and use the line to make predictions.

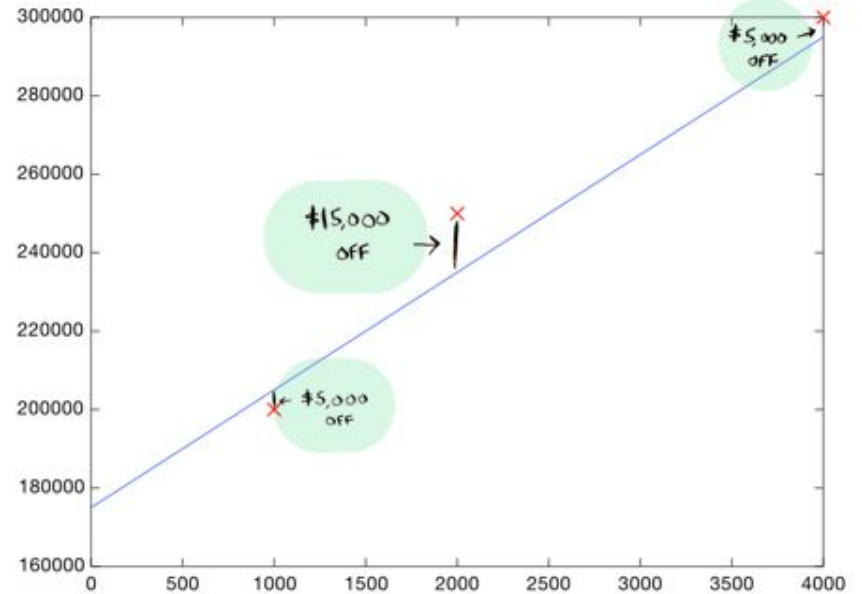
Of course we don't want to eyeball a line, we want to compute exact line that best "fits" our data.

BAD FIT



On average, this line is \$73,333 off ($\$110,000 + \$70,000 + \$40,000 / 3$)

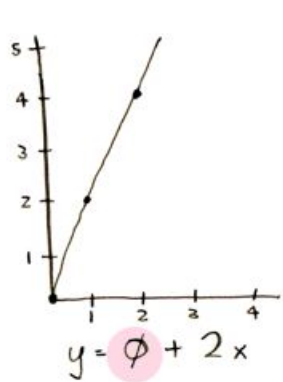
GOOD FIT



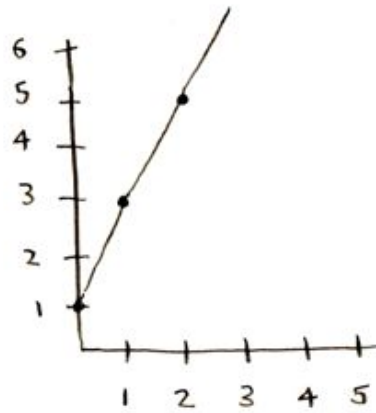
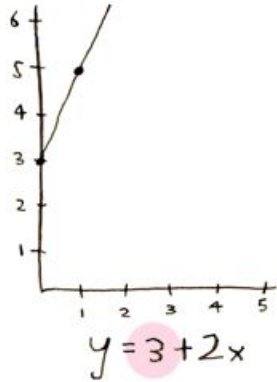
$$\frac{\$5,000 + \$15,000 + \$5,000}{3} = \$8,333 \text{ OFF ON AVERAGE}$$

This \$8,333 is called the cost of using this line.
The "cost" is how far the line is from the real data.

Cost Function



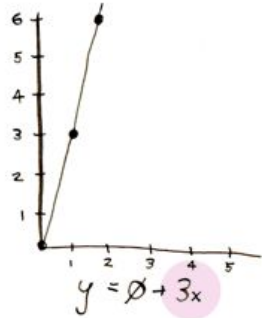
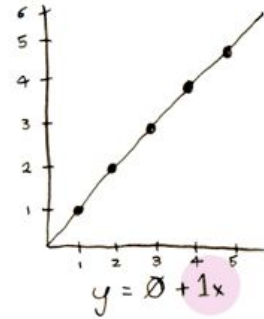
The first number tells you how high the line should be at the start.



"Come up with two pretty good numbers that make the line fit the data"

$$y = \text{theta}_0 + \text{theta}_1 x$$

or $y = b + W x$
or $y = w[1] + w[0]x$



The second number tells you the angle of the line.

Cost Function

FOR A GIVEN THETA 0 AND THETA 1...

2. FIND THE DIFFERENCE BETWEEN THE PREDICTED AND ACTUAL VALUES

1. THE PREDICTED VALUE

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^i) - y^i)^2$$

4. FIND THE AVERAGE

3. FIND ALL THE DIFFERENCES BETWEEN PREDICTED AND ACTUAL

The diagram illustrates the components of the Cost Function formula. A pink bubble at the top left states 'FOR A GIVEN THETA 0 AND THETA 1...'. A yellow bubble at the top right says '2. FIND THE DIFFERENCE BETWEEN THE PREDICTED AND ACTUAL VALUES'. A yellow bubble below it says '1. THE PREDICTED VALUE'. A yellow bubble at the bottom right says '3. FIND ALL THE DIFFERENCES BETWEEN PREDICTED AND ACTUAL'. A yellow bubble at the bottom left says '4. FIND THE AVERAGE'. Brackets connect these steps to parts of the formula: '1. THE PREDICTED VALUE' points to $h_{\theta}(x^i)$; '2. FIND THE DIFFERENCE BETWEEN THE PREDICTED AND ACTUAL VALUES' points to $(h_{\theta}(x^i) - y^i)$; '3. FIND ALL THE DIFFERENCES BETWEEN PREDICTED AND ACTUAL' points to the summation $\sum_{i=1}^m$; and '4. FIND THE AVERAGE' points to the $\frac{1}{2m}$ coefficient.

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

In this formula,

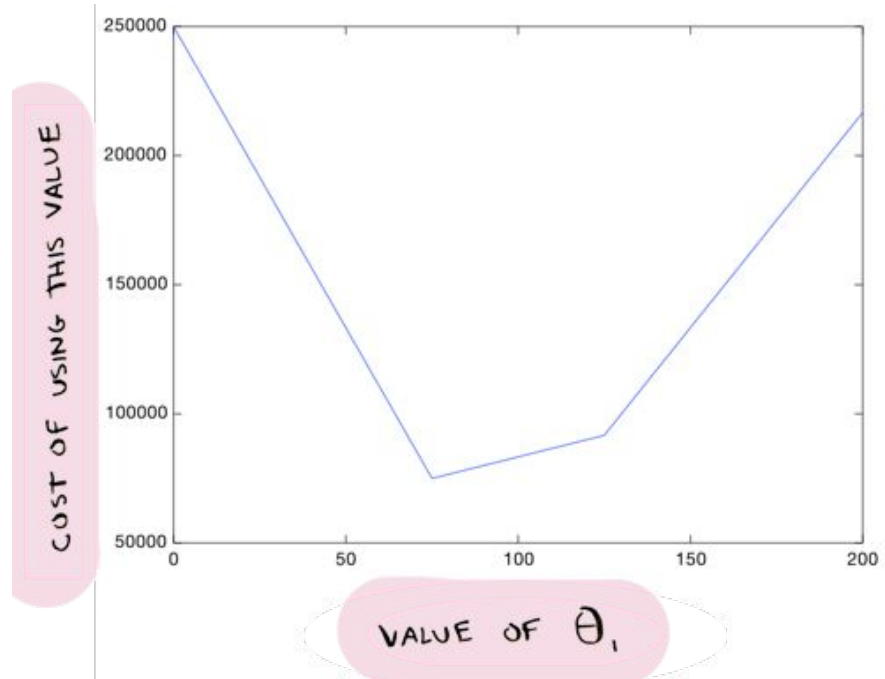
$h(x)$ represents the predicted value

y represents the True value

Gradient Descent

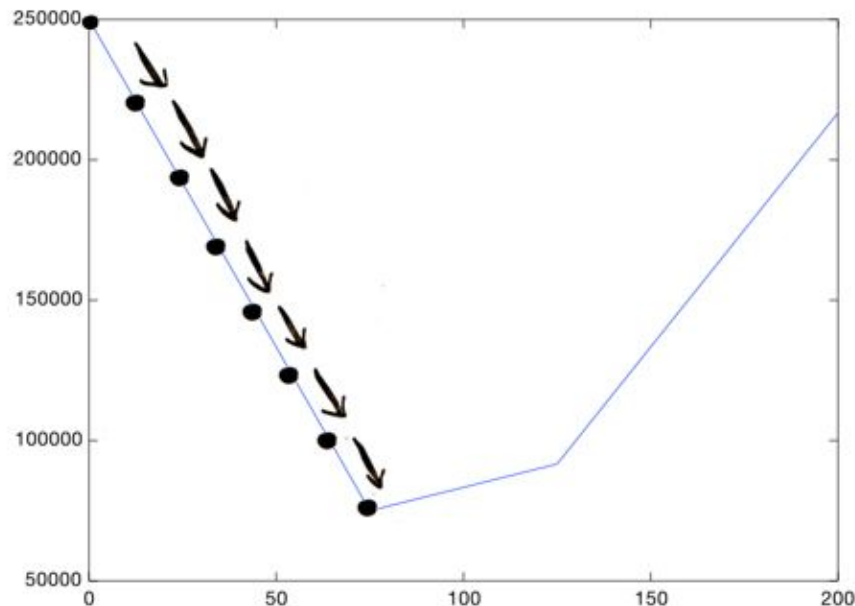
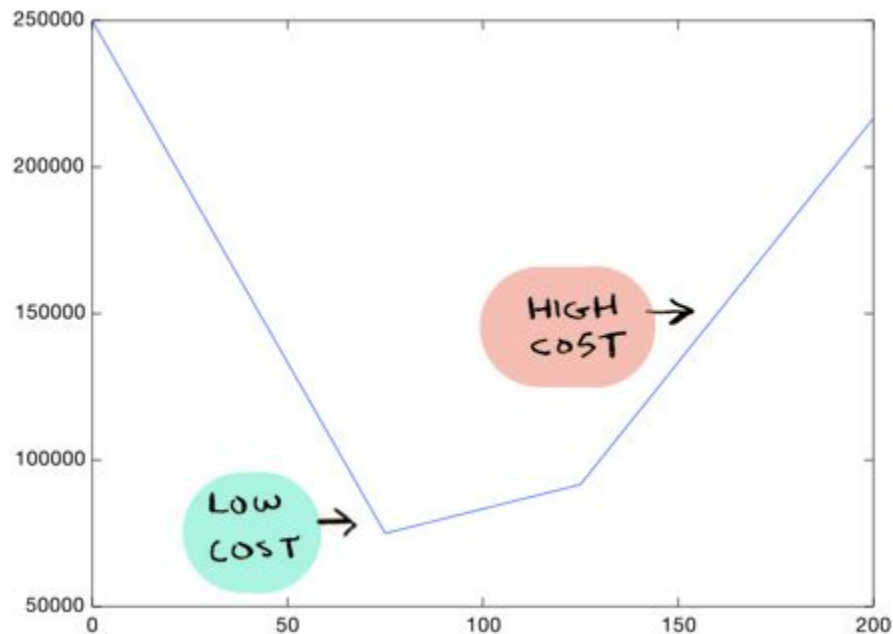


Suppose we decide to leave θ_0 at zero.
So we experiment with what value θ_1



Remember above graph is not price data graph but the cost data graph obtained by changing parameters

Gradient Descent - Reaching Low Cost



This is *gradient descent*: going down bit by bit till you hit the bottom.

How do you figure out which way is down?

The answer will be obvious to calculus experts but not so obvious for the rest of us:
you take the derivative at that point

Gradient Descent - Update Parameters

TO UPDATE θ_0 :

ALPHA (α), CONTROLS
HOW FAST θ_0 GETS UPDATED

$$\theta_0 = \theta_0 - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^i) - y^i)}_{\text{PARTIAL DERIVATIVE}}$$

TO UPDATE θ_1 :

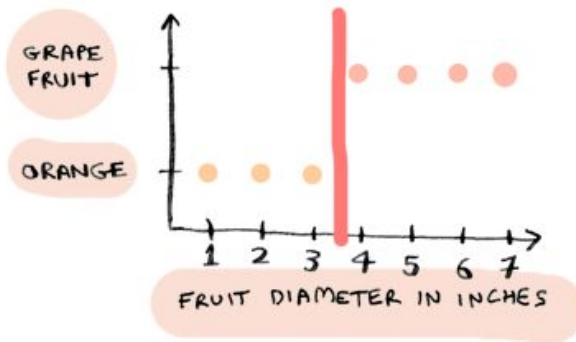
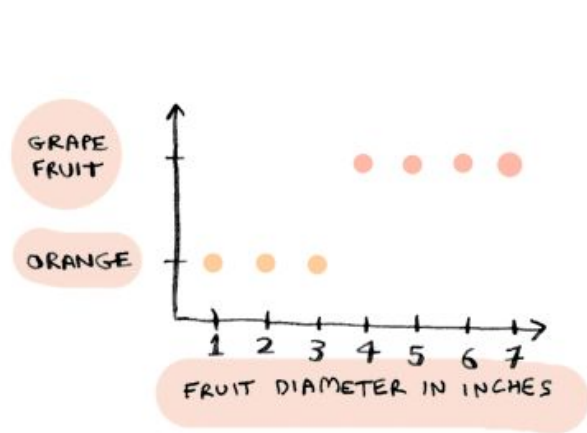
$$\theta_1 = \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^i) - y^i) \cdot x^i$$

Let's code above concepts

[Linear Regression - Numpy and Tensorflow](#)

Logistic Regression - But used for Classification

So instead of trying to predict a number (what price should I sell my house for?),
You are trying to classify something (hey is this a grapefruit?)



Fruit Diameter is our data points (x)
Orange or Grape is our output (y)

You plot your data, then draw a line.
But we call line as **decision boundary**

Prediction Function - Logistic Regression

X	y
1000	\$200k
2000	\$250k
4000	\$300k

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

Linear Regression

Output can be >1 or <0
directly takes y as output.

FRUIT DIAMETER IN INCHES	X	y
	1	0
	2	0
	3	0
	4	1
	5	1

0 = NOT A GRAPEFRUIT
1 = GRAPEFRUIT

$$h_{\theta}(x) = g(\theta_0 + \theta_1 x)$$

SIGMOID
FUNCTION

Logistic Regression
(Classification)

X	y	PREDICTED
1	0	0.6%
2	0	4.7%
3	0	26.9%
4	1	73.1%
5	1	95.2%

ACTUAL VALUE
1 = IS A GRAPEFRUIT

PERCENT
CHANCE
THAT THIS
IS A GRAPEFRUIT

Prediction function for logistic regression
has to be a value in range $0 \leq y \leq 1$ which
is later converted to 0% or 100%

Sigmoid Function

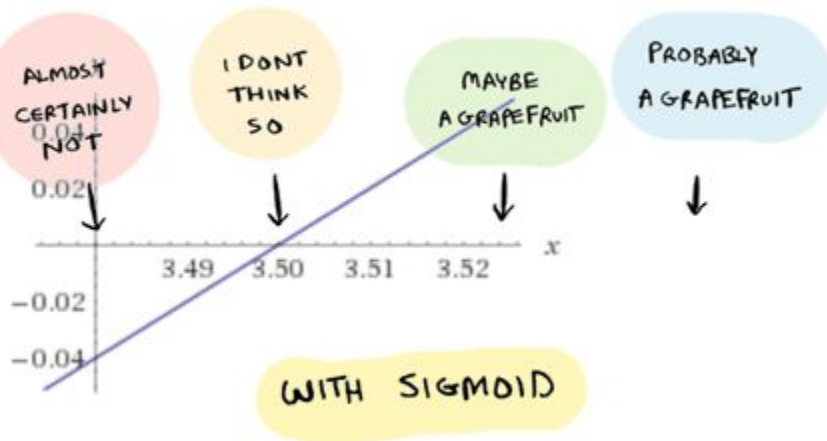
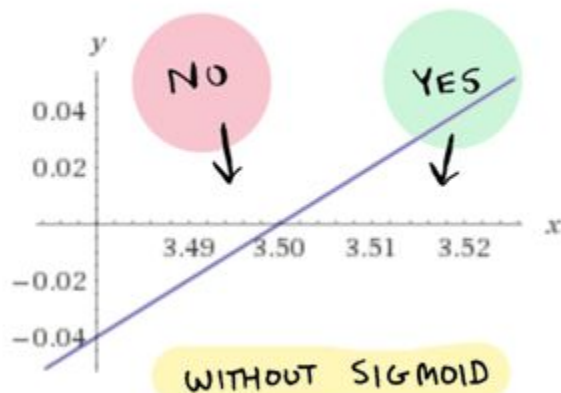
$$\text{Let } z = h_{\theta}(x) = \theta_0 + \theta_1 x$$

The sigmoid function constraints the result to between **0** and **1**

$$g(z) = \frac{1}{1 + e^{-z}}$$

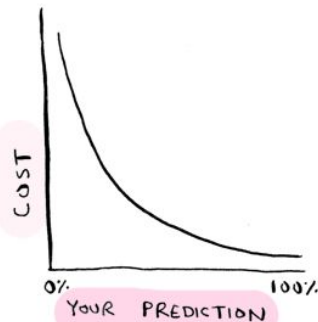
$$h_{\theta}(x) = g(-7 + 2x)$$

SIGMOID
FUNCTION



Cost Function - Logistic Regression

Correct Label $y = 1$ (grapefruit)



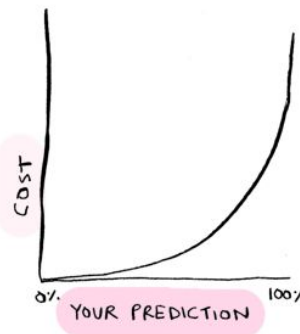
if correct label is 1
you predicted $h(x) = 1$
(100% probability it is a grapefruit),
there's no penalty

But if you predicted as $h(x) = 0$
(0% probability it is a grapefruit),
then you get penalized heavily.

$$-\frac{1}{m} \sum_{i=1}^m \begin{cases} -\log(h_{\theta}(x_i)) & \text{if } y_i = 1 \\ -\log(1 - h_{\theta}(x_i)) & \text{if } y_i = 0 \end{cases}$$

$$\begin{aligned} P(y=0) + P(y=1) &= 1 \\ P(y=0) &= 1 - P(y=1) \end{aligned}$$

Correct Label $y = 0$ (orange)



if correct label is 0
you predicted $h(x) = 0$
(100% probability it is an orange),
there's no penalty

But if you predicted as 1
(0% probability it is an orange),
then you get penalized heavily.

Optimized Cost Function

$$-\frac{1}{m} \sum_{i=1}^m \begin{cases} -\log(h_{\theta}(x_i)) & \text{if } y_i = 1 \\ -\log(1 - h_{\theta}(x_i)) & \text{if } y_i = 0 \end{cases}$$

THIS EVALUATES
TO ZERO IF
 $y_i = 0$

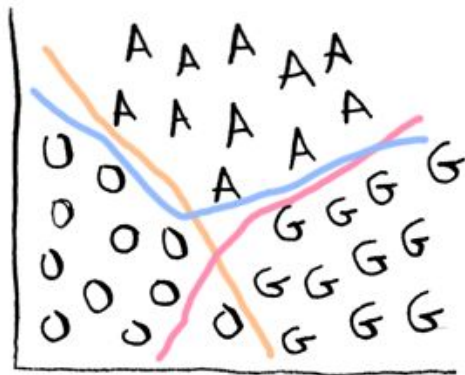
THIS EVALUATES
TO ZERO IF
 $y_i = 1$

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m y_i \cdot \log(h_{\theta}(x_i)) + (1 - y_i) \cdot \log(1 - h_{\theta}(x_i))$$

Multi Class Classification

Introduces some non-linearity

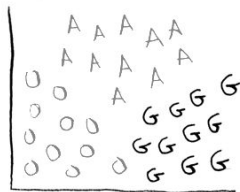
$$h_{\theta}(x) = g(\theta_0 + \theta_1 x + \theta_2 (x - \theta_3)^2)$$



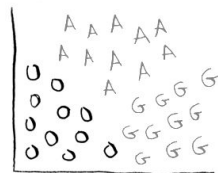
(O = orange, G = grapefruit, A = apple).

We need multiple decision boundaries

one vs all classification



So let's say
25% chance this is a
grapefruit, and
there's a 75% chance
orange or apple



So either it IS an orange,
or ISN'T (grapefruits
and apples get lumped
together). Lets say output
for orange is 75%.



Similarly, Let's say
for apple it is 30%

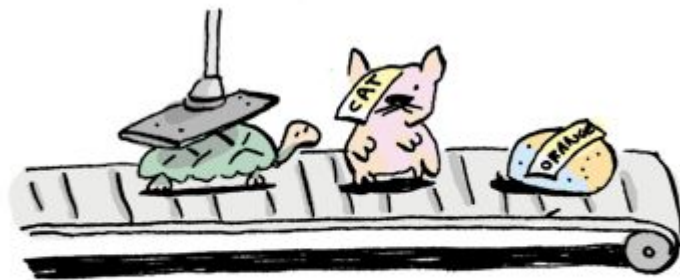
**25% chance this is a grapefruit,
75% chance this is an orange, and
30% chance this is an apple**

Time for Neural Networks

Logistic regression is not good for something like this:



This is where non-linear classifiers shine. Neural networks and support vector machines (SVMs) are both good at non-linear classification.

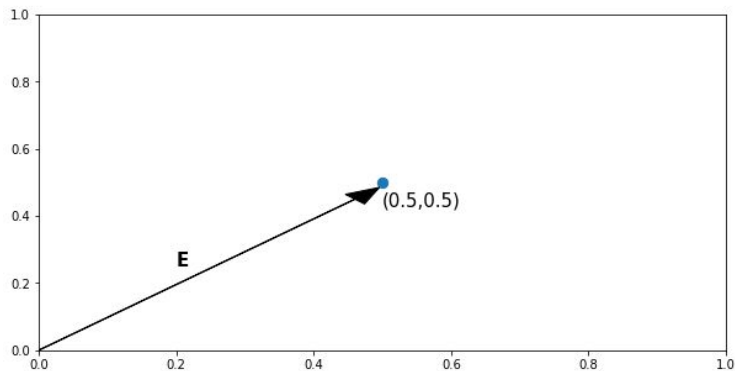


CLASSIFICATION MACHINE

Let's Code

<https://colab.research.google.com/drive/1BMM4rVv8ltkX1kK3JGmJOYIM03YRWYJj?usp=sharing>

Demystifying Vectors



A quantity that does not require additional information (such as direction) with it is represented as **scalar**.

Whereas, a quantity that needs direction to be specified alongside its magnitude is represented with a **vector**.

The benefit of representing data as vectors is — we can leverage vector algebra to find patterns or relationships within our data.

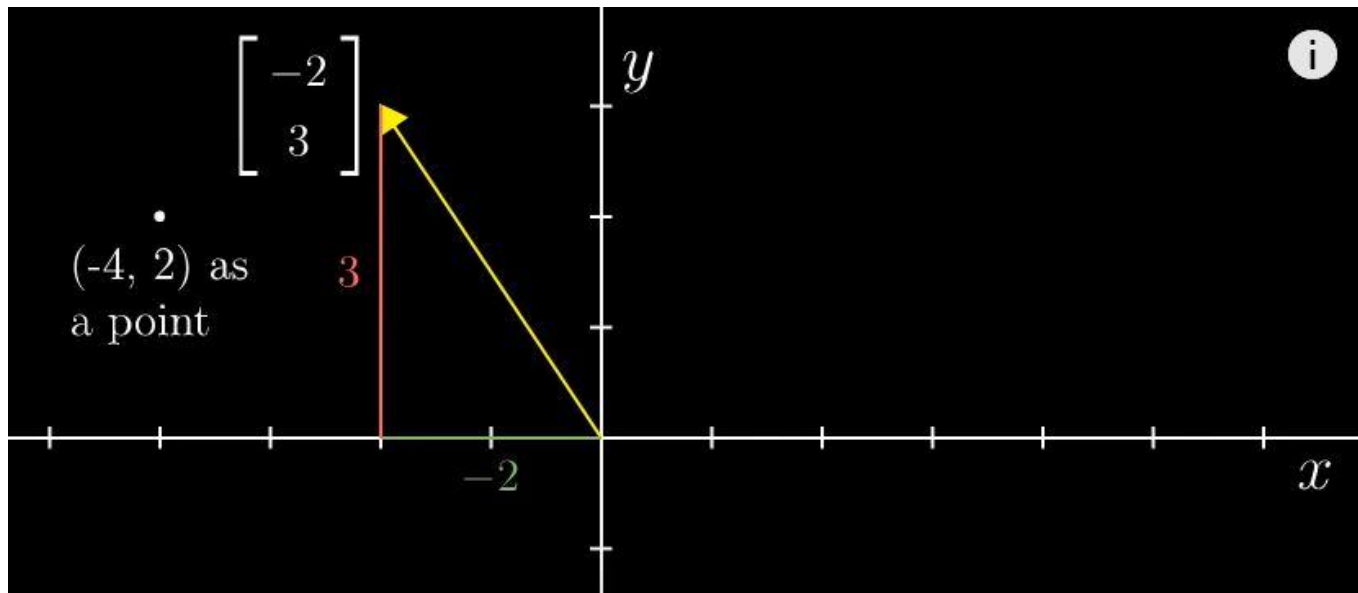
Demystifying Vectors

Vectors \Leftrightarrow lists of numbers

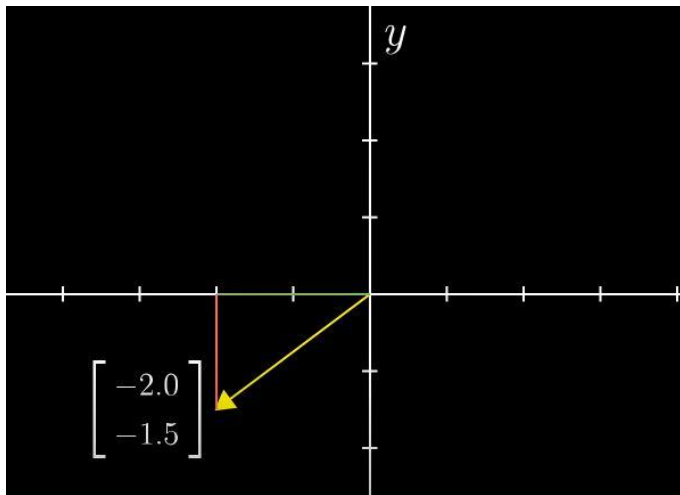


$\left[\begin{array}{c} 2,600 \text{ ft}^2 \\ \$300,000 \end{array} \right]$ } 2 dimensional

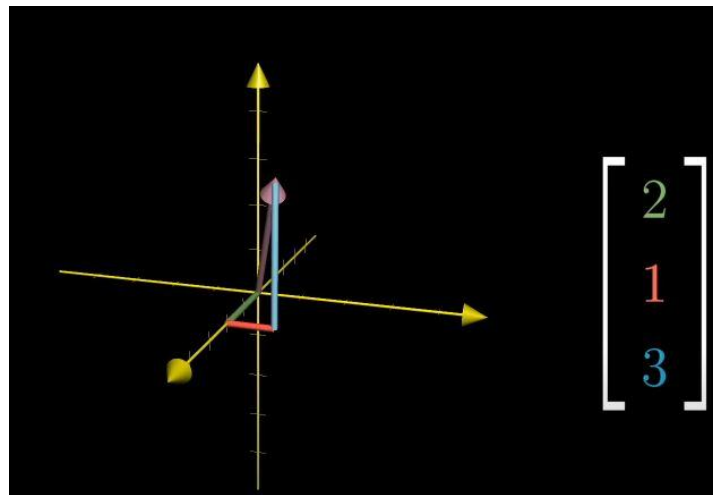
Demystifying Vectors



Demystifying Vectors

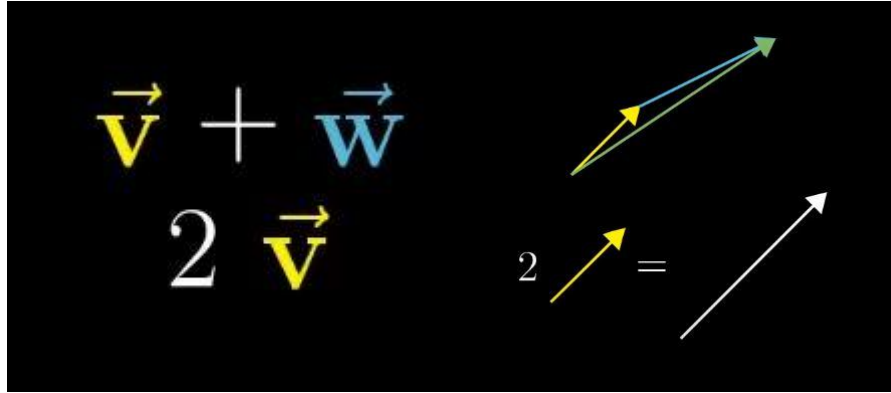


2D



3D

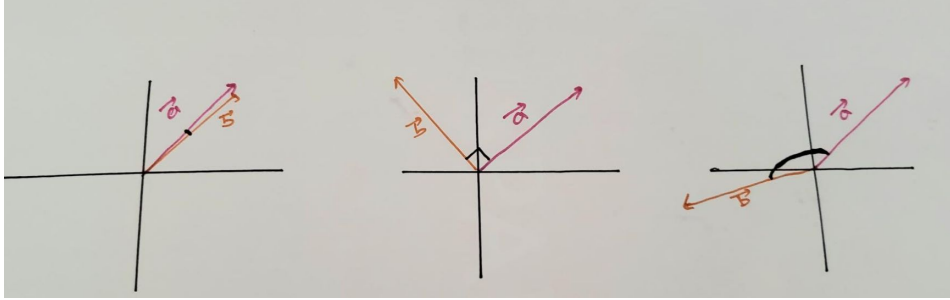
Demystifying Vectors



Vector Addition &
Vector Multiplication

$$\begin{bmatrix} 3 \\ -5 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 + 2 \\ -5 + 1 \end{bmatrix}$$
$$2 \begin{bmatrix} 3 \\ -5 \end{bmatrix} = \begin{bmatrix} 2(3) \\ 2(-5) \end{bmatrix}$$

Demystifying Vectors



The angle between the vectors indicates “similarity” between them.

The vectors in the **same direction** (close to 0 degrees angle) **are similar**

The vectors in the **opposite direction** (close to 180 degrees angle) **are dissimilar**.

Looking at the vectors in vector space, you can quickly compare them to check if there is a **relationship**.

For example, you deduce that **similar vectors** will have smaller angle between them i.e their orientation will be close to each other.

More Resources for Vector Algebra

[3Blue1Brown](#)

[What's a Tensor?](#)

Parametric Vs Non-Parametric Models

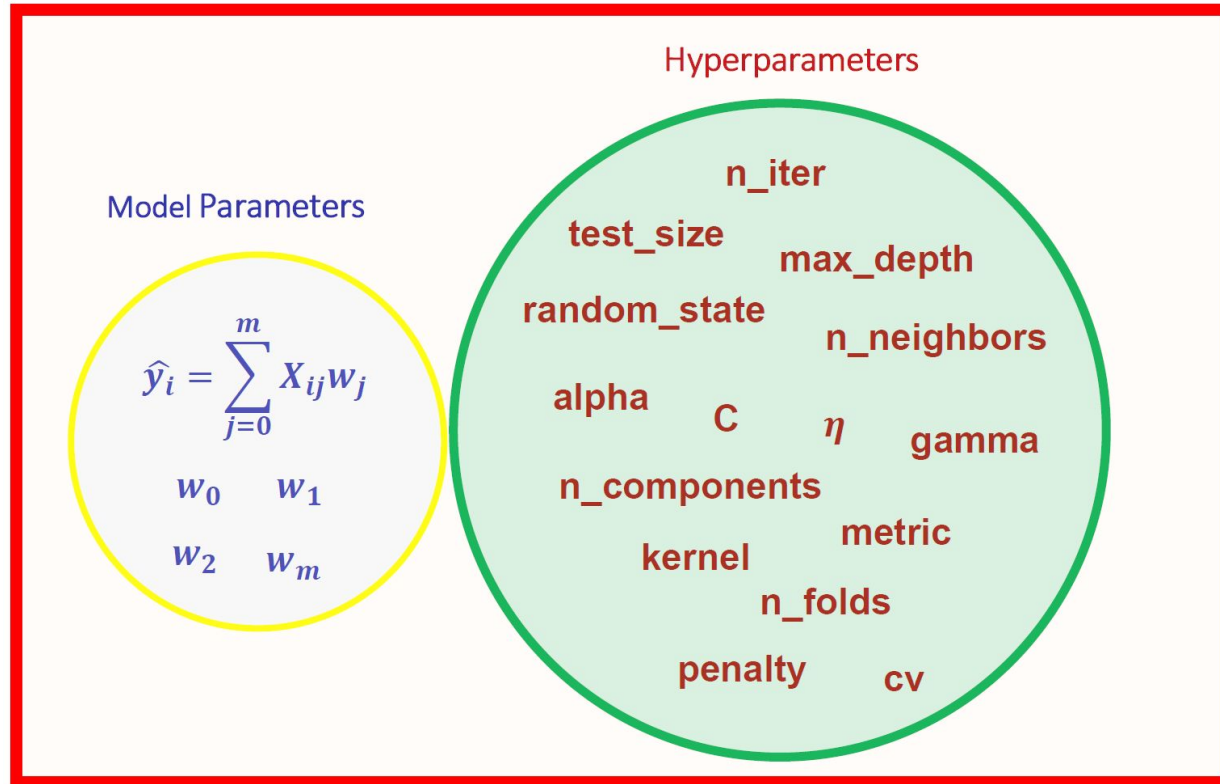
	Classif/regr	Gen/Discr	Param/Non
Discriminant analysis	Classif	Gen	Param
Naive Bayes classifier	Classif	Gen	Param
Tree-augmented Naive Bayes classifier	Classif	Gen	Param
Linear regression	Regr	Discrim	Param
Logistic regression	Classif	Discrim	Param
Sparse linear/ logistic regression	Both	Discrim	Param
Mixture of experts	Both	Discrim	Param
Multilayer perceptron (MLP)/ Neural network	Both	Discrim	Param
Conditional random field (CRF)	Classif	Discrim	Param
K nearest neighbor classifier	Classif	Gen	Non
(Infinite) Mixture Discriminant analysis	Classif	Gen	Non
Classification and regression trees (CART)	Both	Discrim	Non
Boosted model	Both	Discrim	Non
Sparse kernelized lin/logreg (SKLR)	Both	Discrim	Non
Relevance vector machine (RVM)	Both	Discrim	Non
Support vector machine (SVM)	Both	Discrim	Non
Gaussian processes (GP)	Both	Discrim	Non
Smoothing splines	Regr	Discrim	Non

Hyperparameters

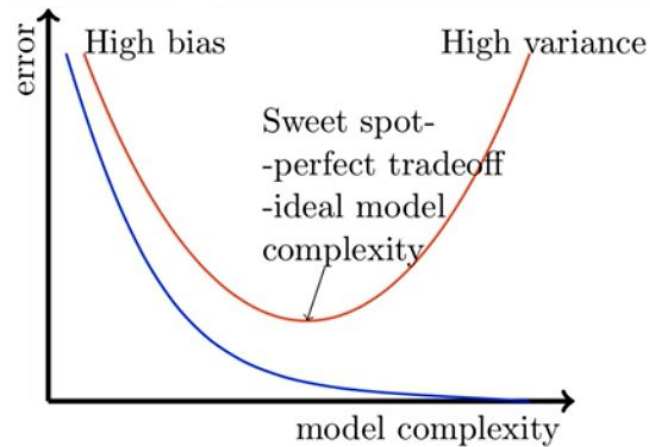
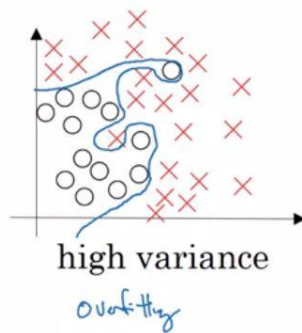
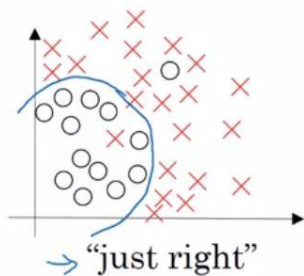
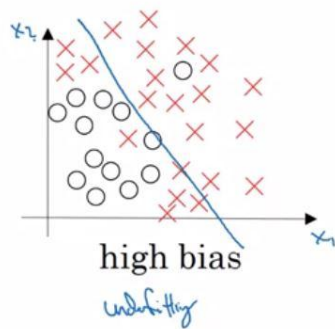
- A hyperparameter is a configuration variable that is external to the model.
- It is defined manually before the training of the model with the historical dataset.
- Its value cannot be evaluated from the datasets.
- It is not possible to know the best value of the hyperparameter.
- But we can use rules of thumb or select a value with trial and error for our system.

PARAMETER	HYPERPARAMETER
Estimated during the training with historical data.	Values are set beforehand.
It is a part of the model.	External to the model.
The estimated value is saved with the trained model.	Not a part of the trained model and hence the values are not saved.
Dependent on the dataset that the system is trained with.	Independent of the dataset.

Parameters Vs HyperParameters



Hyperparameter Tuning



$train_{err}$ (say, mean square error)

$test_{err}$ (say, mean square error)

Bias and Variance

Cat classification

$y=1$



$y=0$



Train set error:

1%

15%

15%

0.5%

Dev set error:

11%

16%

30%

1%

high variance

high bias

high bias
& high variance

low bias
low variance

Human: ~0%

Adding Regularization - To reduce Overfitting

$$\mathcal{L}_{train}(\theta) = \sum^N (y_i - \hat{f}(x_i))^2$$

$$\theta = [W_{111}, W_{112} + \dots + W_{Lnk}]$$

Normal Loss

When our training algorithm tries to minimize the regularized loss function, it will decrease both the original loss function and a regularization term.

By adding the regularized term, we're fooling the model such that it won't drive the training error to zero, which in turn reduces the complexity of the model. Therefore, L2 regularization helps reduce the overfitting of data.

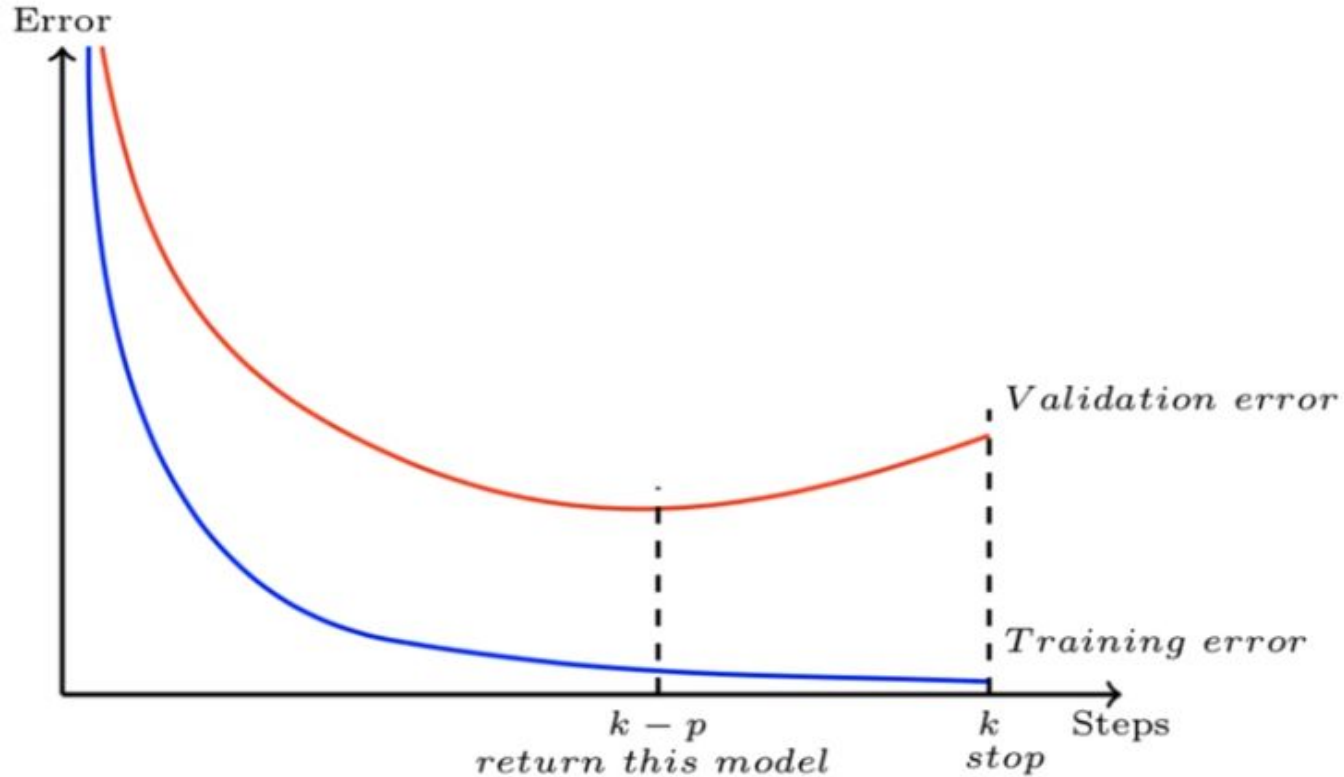
$$\min_{\theta} \mathcal{L}(\theta) = \mathcal{L}_{train}(\theta) + \Omega(\theta)$$

Regularized Loss Function

$$\begin{aligned}\Omega(\theta) &= ||\theta||_2^2 \\ &= W_{111}^2 + W_{112}^2 + \dots\end{aligned}$$

L2 regularization

Adding Early Stopping - To reduce Overfitting



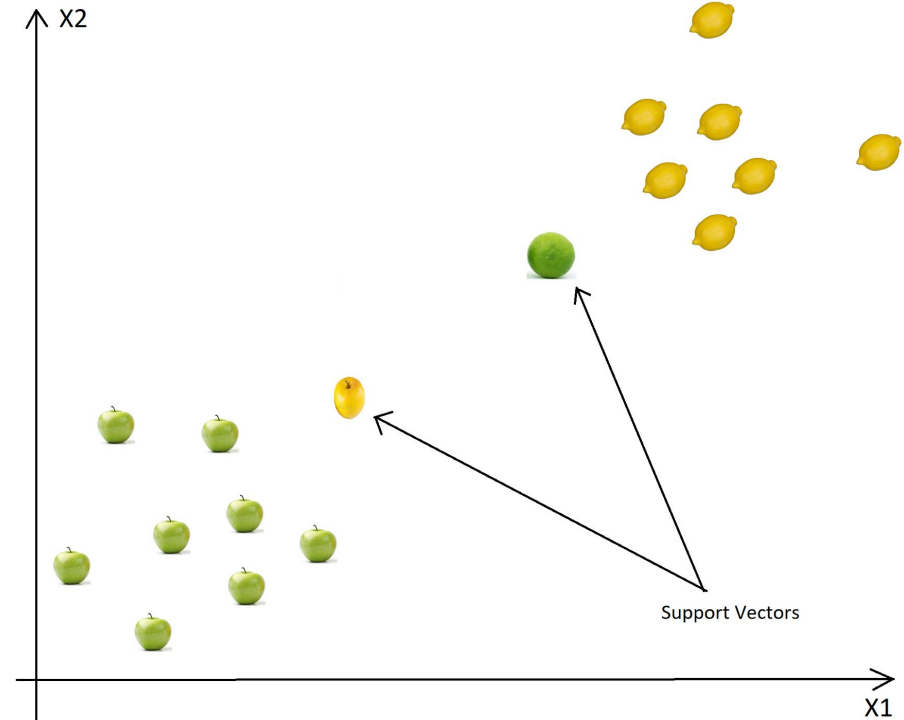
Support Vector Machine

The **Support Vector Machine** is a **supervised learning algorithm** mostly used for **classification** but it can be used also for **regression**.

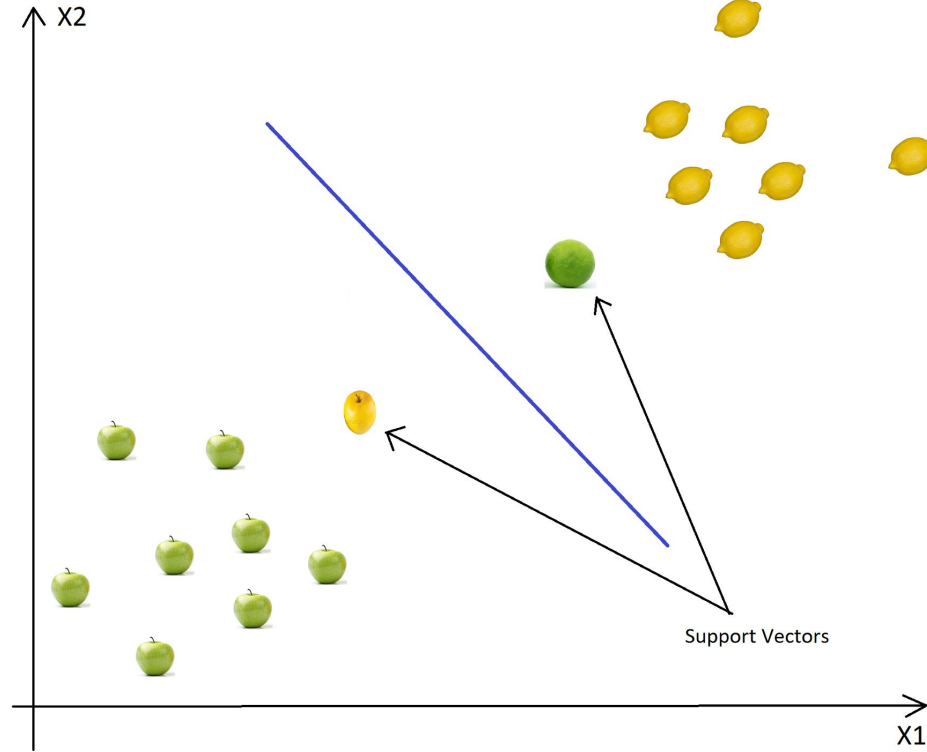
The main idea is that based on the labeled data (training data) the algorithm tries to find the **optimal hyperplane** which can be used to classify new data points.

SVM **finds the most similar examples** between classes. Those will be the support vectors.

So other algorithms learn the differences while **SVM learns similarities**.

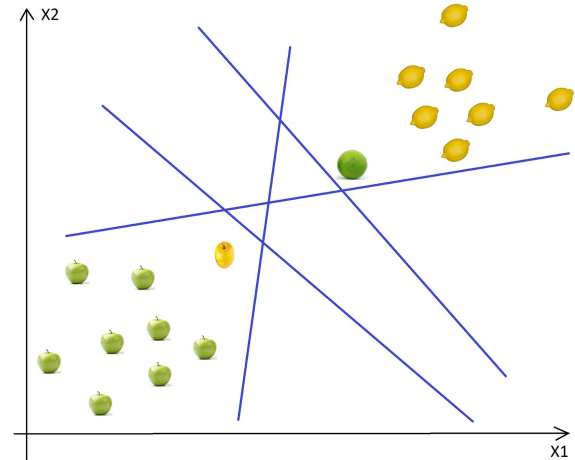


Hyperplanes

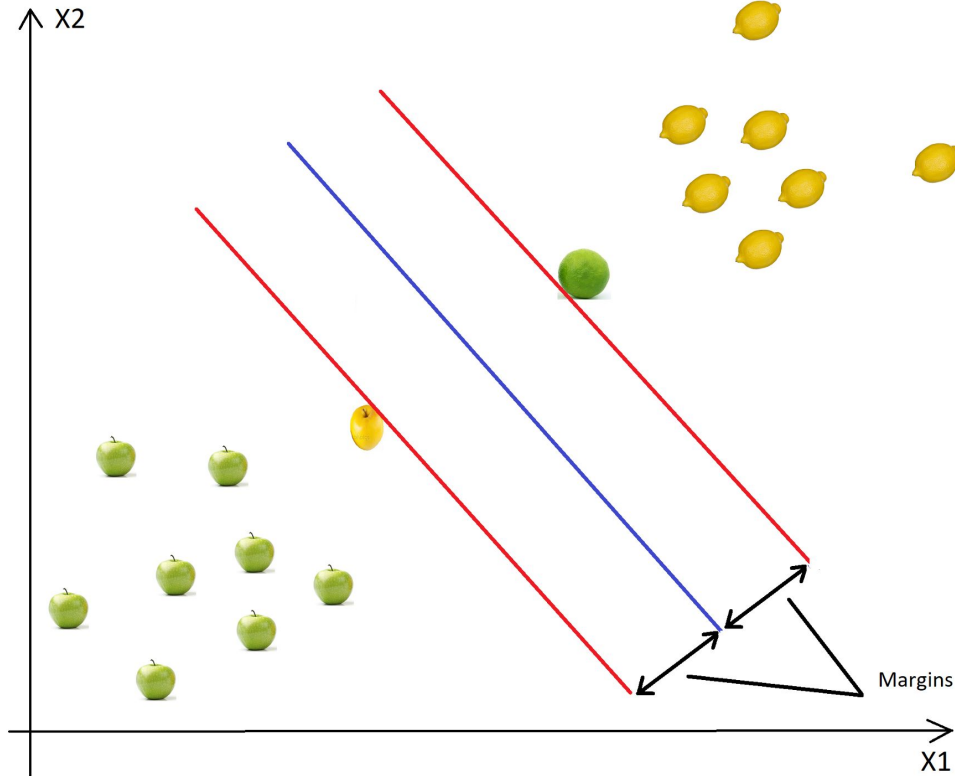


Based on these support vectors, the algorithm tries to find **the best hyperplane that separates the classes**.

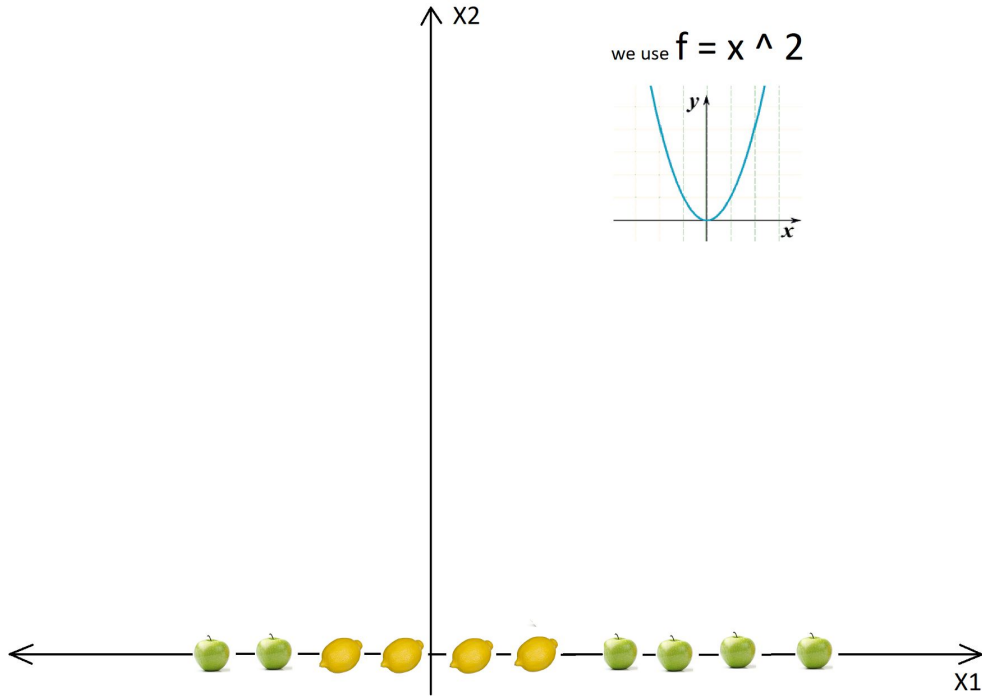
In 2D the hyperplane is a line, so it would look like this:



Find optimal Margins



Support vectors are data points that **defines the position and the margin of the hyperplane**. We call them “**support**” vectors,

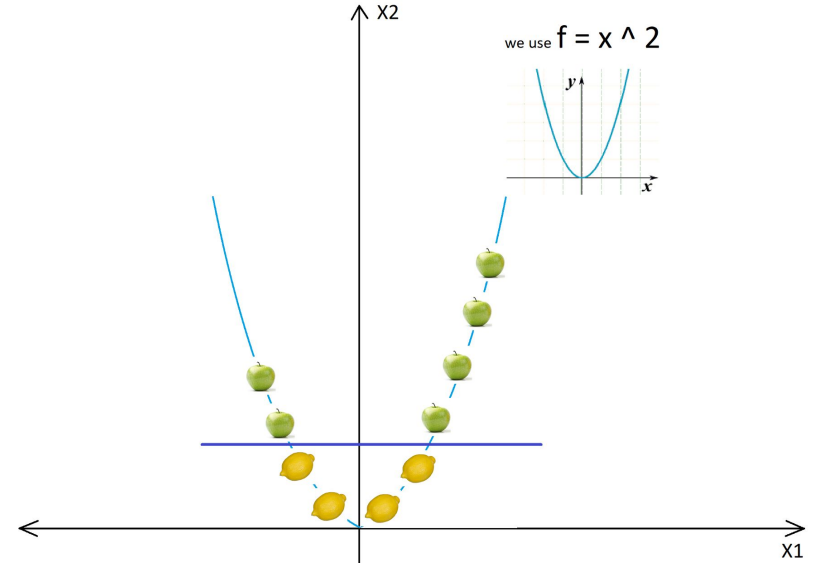
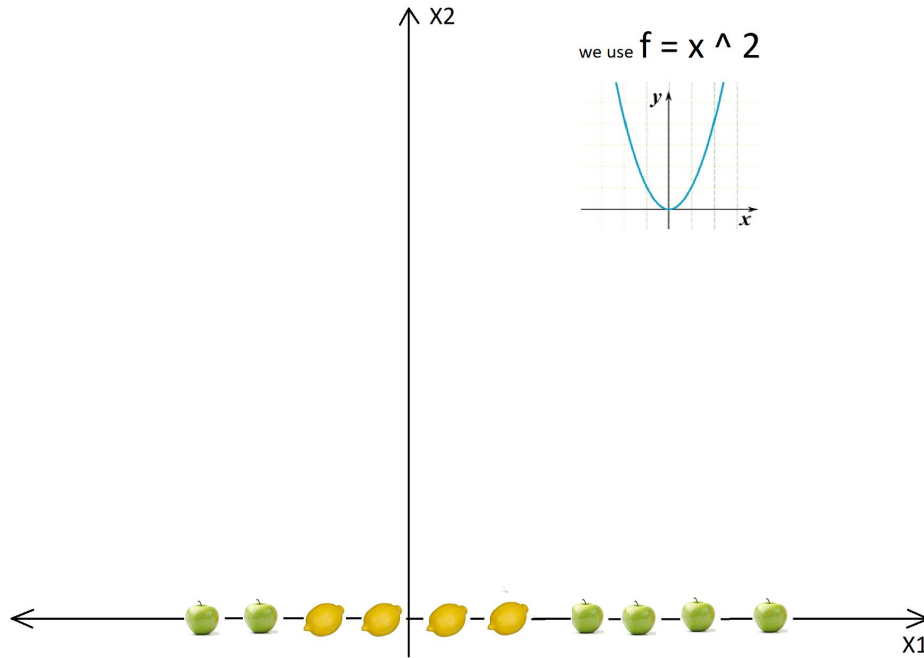


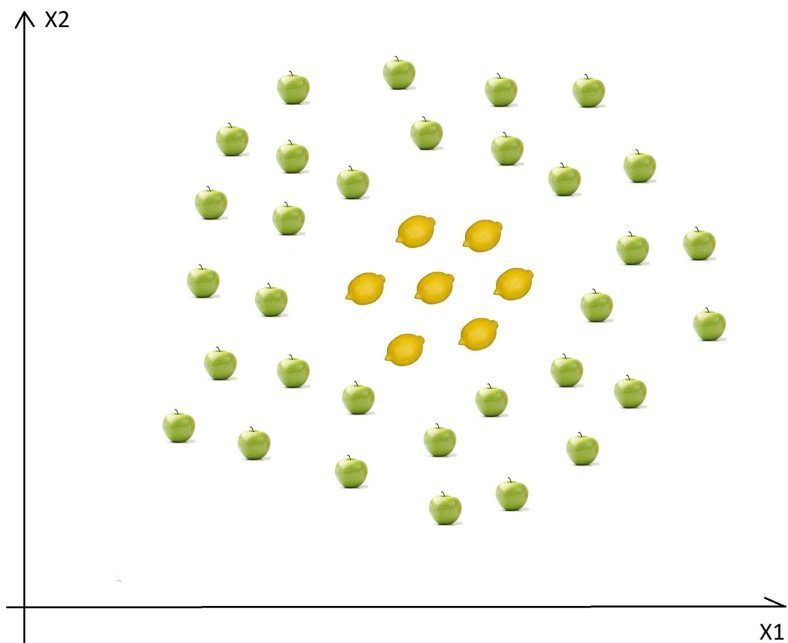
SVM - Kernels

In this case **we cannot find a straight line** to separate apples from lemons.

So how can we solve this problem. We will use the **Kernel Trick!**

Mapping from 1D to 2D



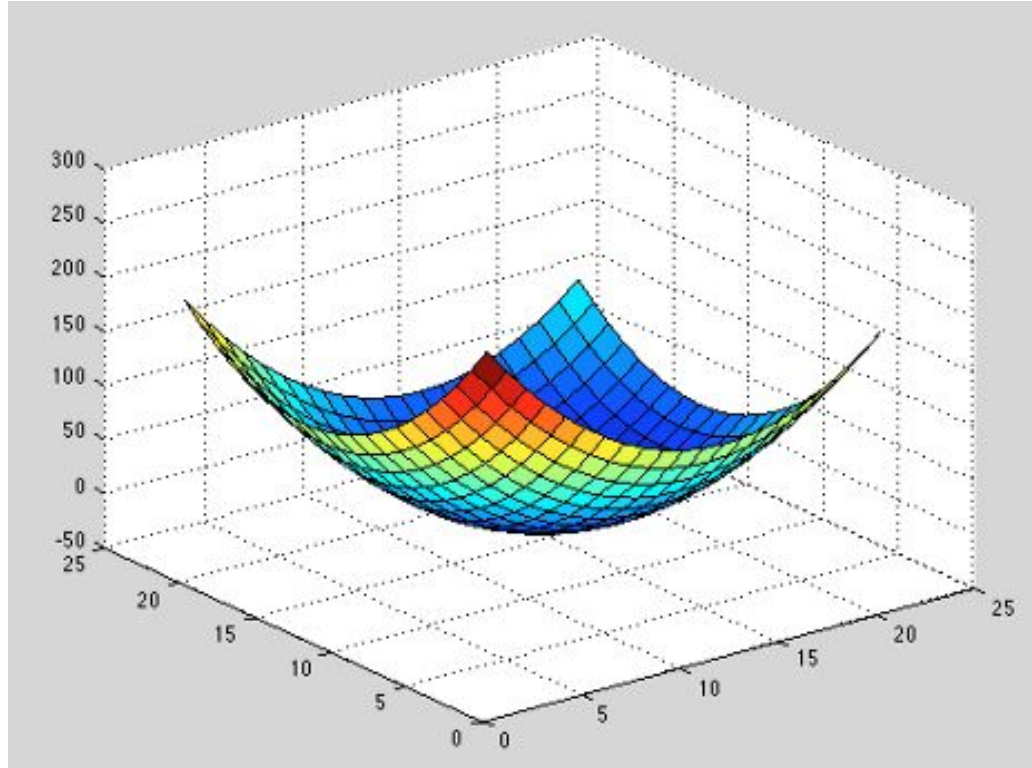


SVM For Non Linear Dataset

In this case also **we cannot find a straight line** to separate apples from lemons.

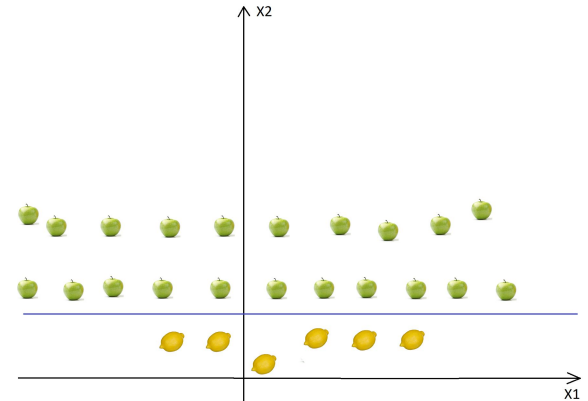
So apply **Kernel Trick!**

Mapping from 2D to 3D



The basic idea is that when a data set is inseparable in the current dimensions, **add another dimension**, maybe that way the data will be separable.

in the new dimension the points are organized using this formula $\mathbf{x1^2 + x2^2}$.



Steps in SVM

1. select hyperplanes which separates the data with no points between them
2. maximize their distance (the margin)
3. the average line will be the decision boundary

But finding the best margin, the optimization problem is not trivial

why not Gradient descent??

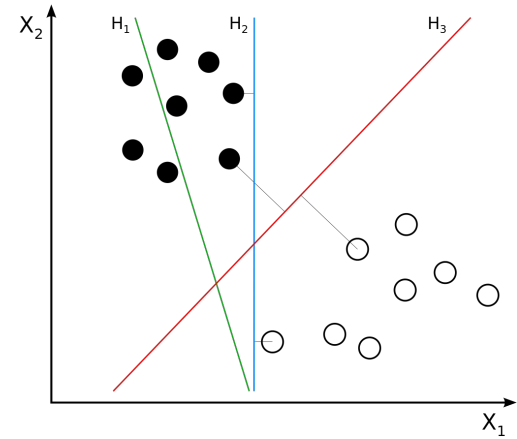
SVM optimization problem is a case of constrained optimization problem, and it is always preferred to use dual optimization algorithm

To solve the optimization problem, we use the Lagrange Multipliers. [Check PDF]

Multi class classification with SVM ?

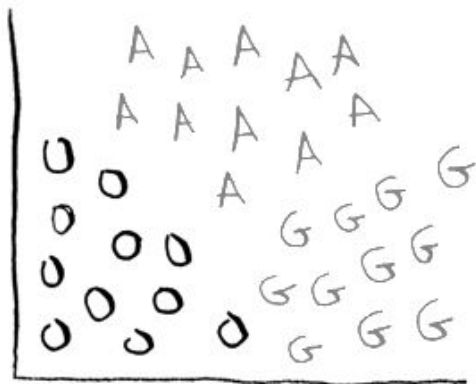
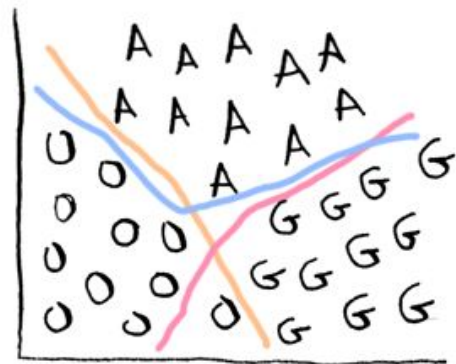


SVMs are maximum-margin classifiers, which means that they attempt to generate a decision boundary that is equidistant from the two classes of data.



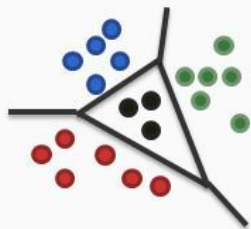


(O = orange, G = grapefruit, A = apple).
You need multiple decision boundaries:



We need 3 SVM's to handle this problem

One-vs-all may not always work

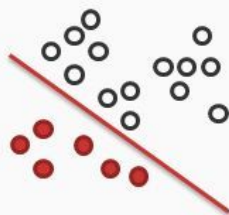


Black points are not separable with a single binary classifier

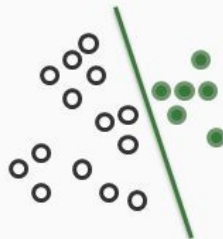
The decomposition will not work for these cases!



$w_{\text{blue}}^T x > 0$
for **blue**
inputs



$w_{\text{red}}^T x > 0$
for **red**
inputs



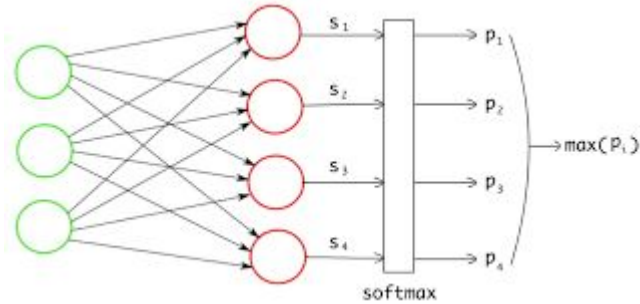
$w_{\text{green}}^T x > 0$
for **green**
inputs



???

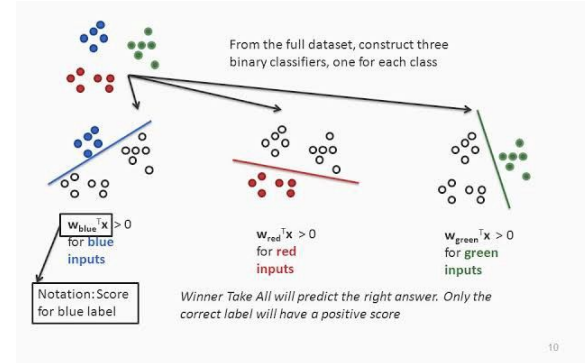
ANN is a parametric classifier

ANNs can handle multi-class problems by producing probabilities for each class.



SVM is a non-parametric classifier

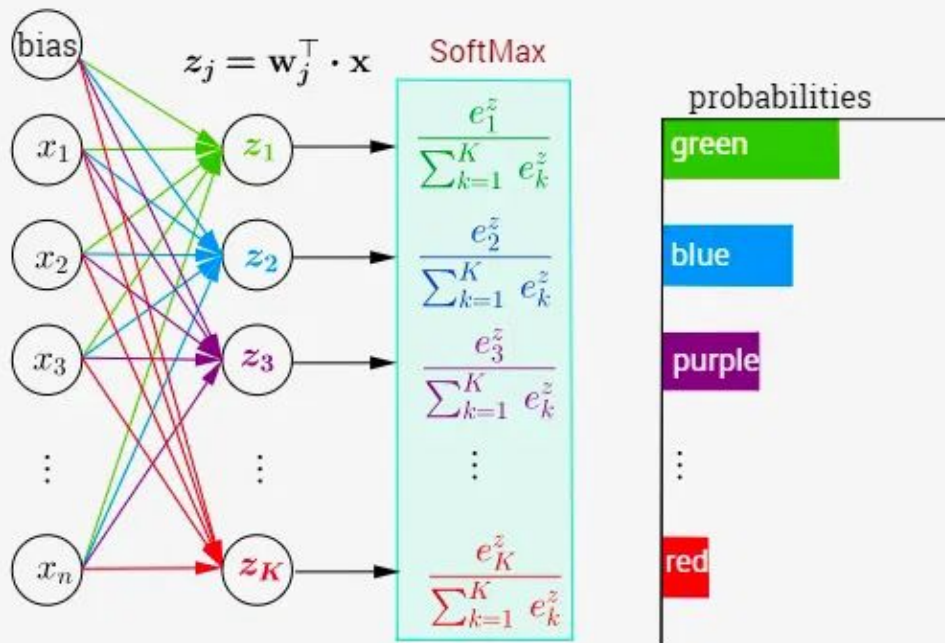
SVMs handle multi class problems using one-vs-one classifiers



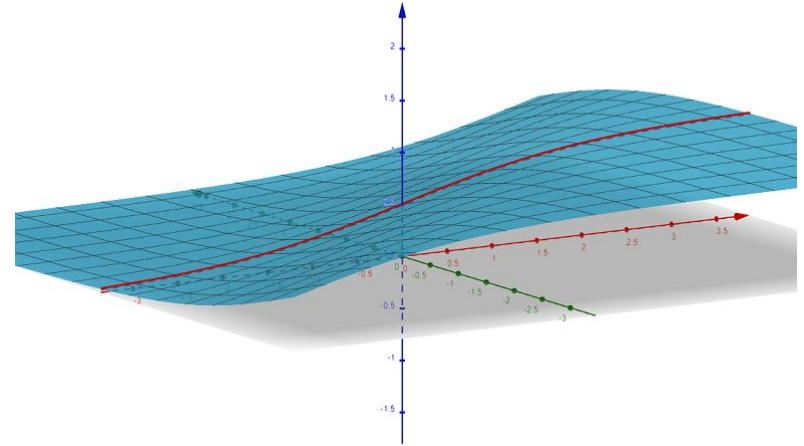
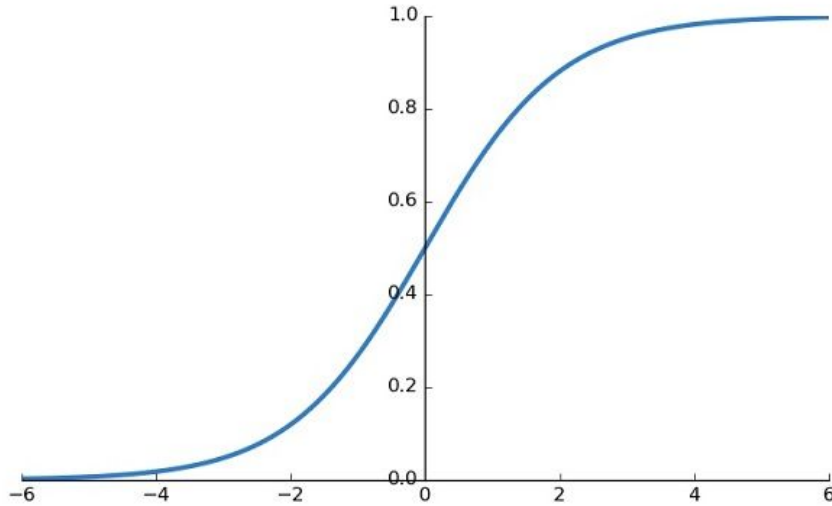
Neural network with Multi class outputs

Multi-Class Classification with NN and SoftMax Function

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$



Sigmoid Vs Softmax activation function



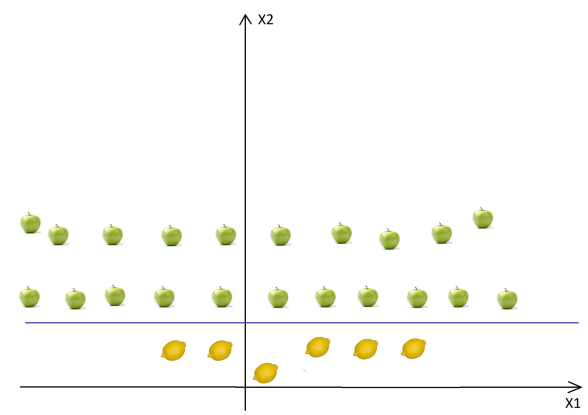
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \text{ for } i = 1, \dots, K \text{ and } \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K$$

<https://www.geogebra.org/3d/r3rsewaf>

SVM for Anomaly Detection

THE BEST BINARY CLASSIFIER

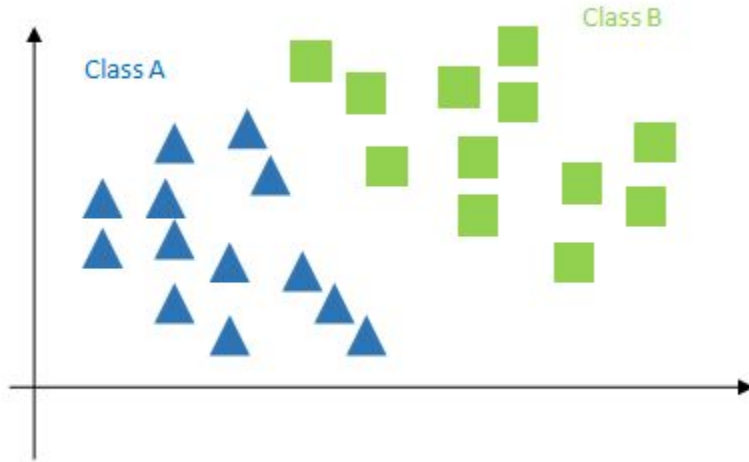


SVM generally do not suffer condition of overfitting and performs well when there is a clear indication of separation between classes.

SVM can be used when total no of samples is less than the no of dimensions and performs well in terms of memory.

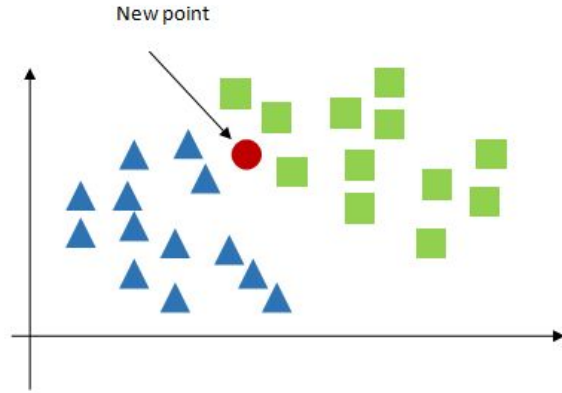
SVM Classifier is dependent ideally only on a subset of points (support vectors), while maximizing distance between closest points of two classes (Margin), So We do not need to take care and take into account all the points but taking only subset of points become helpful

K Nearest Neighbours



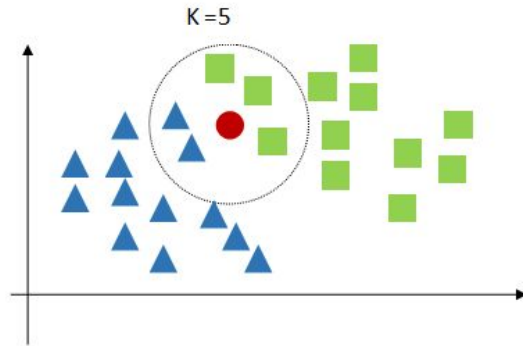
KNN is supervised machine learning algorithm which can be used for both classification and regression problems.

In the case of classification K_{nearest} neighbor can be used for both binary and multi-class classifications.



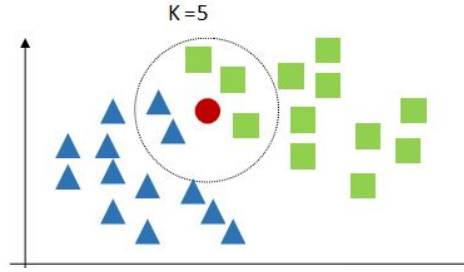
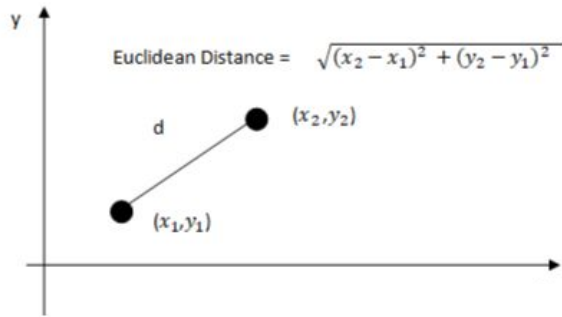
What is K in K - Nearest Neighbours ?

Now we add a new point to the dataset, we need to determine which class this new point belongs to.

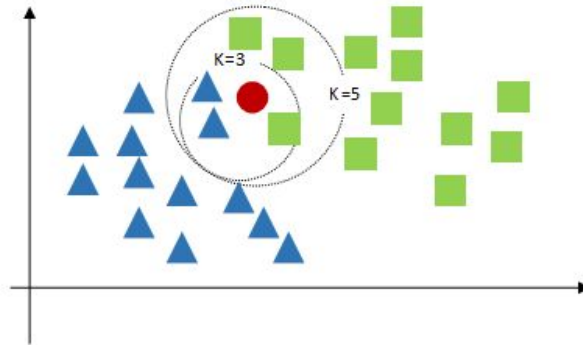


We need to choose the number of neighbors (K) from both the classes

How to Identify the nearest neighbours ??



The nearest neighbors are identified based on the Euclidean distance of the new point to those neighbors.

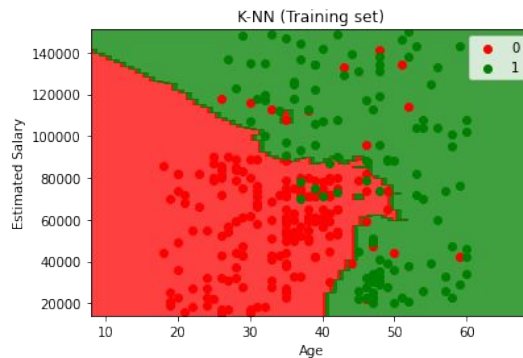


If K is too small, noise would have higher impact on the outcome.

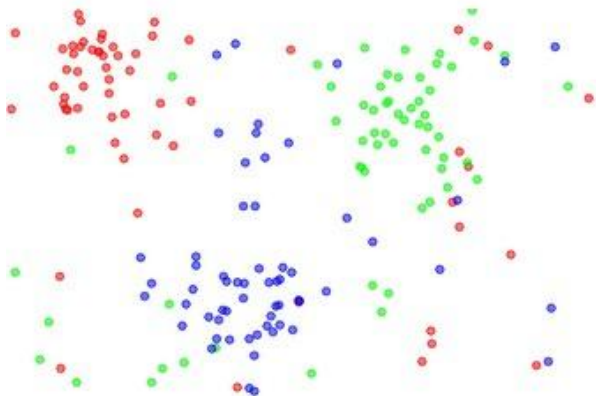
If k is too big computation would be more complex and lengthy.

Value of K can be selected as the square root of number of training samples

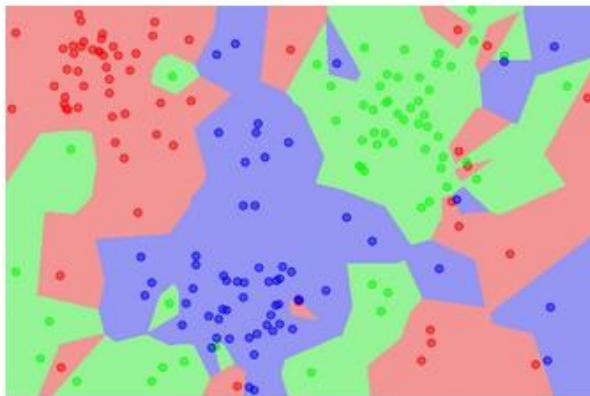
Visualising KNN



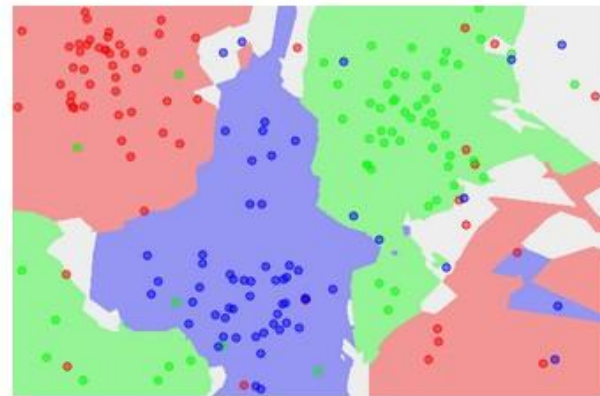
the data



NN classifier



5-NN classifier



K-NN Classifier with Sklearn

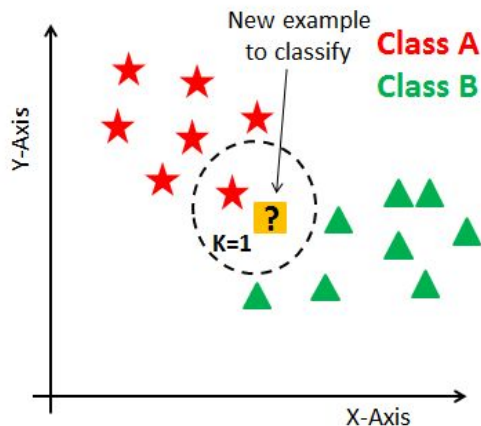
Classification with finding Best K - value

<https://colab.research.google.com/drive/15kNX6nzeMUbawb0IBbBeWVhfu1nHqhZP?usp=sharing>

NN vs KNN - Instance Based Learning

Given query instance x_q (new data point),

Locate nearest training example x_n and estimate



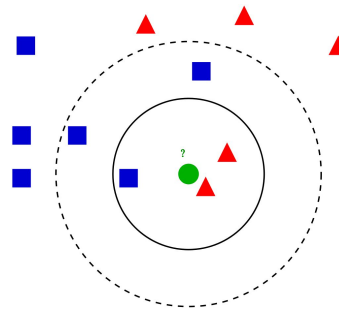
$$f(x_q) \leftarrow f(x_n)$$

distance function

Given x_q ,

Take vote among its k nearest nbrs
(if discrete-valued)

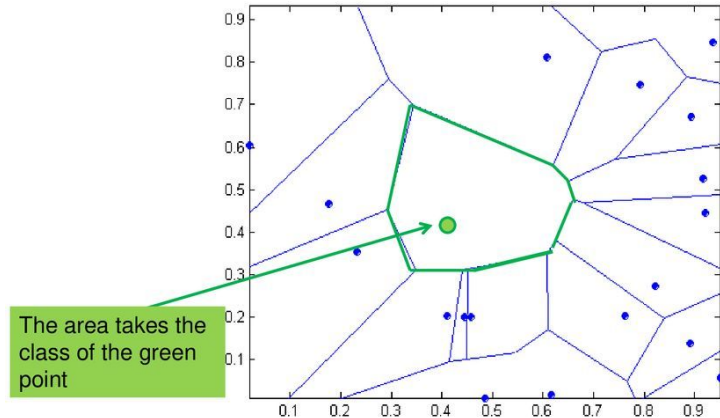
Take mean of distance values of k nearest nbrs
(if real-valued)



$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^k f(x_i)}{k}$$

Voronoi Problem with KNN

Voronoi Diagram defines the classification boundary



Based on every new data point,

New decision boundaries are created, so it's not very generalised way of classification.

This causes multiple distances boundaries that looks like a polyhedron

Need for weighted KNN

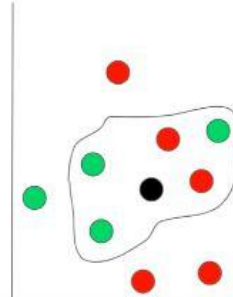
Spurious or less relevant points need to be downweighted

Options for determining the class from nearest neighbor list

- 1. majority vote of class labels among the k-nearest neighbors
- 2. Weight the votes according to distance
 - example: weight factor $w = (1 / d^2)$

Distance Increases
Weight Decreases

Weighted KNN makes model more generalisable



Point	Label	Distance	Weight
(x1,y1)	Red	0.2	5
(x2,y2)	Red	0.5	2
(x3,y3)	Green	0.7	1.4
(x4,y4)	Green	1.2	0.8
(x5,y5)	Green	1.5	0.6

Calculate Weight

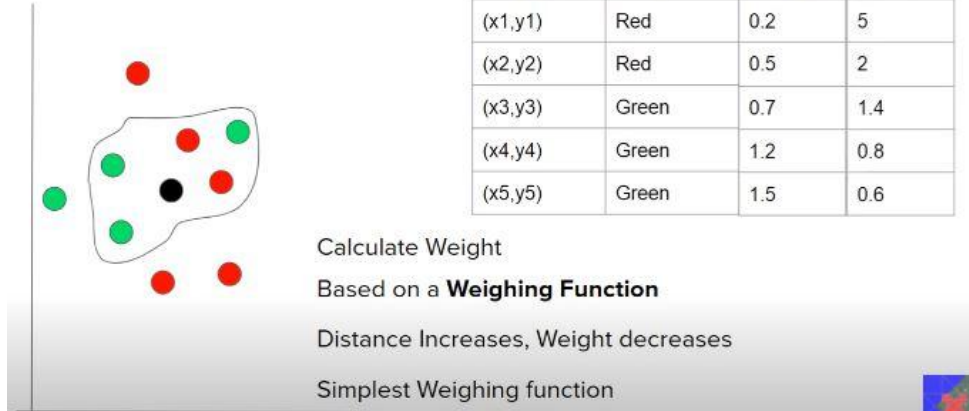
Based on a **Weighing Function**

Distance Increases, Weight decreases

Simplest Weighing function



Weighted KNN




When Distance alone taken into consideration

New Data point is classified as **Green Class**
based on majority ratio

Red -> only 2 neighbours out of 5
Green -> 3 neighbours out of 5

Above method is not generalised at all

But with weighted sum of distances we get
new data point to be classified as **Red Class**

 $1.4 + 0.8 + 0.6 = 2.8$

 $5 + 2 = 7$

Case Study

Outlier Detection using KNN and SVM

<https://colab.research.google.com/drive/1VlzC8dhErJMgNpv6UK2p9vTIhQVE6KCC?usp=sharing>

Understanding Complexity of ML Models

Time complexity can be seen as the measure of how fast or slow an algorithm will perform for the input size.

Time complexity is always given with respect to some input size (say n).

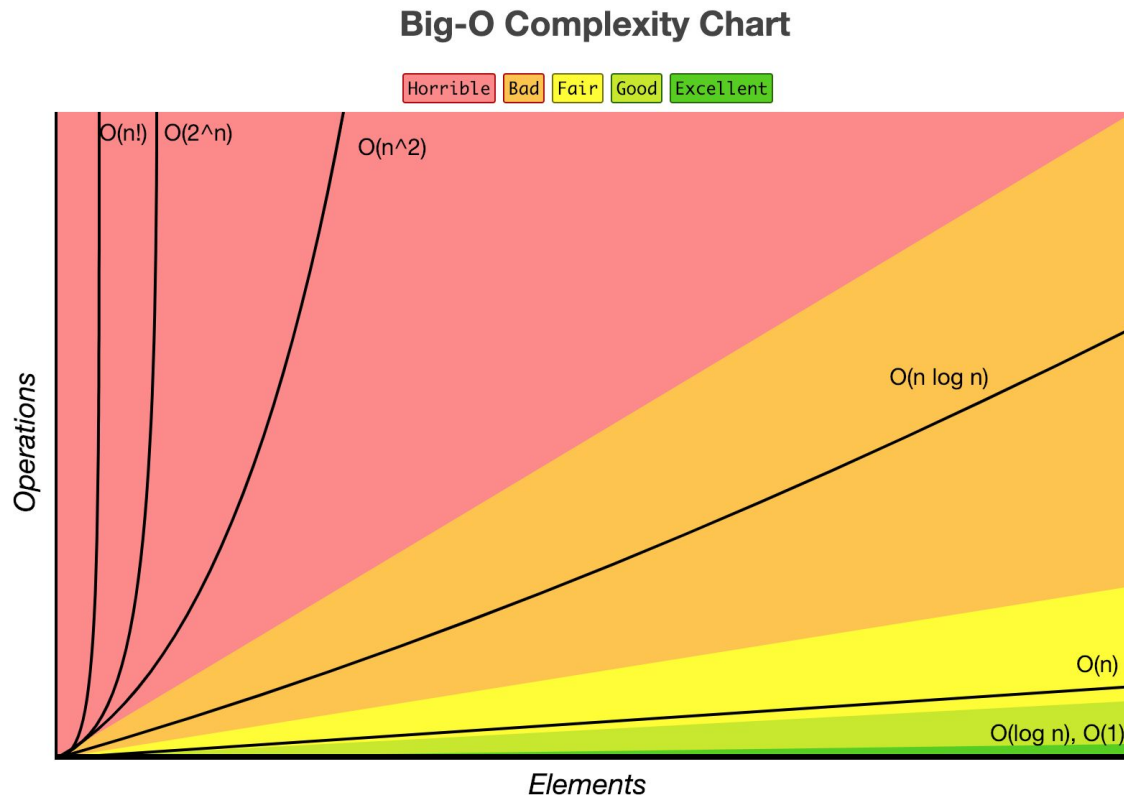
Space complexity can be seen as the amount of extra memory you require to execute your algorithm.

Like the time complexity, it is also given with respect to some input size (n).

Understanding Complexity of ML Models

The complexity of an algorithm/model is often expressed using the

Big O Notation, which defines an upper bound of an algorithm, it bounds a function only from above.



Complexity Analysis of KNN

The complexity of K Nearest Neighbors to find the k closest neighbor

Train Time Complexity = $O(knd)$

Loops through every training observation and computes the distance d between the training set observation and new observation.

Time is linear with respect to the number of instances (n) and dimensions (d).

Space Complexity = $O(nd)$

K Nearest Neighbors store the data.

Testing takes longer because you have to compare every test instance to the whole training data.

UnSupervised Learning

Unsupervised learning is a type of machine **learning** in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision.

There are three cases in Unsupervised Learning

Clustering, Dimensionality Reduction, and Association Rule

**K-means
Clustering,

Hierarchical
Clustering**

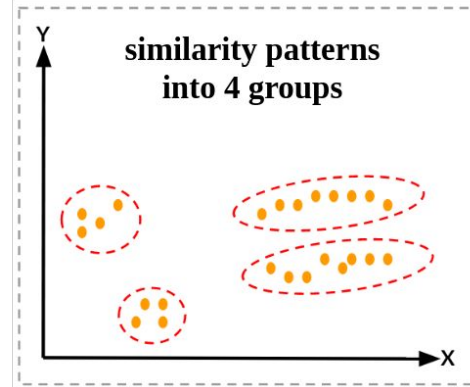
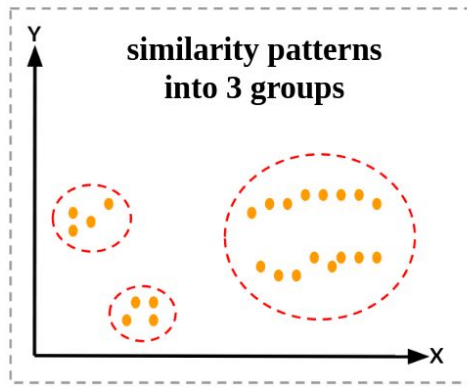
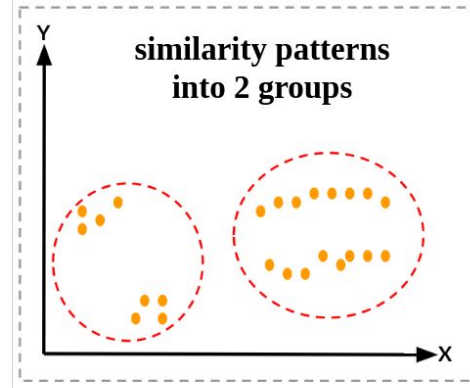
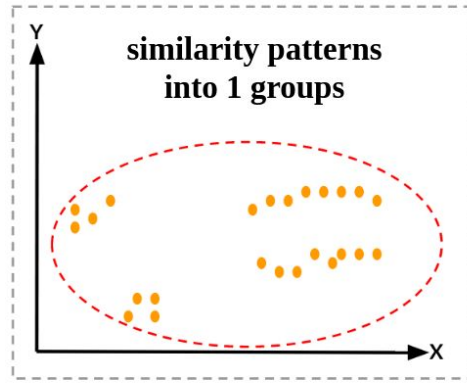
**AutoEncoders,

PCA,

LDA**

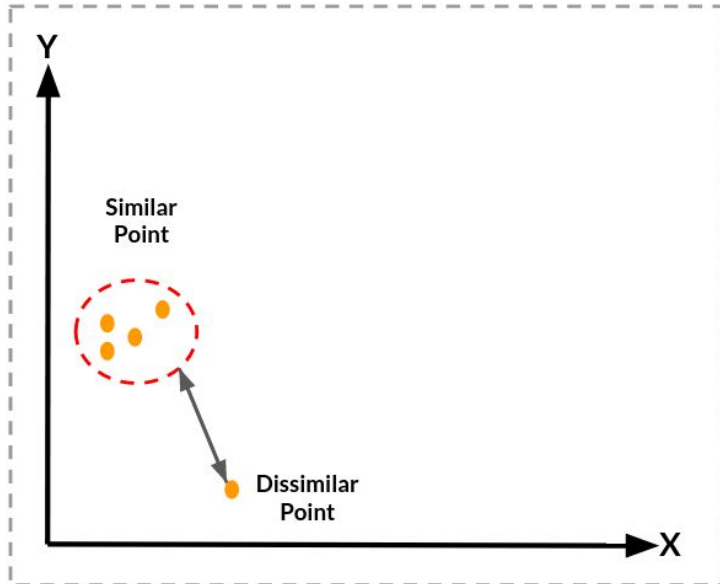
Apriori algorithm

Clustering



How do we know a point has the same group as another point?

Based on similarity patterns

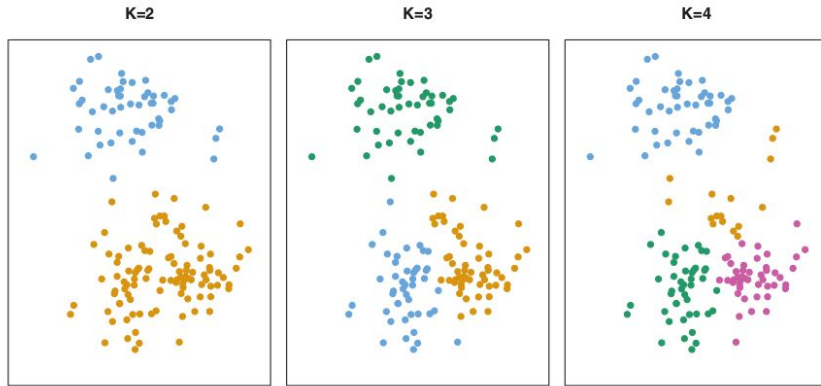


The similarity between each points are calculated based on the distance between the points

- Euclidean Distance
- Manhattan Distance
- Minkowski Distance
- Hamming Distance

K-means Clustering

What is K-means ?

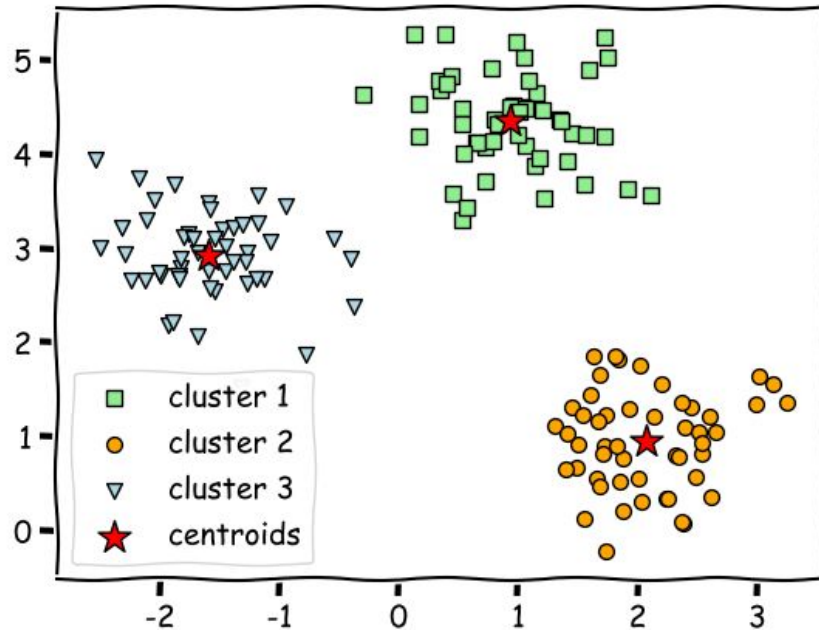


K-means clustering is a simple and elegant approach for **partitioning a data set into K distinct, non-overlapping clusters.**

To perform K-means clustering, we must first specify the desired number of clusters K ;

then the K-means algorithm will assign each observation to exactly one of the K clusters

How to know the pattern similarity with K-means clustering method?



The K-means algorithm aims to choose centroid that minimise the **inertia**, or **within-cluster sum-of-squares criterion**

$$\text{objective function} \leftarrow J = \sum_{j=1}^k \sum_{i=1}^n \|x_i^{(j)} - c_j\|^2$$

Diagram illustrating the objective function J for K-means clustering:

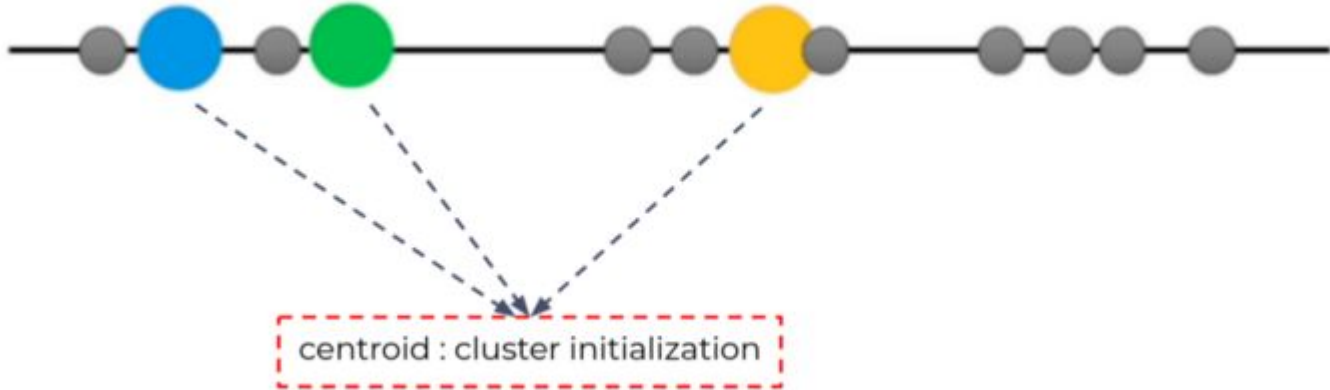
- k : number of clusters
- n : number of cases
- i : case i
- j : centroid for cluster j
- $\|x_i^{(j)} - c_j\|^2$: Distance function (squared distance between case i and centroid c_j)

K-means clustering



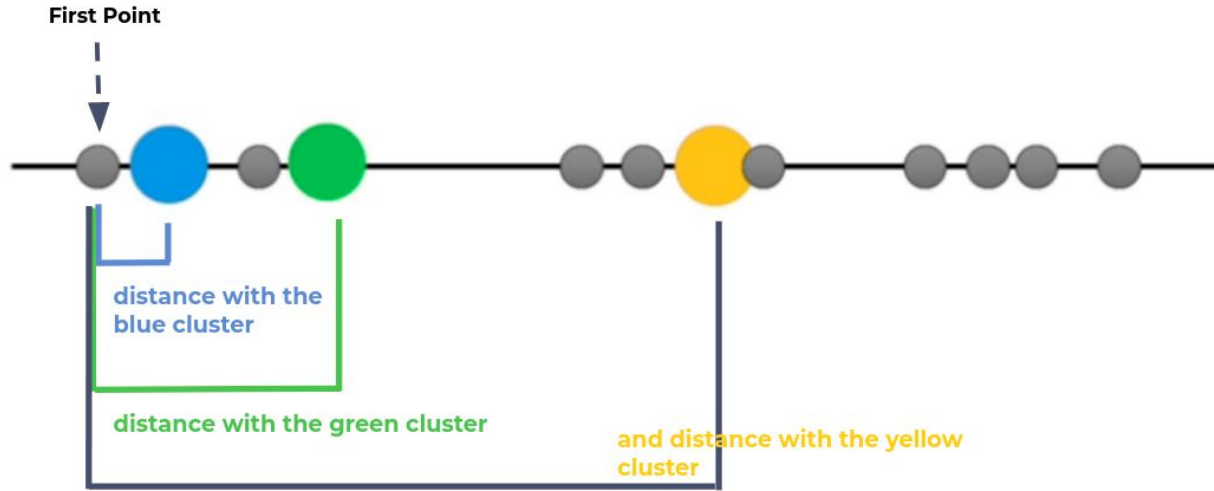
Step 1. Determine the value “K”, the value “K” represents the number of clusters.

we'll select $K=3$. So, we want to identify 3 clusters.

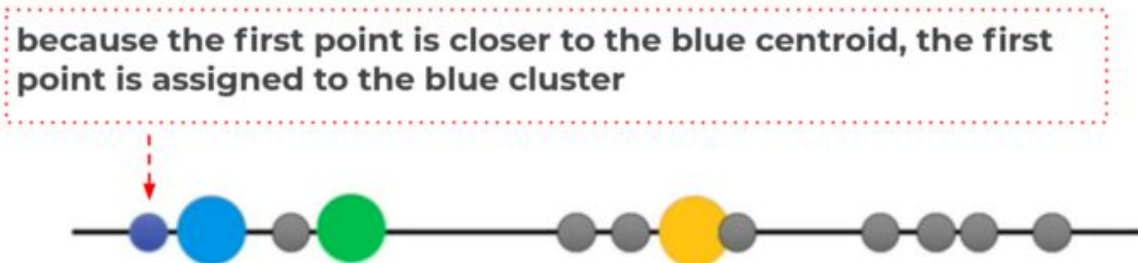


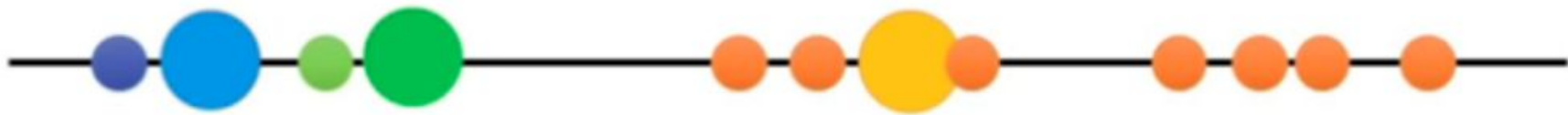
Step 2. Randomly select 3 distinct centroid (new data points as cluster initialization)

Step 3. Measure the distance ([euclidean distance](#)) between each point and the centroid



Step 4. Assign the each point to the nearest cluster





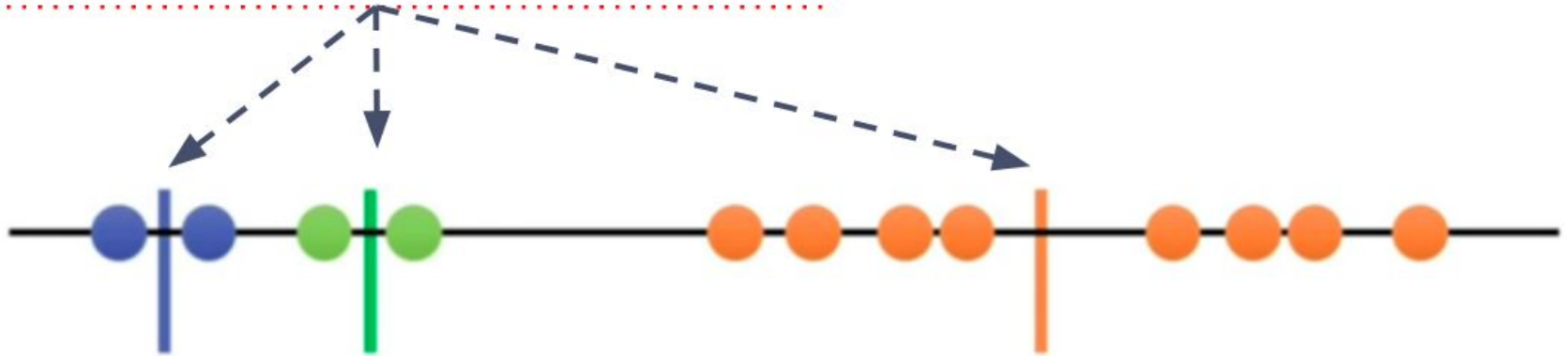
**Assign the each point
to the nearest cluster**



Do the same treatment for the other unlabeled point, until we get this

Step 5. Calculate the mean of each cluster as new centroid

Calculate the mean of each cluster

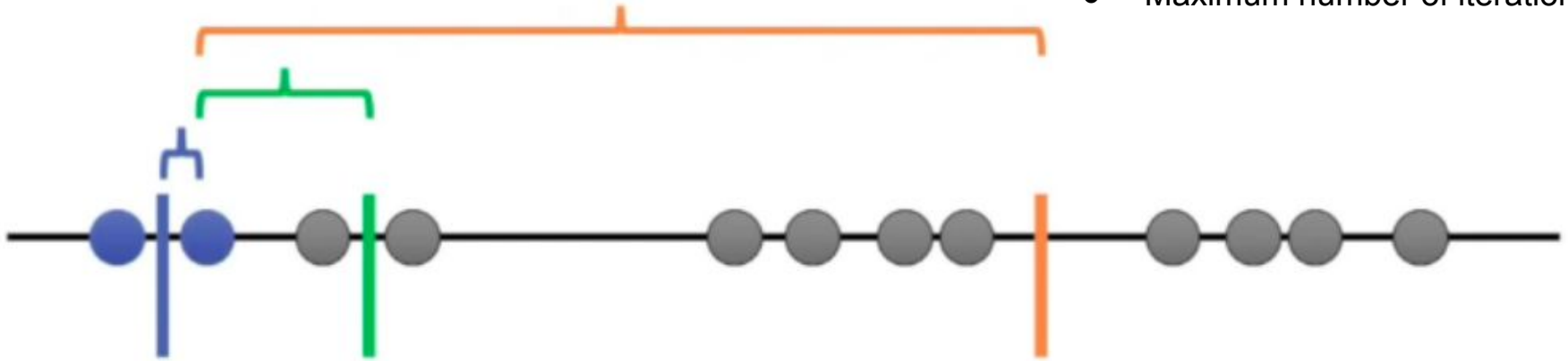


Update the centroid with mean of each cluster

Step 6. Repeat step 3–5 with the new center of cluster

Repeat until stop:

- Convergence.
(No further changes)
- Maximum number of iterations.



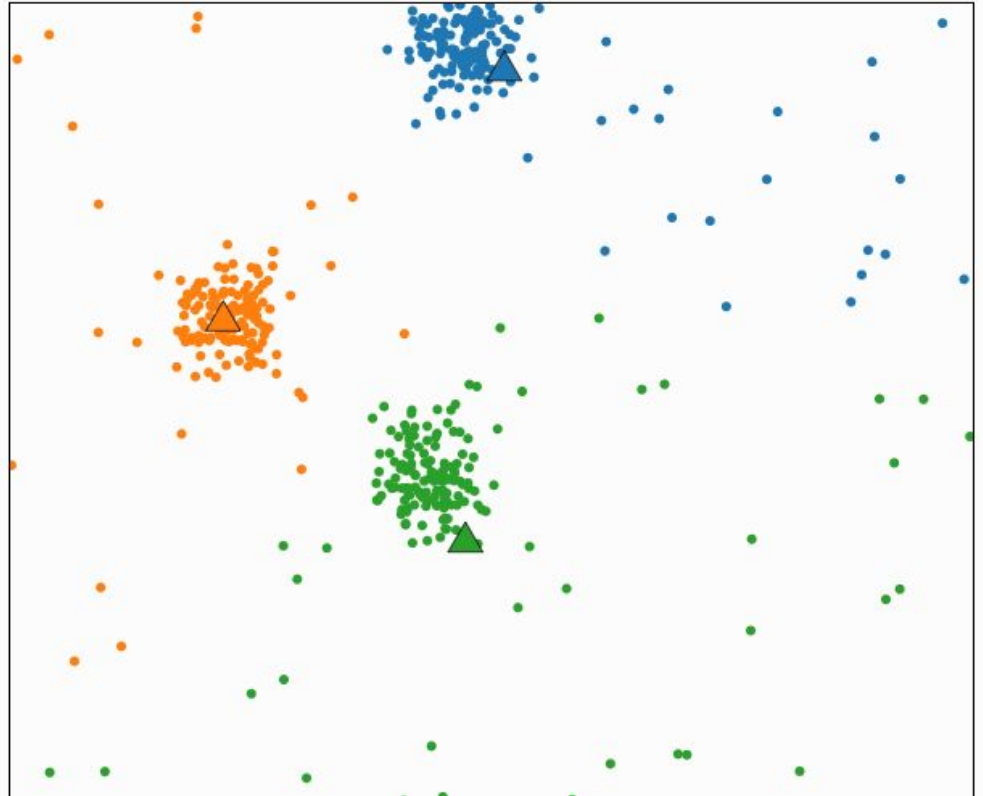
Final Clustering



K-Means Clustering

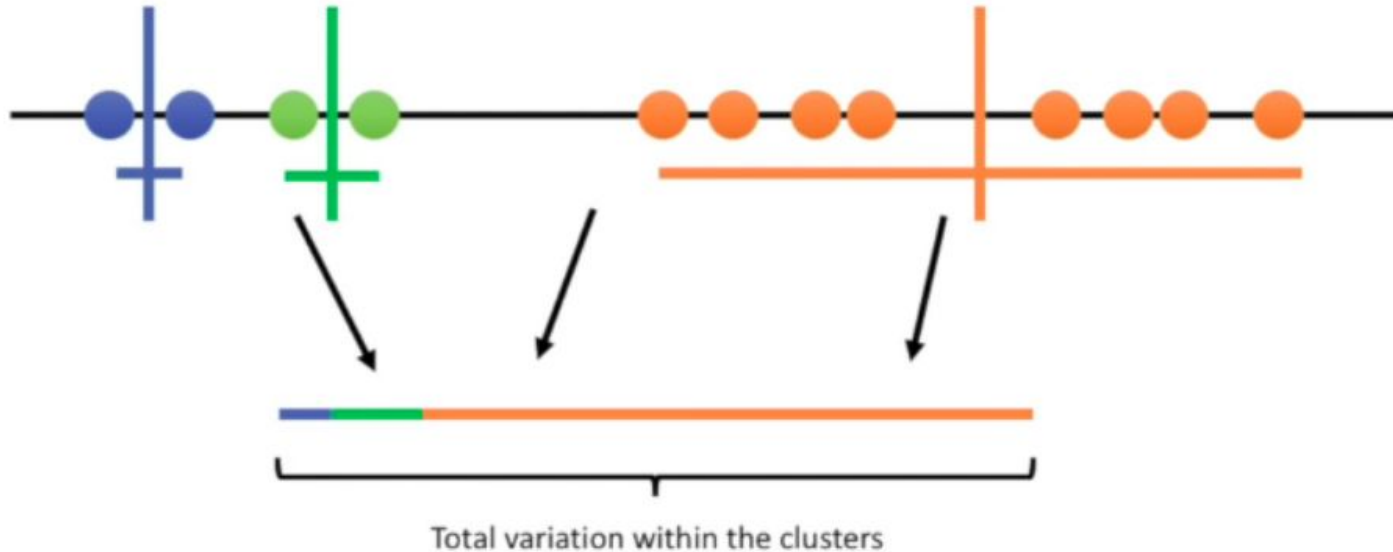
Final 3 Clusters Looks like this

Has this process been
completed?
of course not



How to assess the results of K-means clustering?

Step 7. Calculate the variance of each cluster



Attempt 2 with different random centroid

Step 8. Repeat step 2–7 until get the lowest sum of variance

Step 2



Step 3-4

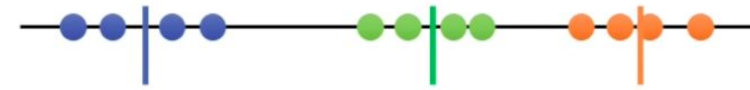


Step 5

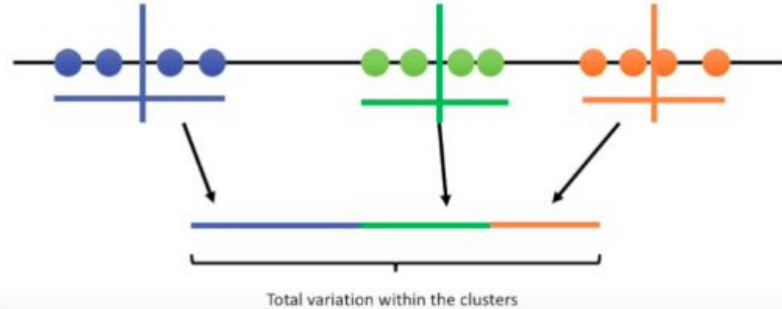


Step 6

Iterate to step 2
until the cluster no
longer change...

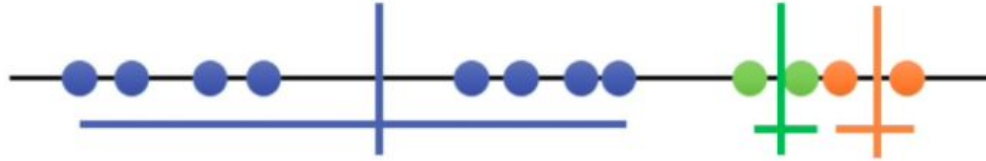


Step 7



Attempt 3 with different random centroid

Step 7



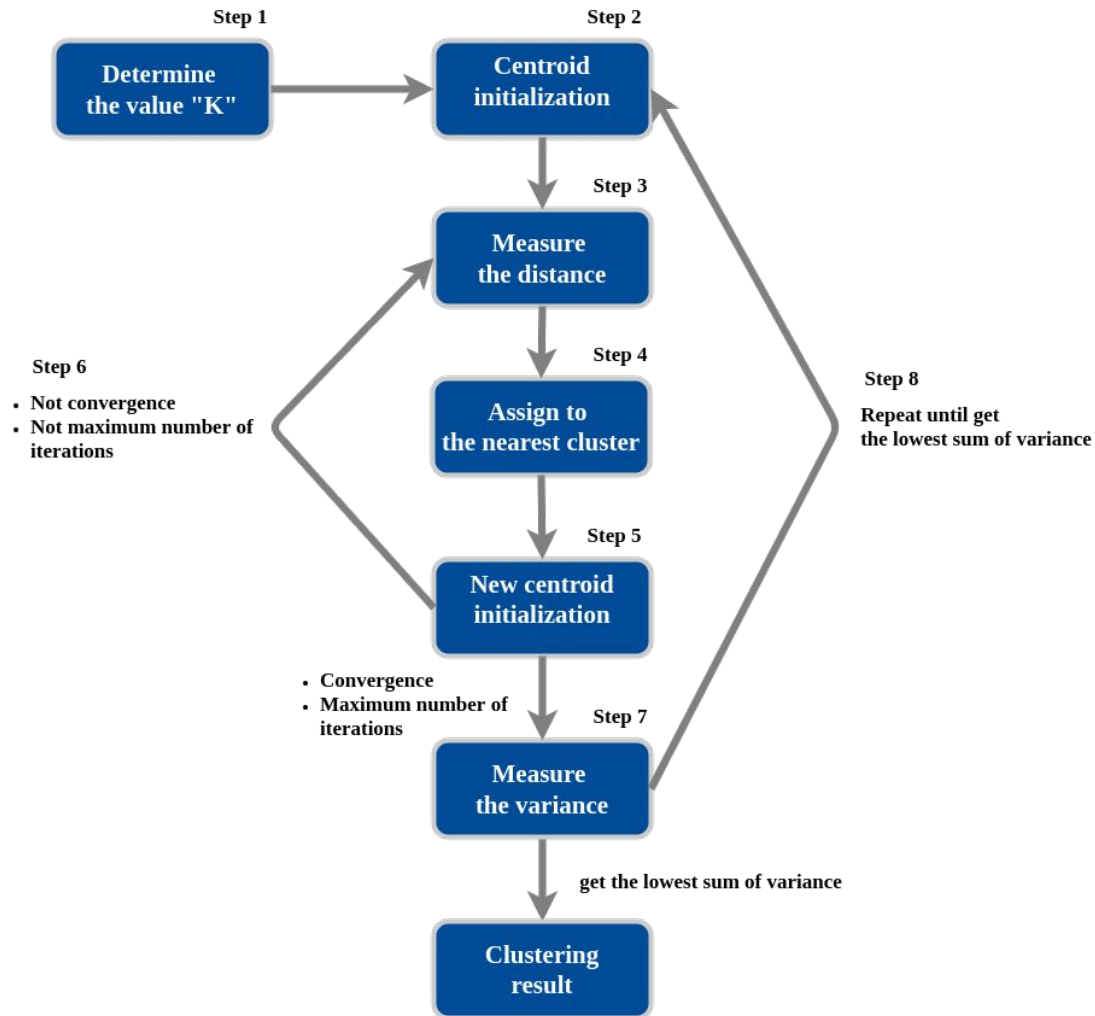
1st cluster attempt: 

2nd cluster attempt: 

The winner!!

3rd cluster attempt: 

K -means Clustering Overview



K-means Clustering

The approach kmeans follows to solve the problem is called **Expectation-Maximization**.

The E-step is assigning the data points to the closest cluster.

The M-step is computing the centroid of each cluster.

$$J = \sum_{i=1}^m \sum_{k=1}^K w_{ik} \|x^i - \mu_k\|^2$$

The objective function is:

where $w_{ik}=1$ for data point x_i if it belongs to cluster k ; otherwise, $w_{ik}=0$.

Also, μ_k is the centroid of x_i 's cluster.

It's a minimization problem of two parts.

We first minimize J w.r.t. w_{ik} and treat μ_k fixed. Differentiate J w.r.t. w_{ik} first and update cluster assignments (*E-step*).

$$\begin{aligned}\frac{\partial J}{\partial w_{ik}} &= \sum_{i=1}^m \sum_{k=1}^K \|x^i - \mu_k\|^2 \\ \Rightarrow w_{ik} &= \begin{cases} 1 & \text{if } k = \operatorname{argmin}_j \|x^i - \mu_j\|^2 \\ 0 & \text{otherwise.} \end{cases}\end{aligned}$$

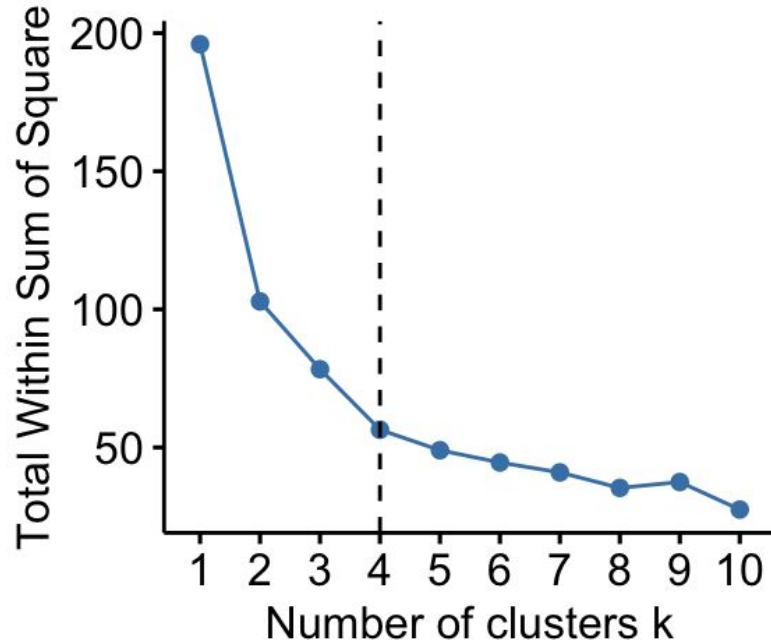
Then we minimize J w.r.t. μ_k and treat w_{ik} fixed. Differentiate J w.r.t. μ_k and recompute the centroids after the cluster assignments from previous step (*M-step*).

$$\begin{aligned}\frac{\partial J}{\partial \mu_k} &= 2 \sum_{i=1}^m w_{ik} (x^i - \mu_k) = 0 \\ \Rightarrow \mu_k &= \frac{\sum_{i=1}^m w_{ik} x^i}{\sum_{i=1}^m w_{ik}}\end{aligned}$$

How to choose correct K?

Optimal number of clusters

Elbow method



The basic idea behind this method is that it plots the various values of cost with changing k . The point where this distortion declines the most is the **elbow point, which works as an optimal value of k .**

K-means Clustering

Customer Segmentation Case Study

<https://www.kaggle.com/subhailamathy/mall-customers-kmeans-clustering-jerry>

TO BE CONTINUED..