ANALYTICS CLUB IITB

DAV TEAM

Undergraduate Academic Council

UGAC

MACHINE LEARNING BOOKLET

2023

INTRODUCTION

Welcome to the Machine Learning Booklet brought to you by Analytics Club in collaboration with DAV Team.

This compilation aims to provide a comprehensive understanding of fundamental machine learning algorithms.

Whether you're a beginner* exploring the basics or an experienced practitioner seeking a quick reference, this booklet is designed to cater to a diverse audience.

*Familiarity with basic mathematics and programming concepts is recommended.

TABLE OF CONTENTS

01
Linear Regression

02
Logistic Regression

03
K-Nearest Neighbours(KNNs)

04
K-means clustering

05
Decision Trees

06
Random Tree Forest

07
Support Vector Machines

08
Gaussian Mixture Model

09
Principle Component Analysis

LINEAR REGRESSION

A Linear Regression algorithm attempts to model a relationship between dependent variable/s and independent variables by fitting a straight line. This line is represented using the following equation which is:

$$y = \beta_0 + \beta_1 x + u$$

where,
y is the dependent variable.
x is the independent variable.
β0 is the intercept.
β1 is the slope.
u is the random error component.

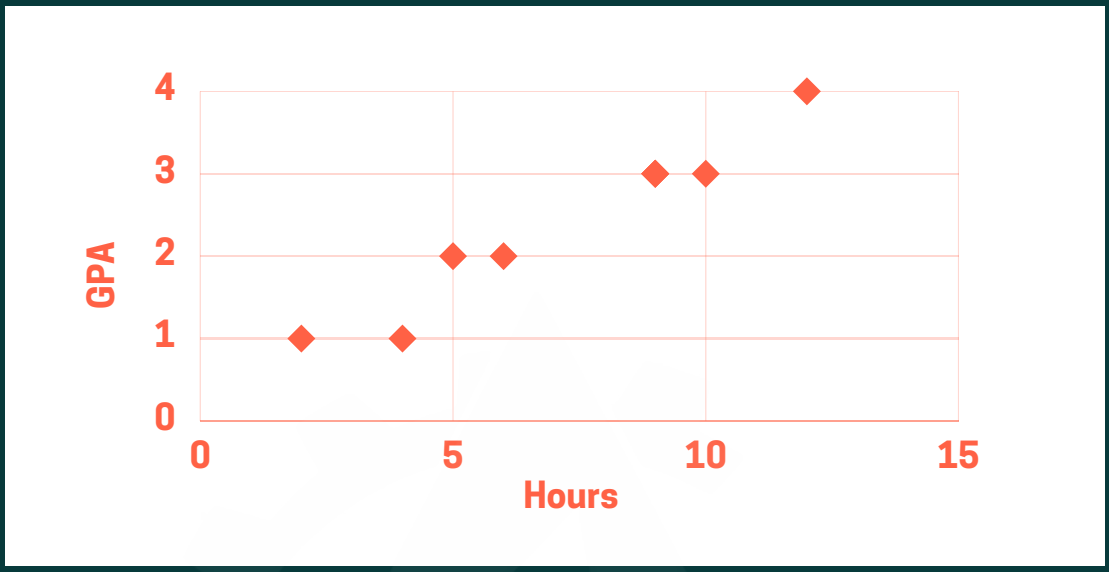
Let's take an example:

Imagine you are a student and you want to find out what GPA you will be getting this semester based on the no. of hours/day you are studying. So, what you would want to do is to establish some relationship between the two variables. Firstly, you go and collect data from all your seniors/batchmates on the same. So here y is GPA and x is average no. of hours studied/day.

Student	Hours	GPA
1	10	3
2	12	4
3	5	2
4	10	3
5	9	3

Student	Hours	GPA
6	2	1
7	4	1
8	9	3
9	9	3
10	6	2

LET'S PLOT IT ON A SCATTER PLOT



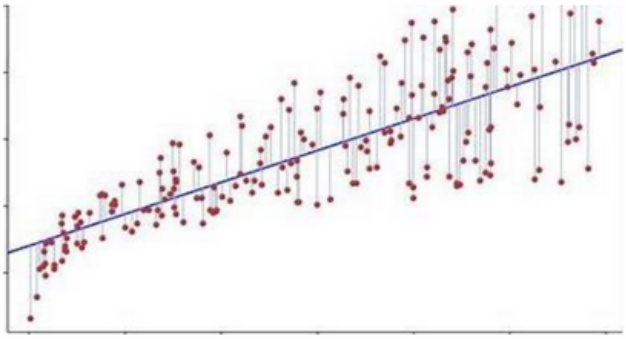
As you can clearly see there is some sort of linear relationship between the two variables, our job is to find out the best fit line which explains the relationship and can be used to predict future GPAs based on average number of study hours per day.

For that we need to define a **cost function/loss function** and minimize it. In Linear Regression our cost function will generally be Mean Squared Error (MSE).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

MSE= mean squared error
n= number of data points
Y_i = observed values
$\hat{Y}_i$ = predicted values

GRADIENT DESCENT



So, as we can see the line that we have chosen has errors (the vertical distance of original points from the line/predicted points), our job is to choose a line that minimizes the sum of squared errors (Why square? The squaring is necessary to remove any negative signs. It also gives more weight to larger differences).
A very naive approach could be trying all permutations and combinations of β0 and β1, but that could take a lot of time and is not a computationally sound method.
Instead let's use an optimization algorithm called Gradient Descent:
Gradient descent is an algorithm that numerically estimates where a function outputs its lowest values.

Gradient Descent for SE (Squared Error):

Let the squared error be represented by J (β0, β1), according to gradient descent, update rules are:

$$\beta_0 := \beta_0 - \alpha \frac{\partial}{\partial \beta_0} J(\beta_0, \beta_1)$$

$$\beta_1 := \beta_1 - \alpha \frac{\partial}{\partial \beta_1} J(\beta_0, \beta_1)$$

We will iteratively run this algorithm till we reach a local minimum. The values of β0 and β1 corresponding to that local minimum are the parameters for our best fit line. Thus, we have found the equation to our best fit line.

LOGISTIC REGRESSION

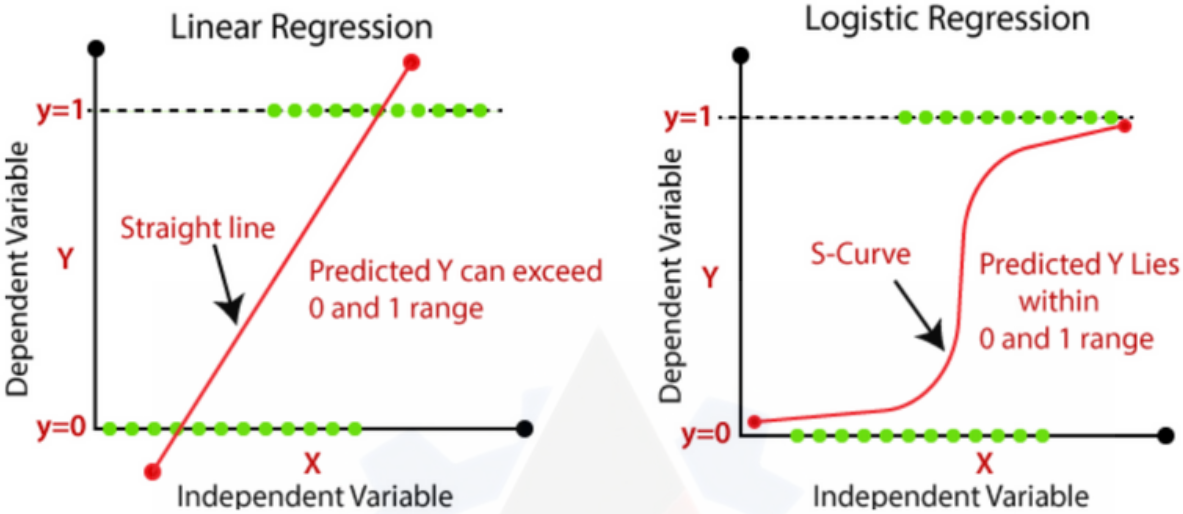
Logistic Regression stands as a fundamental and versatile tool. Despite its name, it's not just about regression; rather, it's a powerful algorithm used for classification tasks. With its intuitive foundation and wide-ranging applications, Logistic Regression has earned its place as a go-to method for both beginners and experts alike.
A Logistic Regression algorithm is used to model probabilities of the outcome, and how likely is something to happen. Will you pass or fail, does someone have cancer or not etc., things like that can be given a probability value between 0 and 1 based on the data.

This is also the major difference in the approach of Logistic Regression from Linear Regression. In the former we try to model the outcome which might be a continuous variable, in the latter we model the probability which is always between 0 and 1.

The sigmoid function is at the heart of Logistic Regression. It takes any input and maps it to a value between 0 and 1, which is crucial for predicting probabilities. The sigmoid function's mathematical form is:

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

.



How do we train our logistic regression model i.e. how do we learn the parameters?

This is normally done employing maximum likelihood estimation, which we conduct through gradient ascent. Let's see how -
Our hypothesis:

$$h(x) = \sigma(\theta^\top \bar{x}) = \frac{1}{1 + e^{-\theta^\top x}}$$

$$P(y_i = 1 \mid x_i, \theta) = \frac{1}{1 + e^{-\theta^\top x}}$$

$$P(y_i = 0 \mid x_i, \theta) = 1 - \frac{1}{1 + e^{-\theta^\top x}}$$

Let's combine both of the probabilities, the equation below is an outcome of Bernoulli distribution. Probability of Y given a particular X as an input and model parameter θ:-

$$P(y_i \mid x_i; \theta) = h(x_i)^{y_i} * (1 - h(x_i))^{1-y_i}$$

So let's say we have more than 1 independent variable, let's call all our independent variables X, dependent binary variable Y, total observations m and model parameters θ and we can say:

$$P(Y \mid X; \theta) = \prod_{i=1}^{m} h(x_i)^{y_i} * (1 - h(x_i))^{1-y_i}$$

This is the famous likelihood function(L(θ)). Now what we will do is maximize the likelihood function. But there is a problem as our sigmoid function (through which we are multiplying probabilities) is not always convex. The solution is log as the logarithm of the likelihood function is always convex. (Convex functions are easier to optimise)

$$P(Y \mid X;\theta) = \sum_{i=1}^{m} y_i Log(h(x_i)) + \ldots\ldots (1 - y_i) Log(1 - h(x_i))$$

Calculating the final derivative, we get:

$$\frac{\partial L(\theta)}{\partial \theta} = (y - \sigma(\theta^T x)) * x$$

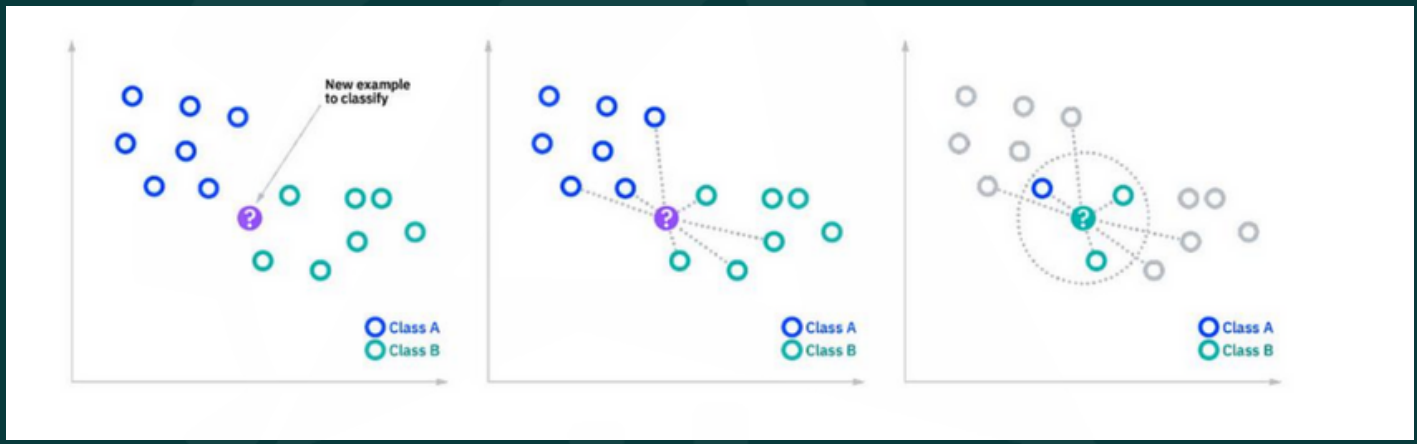
We can now use gradient ascent (Same as Gradient Descent) to optimise and find the best parameters (θ) for our model.

$$\theta^+ = \theta^- + \alpha \frac{\partial L(\theta)}{\partial \theta}$$

K-NEAREST NEIGHBOURS

K-Nearest Neighbours, often abbreviated as KNNs, is a type of non-parametric, lazy learning algorithm. Instead of building an explicit model during training, it retains the entire dataset and makes predictions based on the similarity between data points.
At its core, KNN operates on a simple principle: an object is likely to be similar to its neighbours. To predict the output for a new data point, the algorithm looks at the 'k' most similar data points (neighbours) from the dataset and takes a majority vote in case of classification or an average in case of regression.

*Classification and regression are two types of supervised learning tasks in machine learning



How a KNN model is build:

01 — Data Preprocessing

Before deploying KNN, it's important to scale the features. Since KNN relies on distance measurements, varying scales between features can distort the distances and lead to inaccurate predictions. Common scaling methods include Min-Max scaling and Z-score normalization.

KNN BUILDING

02 — Distance Metric Selection

Choose a distance metric that suits the nature of the data. The Euclidean distance is widely used, but other metrics like Manhattan and Minkowski might be more appropriate depending on the dataset's characteristics.

- *Euclidean* Norm $= \sqrt{\sum_{i=1}^{n} |x_i|^2}$
- *Manhattan* Norm $= \sum_{i=1}^{n} |p_i - q_i|$
- *Minkowski* Norm $= \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$

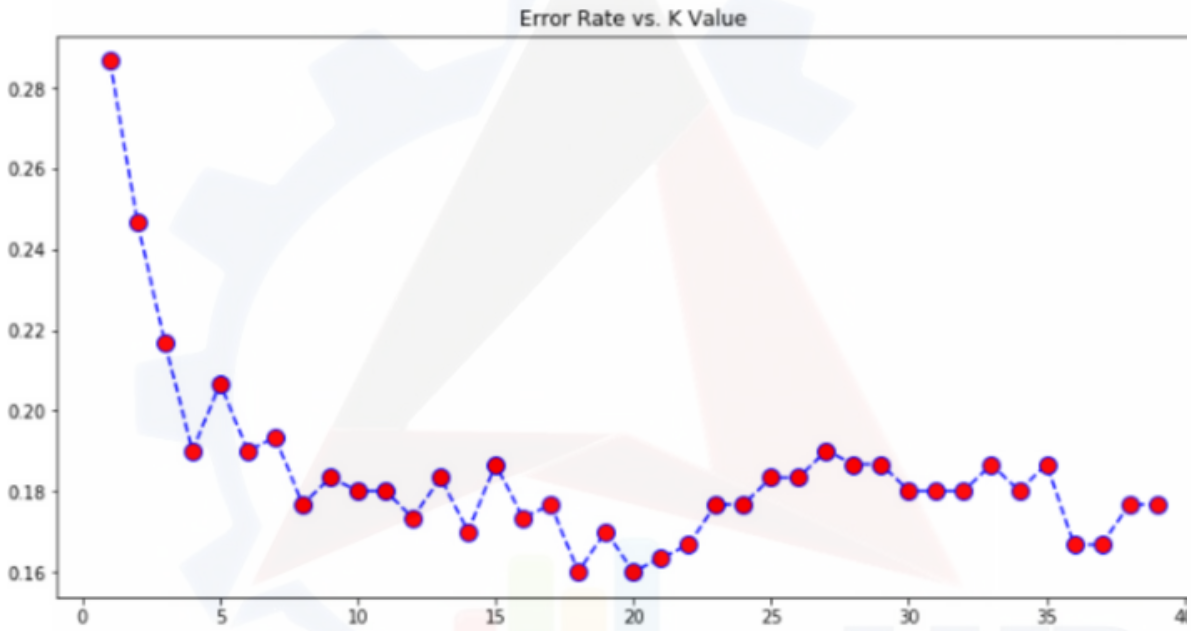
03 — Determining the optimal 'k' value

One of the primary decisions when working with KNN algorithm is determining the value of 'k' – the number of neighbours to consider while making predictions. A smaller 'k' value, like 1 or 3, can capture fine details in the data, but may also be sensitive to noise and outliers. Conversely, a larger 'k' might provide more generalized predictions, reducing the risk of overfitting*, but it could underfit* if taken to an extreme.

- Underfitting: The model is too simple and doesn't grasp the data's patterns, performing poorly on both training and new data.
- Overfitting: The model is too complex, fitting the training data too closely, memorizing noise, and failing to generalize well to new, unseen data

KNN BUILDING

The Elbow method offers a systematic approach to finding an optimal 'k' value. By plotting the error rates for various 'k' values, one can identify a point where the error rate starts to level off, suggesting an optimal 'k' value. An 'elbow' in the graph implies increasing 'k' further might not yield significant improvements in error rate.
From the error rate v/s 'k'-Value graph below we can see that there is an elbow formation around k = 20. Thus the optimal 'k'-value for this model will be 20.



04 — Prediction

Prediction: For a new data point, the algorithm calculates the distance between this point and all other points in the dataset. It then selects the 'k' nearest points and takes a majority vote (for classification) or an average (for regression) to predict the outcome.

KNN BUILDING

05 — Model Evaluation

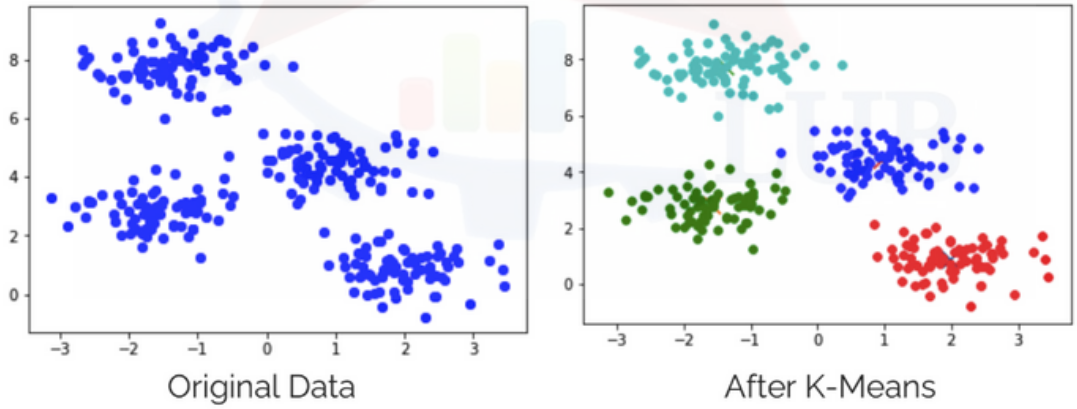
After predictions, assess the model's accuracy using appropriate evaluation metrics. For classification tasks, metrics like accuracy, precision, and recall are relevant, while regression tasks might use metrics like Mean Absolute Error or R-squared.

K-MEANS CLUSTERING

K-Means Clustering is an Unsupervised Learning Algorithm, which groups the unlabelled dataset into different clusters. Here K defines the number of pre-defined clusters or groups that need to be created in the process. For K = 5, there will be five clusters, for K = 10, there will be ten clusters, and so on. The K-means clustering algorithm mainly performs two tasks:

1. Determines the optimal value of number of clusters (K)

2. Assigns each data point to its closest K-center. Groups are assigned based on K center points by measuring the distance between K points and data points.



*The "means" in k-means refer to the centroid of each cluster.

CLUSTERING ALGORITHM

How the Algorithm works:

1. Randomly select K data points as cluster center.

2. Using one of distance norms formula measure the distance between each data point and each cluster center.

3. Assign each data point to that cluster whose center is nearest to that data point. If each cluster centroid is denoted by ci, then each data point x is assigned to a cluster based on:

$$\arg \min_{c_i \in C} dist(ci, x)^2$$

4. Re-compute the center of newly formed clusters. The center of a cluster is computed by taking the mean of all the data points contained in that cluster.

$$c_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i$$

Here, Si is the set of points in a cluster

5. This process runs iteratively until, one of the stopping criteria is met:

- Data points fall under the same cluster
- Reached maximum iterations
- The newly formed cluster does not change in center points

CLUSTERING ALGORITHM

6. Determining the optimal 'k'-value: The optimal value of 'k' can be determined using 2 methods- The Elbow Method and The Silhouette Method.

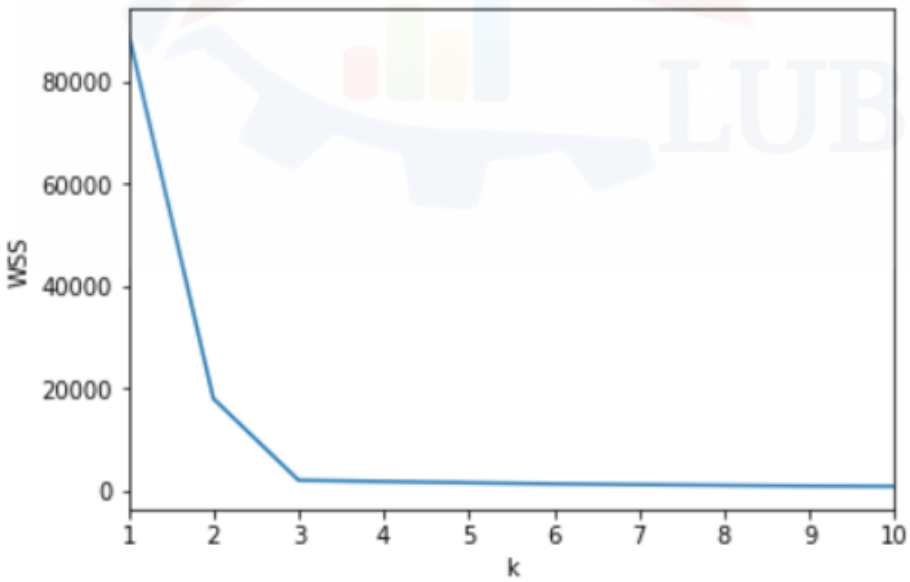
- **The Elbow Method:**

This is probably the most well-known method for determining the optimal number of clusters. Calculate the Within-Cluster-Sum of Squared Errors (WSS) for different values of k, and choose the k for which WSS first starts to diminish.

In the plot of WSS-versus-k, this is visible as an elbow.

The plot looks like an arm with a clear elbow at k = 3

Unfortunately, we do not always have such clearly clustered data. This means that the elbow may not be clear and sharp.



CLUSTERING ALGORITHM

- **The Silhouette Method**

The silhouette value measures how similar a point is to its own cluster (cohesion) compared to other clusters (separation). The range of the Silhouette value is between +1 and -1.
A high value is desirable and indicates that the point is placed in the correct cluster. If many points have a negative Silhouette value, it may indicate that we have created too many or too few clusters. Silhouette Value s(i) for each data point i, is defined as follows:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}, if\ |C_i| > 1$$

and $$s(i) = 0 \quad , if\ |C_i| = 1$$

Here, a(i) is the measure of similarity of the point i to its own cluster. It is measured as the average distance of i from other points in the cluster.

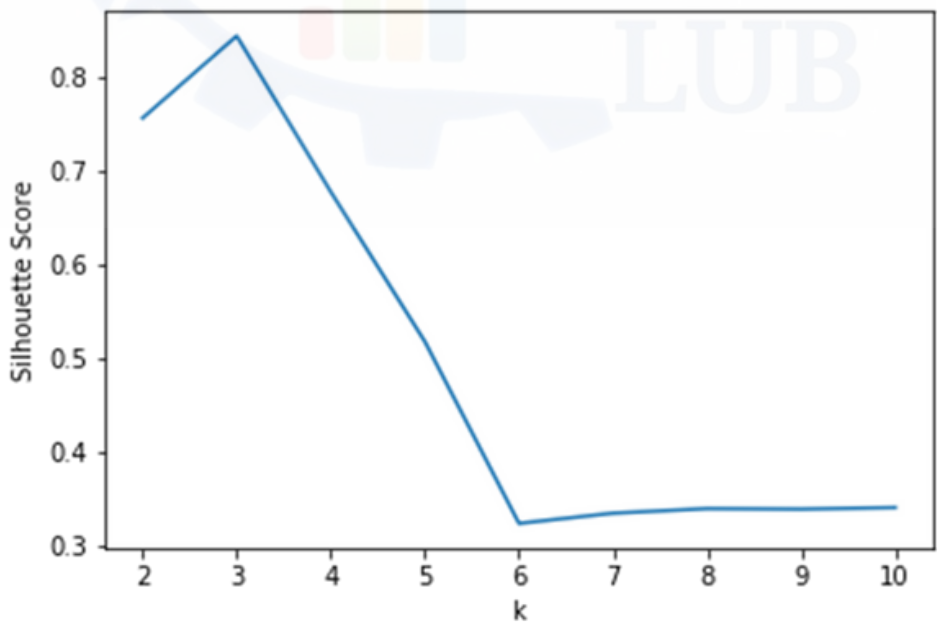
CLUSTERING ALGORITHM

Similarly, b(i) is the measure of dissimilarity of i from points in other clusters

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j)$$

$$b(i) = \min_{i \neq j} \frac{1}{|C_i|} \sum_{j \in C_j} d(i, j)$$

d(i, j) is the distance/norm between points i and j. Generally, Euclidean Distance is used as the distance metric.



CLUSTERING ALGORITHM

There is a clear peak at k=3, hence it is the optimal ‘k’-value

The Elbow Method is more of a decision rule, while the Silhouette is a metric used for validation while clustering. Thus, it can be used in combination with the Elbow Method. Therefore, the Elbow Method and the Silhouette Method are not alternatives to each other for finding the optimal K. Rather they are tools to be used together for a more confident decision.

DECISION TREES

At a high level, decision trees can be viewed as a machine learning construct used to perform either classification or regression on some data in a hierarchical structure. Decision trees use machine learning to identify key differentiating factors between the different classes of our data. By doing so, decision trees can take some input data and predict a class by running the data through a set of differentiating questions that it forms using machine learning.

The questions that are formulated are yes-no questions and after each question, there are two paths- the "yes" path and the "no" path (almost like the 'if...else' construct that we use commonly while writing code). Each of these paths will either lead to the next question or to a final output.

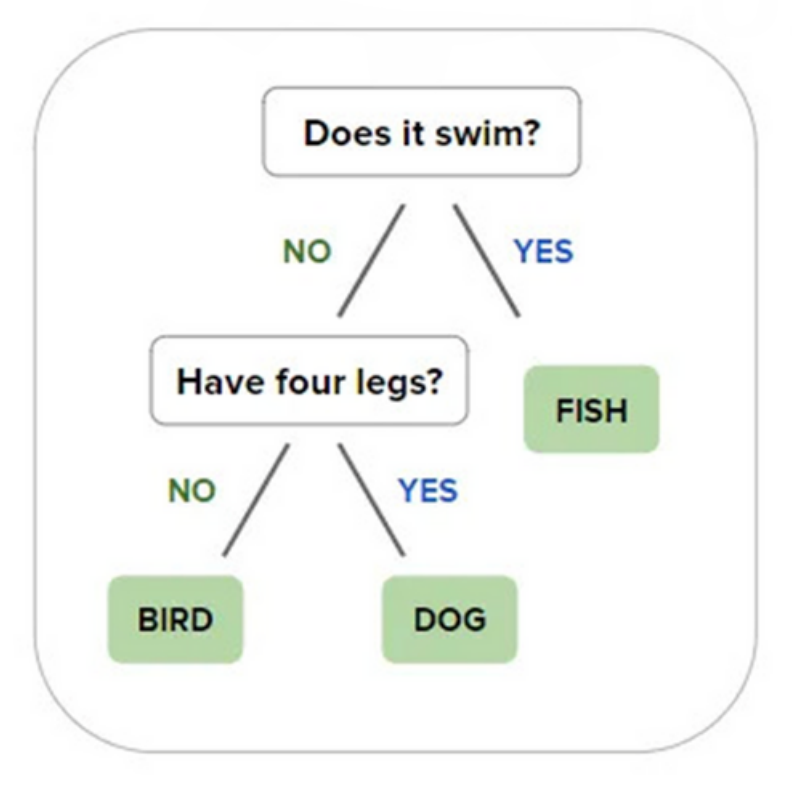
Important Terminologies

- Nodes: The various questions formed by the decision tree are known as the nodes. These can be viewed as points in the decision tree where we form a split (or branches). Each of these is a 'yes' or 'no' question and once this is answered, we move one step closer to identifying the class that the data belongs to.

DECISION TREES

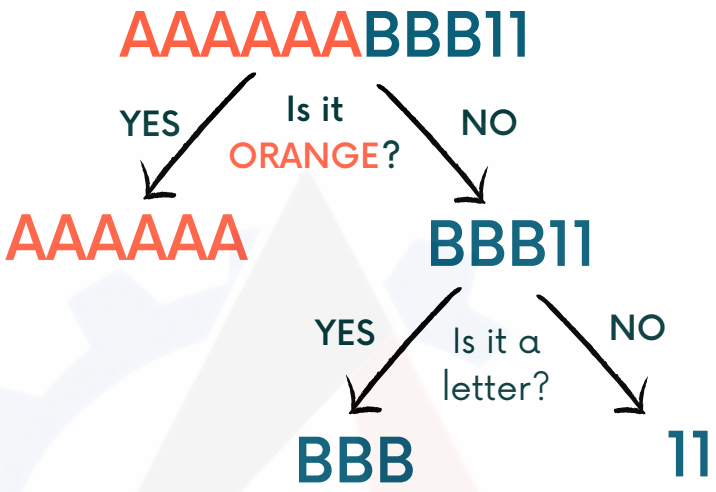
Leaves: Due to the presence of nodes, there are many alternate pathways that get created in any decision tree with even a few layers of nodes. The end points of each of these pathways are known as a "leaf". These leaves represent the final value or class that will be predicted for the given input data.

The image below depicts a highly simplified view of what a portion of a decision tree, that is being used to classify an animal as being either a bird, a dog or a fish, could look like. In this diagram, the white boxes are the nodes and the green boxes are the leaves



DECISION TREES

Decision trees, are a fairly simple concept to grasp as they are extremely intuitive, but let us use another example to consolidate this concept.



In this example, our dataset is A A A A A A B B B 1 1

The initial question asked is "Is it orange?" because the model understands that colour is the single biggest differentiating factor in our data. Based on the separation formed, we have orange 'A's at one end and a set of blue 'B's and '1's at the other. At the blue end, the model asks another question- "Is it a letter?". Once this has been answered, we split the blue data into 'B's and '1's. Now, we have obtained all 3 leaves of our model, each representing data belonging to a separate class. We have successfully classified our data!

The most widely used algorithm when actually constructing decision trees makes use of two concepts known as 'entropy' and 'information gain' to design the decision tree.

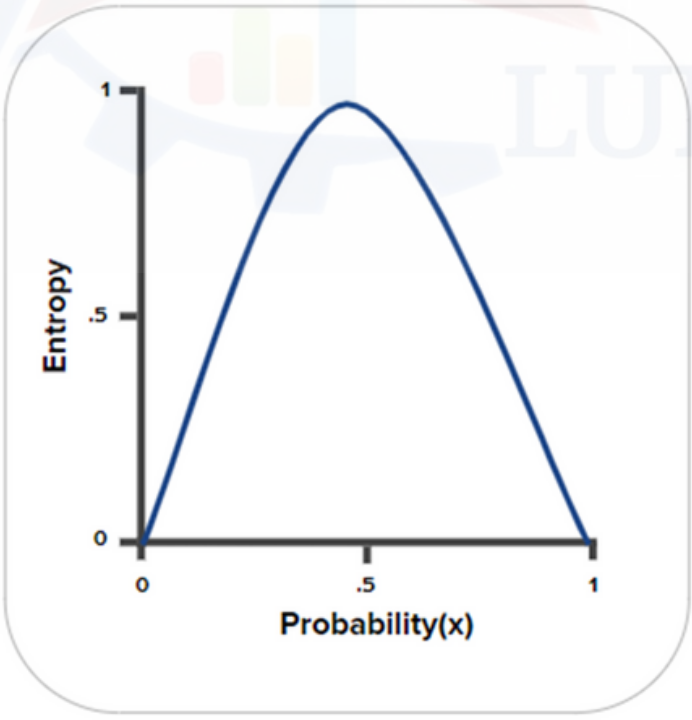
DECISION TREES

Entropy: Usually, the term 'entropy' is used to refer to a measure of disorder in a given system. With decision trees, entropy can be understood as a measure of the homogeneity of a sample of a group of data. Usually, entropy is calculated on a scale of 0 to 1. An entropy of 0 indicates that the data has minimal disorder (it is very homogeneous/pure), whereas an entropy of 1 (or a high value of entropy) indicates that there is a lot of disorder in the data.

The formula for calculating Entropy is:

$$E = \sum_{i=1}^{n} -p_i log_2 p_i$$

Where 'n' is the number of classes and 'pi' is the probability of a data point in our group belonging to the class 'i'.

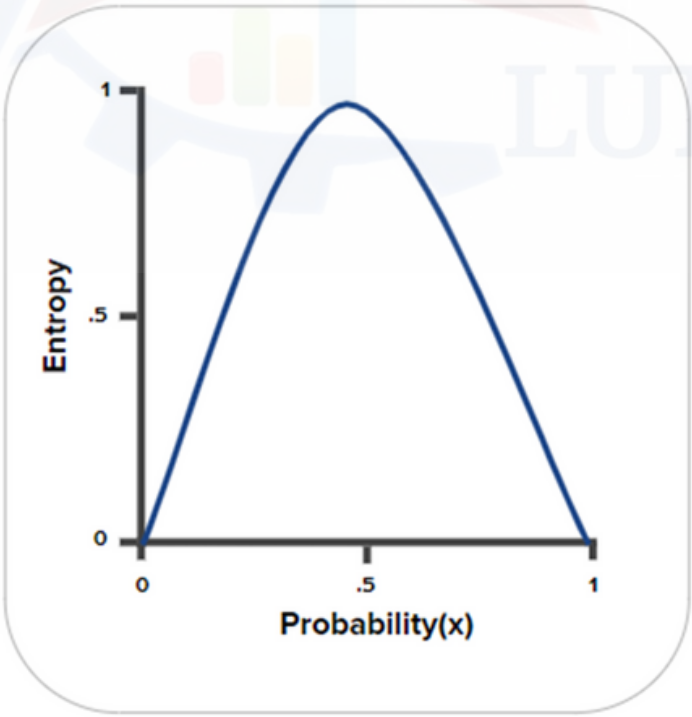


DECISION TREES

Let us assume that we have a dataset which contains data points belonging to one of two different classes- 0 and 1. We have 100 data points out of which 40 belong to class 0 and 60 belong to class 1. The entropy of this dataset will then be:

$$E = -\frac{40}{100}\left(log_2\left(\frac{40}{100}\right)\right) - \frac{60}{100}\left(log_2\left(\frac{60}{100}\right)\right) = 0.97$$

This indicates that such a group of data has a very high entropy. Each time we split our data to form a new group of data, the entropy of that group of data can be calculated. The lower the entropy, the closer we are getting to forming groups with unique data points, i.e. as the entropy reduces, we are getting closer to being able to successfully classify our data.
After understanding the concept of Entropy, it becomes clear that a node with an entropy value of 0 is what is considered to be a leaf (it has data points belonging to only one class).



DECISION TREES

Information Gain: Now that we have found a way to measure the disorder of the dataset in the form of Entropy, we must also ensure that we have a way to ensure that we are reducing the entropy by splitting the data with the questions we ask. Information Gain is what we use to ensure that our entropy is reducing. Thus, information gain can be understood as a measure of the decrease in the amount of disorder

$$Information\ Gain\ (X\ ,\ A)\ =\ E(X)\ -\ E(X\ ,\ A)$$

Here,

- X: The target variable (the data points present at that node)
- A: The attribute on the basis of which this split has been formed
- E(X): The entropy of the data at the node before the split
- E(X , A): The weighted sum of the entropies of the two branches formed after the split based on the attribute 'A'.

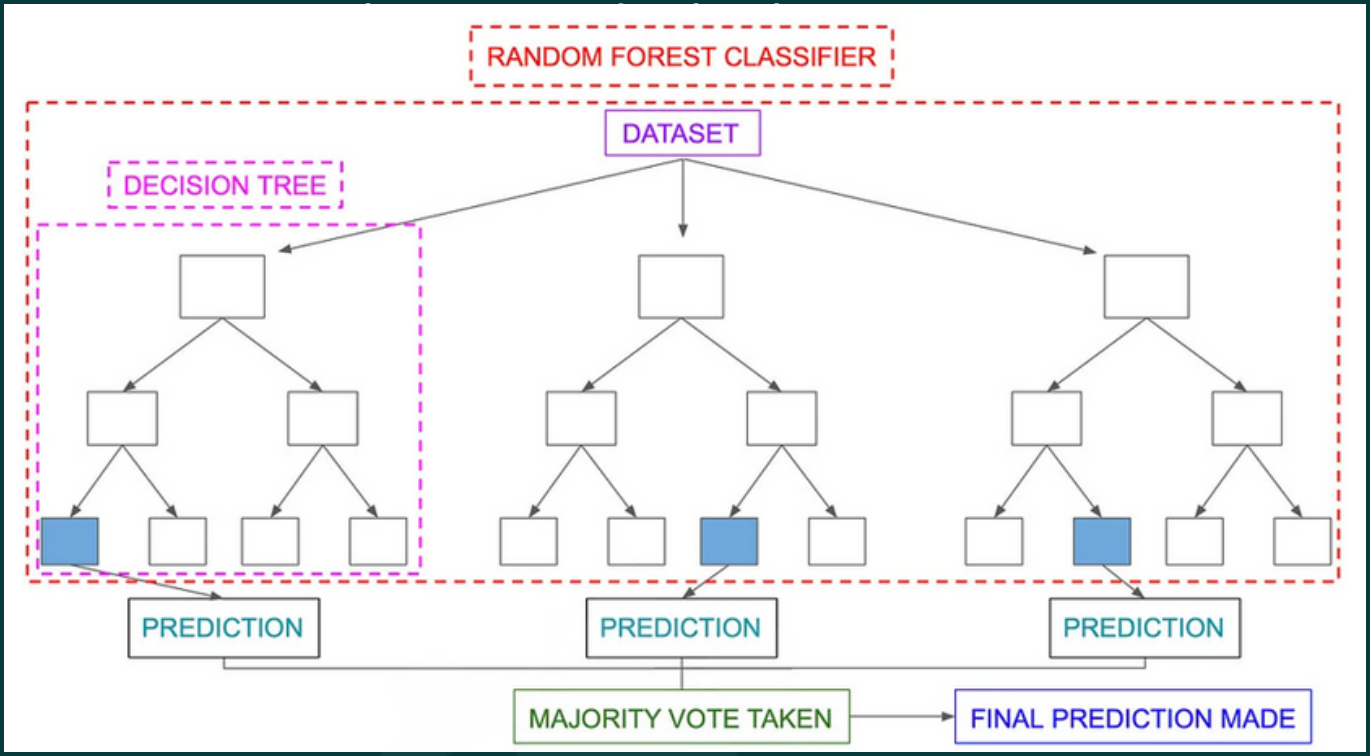
Thus, our goal in the algorithm can be understood as finding the questions to be used to split the data so that we can minimize entropy and maximize information gain. The process described above is repeated until we manage to separate all different kinds of data from each other and only leaves containing one type of data each are left.

RANDOM FOREST CLASSIFIER

Despite all the merits of the decision tree model, there is a limit to how accurately one individual decision tree can perform classification. Thus, rather than using just one decision tree, very often, we tend to create a set of decision trees that perform classification. Such a model is known as a 'Random Forest Classifier' and it yields a significantly higher accuracy than that of an individual decision tree.

A simple way to comprehend this model is to interpret its name in a very literal sense. A forest can be viewed as a collection of trees. This suggests that a Random Forest model will consist of various decision trees. The reason these are called "Random" is because each decision tree in the forest is trained using a random subset of the training data. This technique of training each tree in the forest using a different, random sample of the data is known as Bagging (also known as Bootstrap aggregation). In typical Random Forest models, a randomly sampled subset of the data on which classification is to be performed is passed through each of the decision trees in the forest in order to train that tree. Later, when we are performing classification for a datapoint, each tree constituting the forest gives a predicted class (label). The final prediction is then decided by measuring which prediction was made by the most number of trees in the forest.

RANDOM FOREST CLASSIFIER



Another advantage offered by using random forest classifiers is that they help us eliminate problems that can be caused by outliers in our dataset.

An outlier can be considered to be any datapoint that when plotted on a graph is far away from the rest of the points that are similar to it.
With most machine learning models, such outliers have a fairly large impact on the model and reduce the overall accuracy of the model.

With a random forest classifier, we significantly reduce the problem caused by such outliers. Since we are taking a random sample of the data for each tree, in any dataset of a fair size, while there will be some trees that will be trained with the outliers that might consequently produce wrong predictions, most of the trees will not have been trained using the outliers and thus, the overall prediction we receive will still be correct.

RANDOM FOREST CLASSIFIER

Random forest classifiers act as an ensemble. Since all of the decision trees that constitute the 'random forest' are largely independent of each other due to the random sampling of data points (bagging), it ensures that the predictions that we make on the basis of the majority vote obtained from the trees have a higher accuracy than a single prediction obtained from any one tree. This is also one of the main reasons that random forest classifiers have a much higher accuracy than other classifying models operating on the same dataset.

Even though Random Forest Classifier is an extremely powerful classification algorithm, it has some drawbacks:

- Generating lots of trees can be a time-consuming process since we have to calculate entropies and information gain many times.

- When the data we have is very noisy, these models often tend to overfit (overfitting is when for some reason, our model is trained in such a manner that it becomes very good at classifying only that data which is present in our training set but is poor at classifying other data of the same type, i.e. the model only learns to classify the specific examples that have been used to train it).

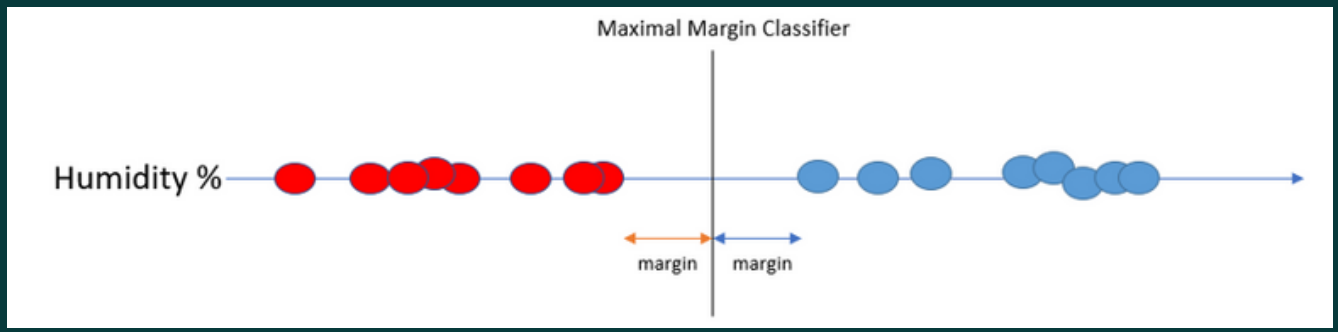
SUPPORT VECTOR MACHINES

Let's understand the Support Vector Machines (SVMs) step by step.

Suppose that we have 1D humidity data, and red dots represent the days that are not rainy and blue dots represent the rainy days.

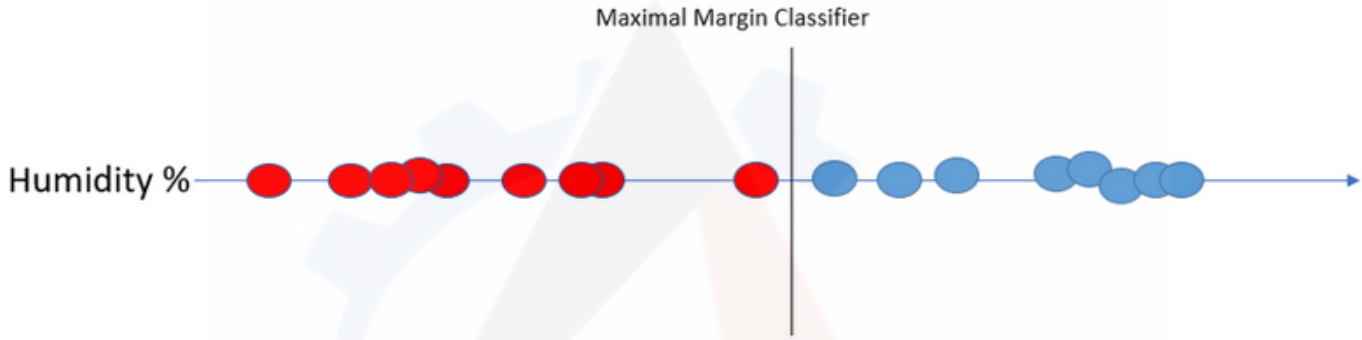
Based on this 1D observation data we have, we can determine a threshold value. This threshold value will act as a classifier. Since our data is 1D, the classifier will have a threshold value. If our data were 2D, we would use a line.

- The shortest distance between the observed data (closest data points) and the classifier threshold is called the margin. The threshold that can provide the largest margin is called Maximal Margin Classifier (Hyperplane). In our case, it will be on the midpoint of both sides' closest data

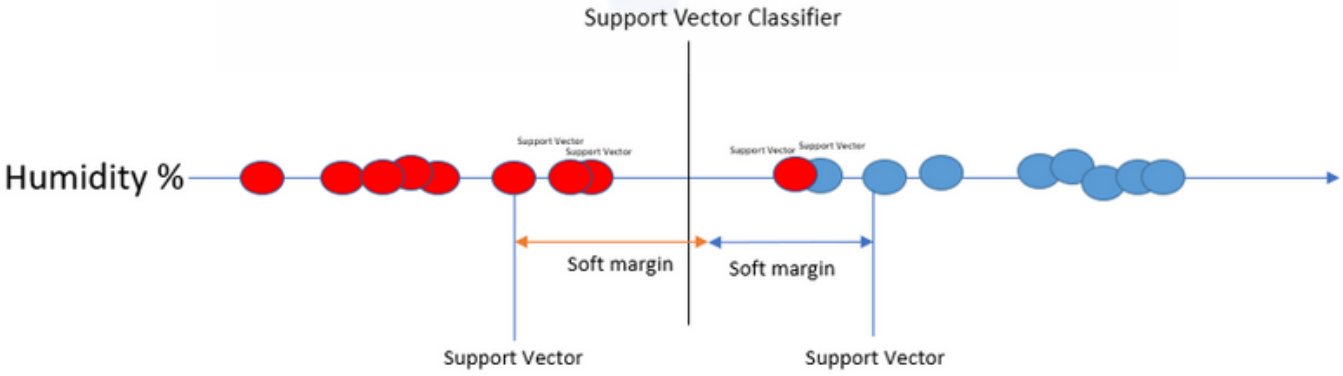


SUPPORT VECTOR MACHINES

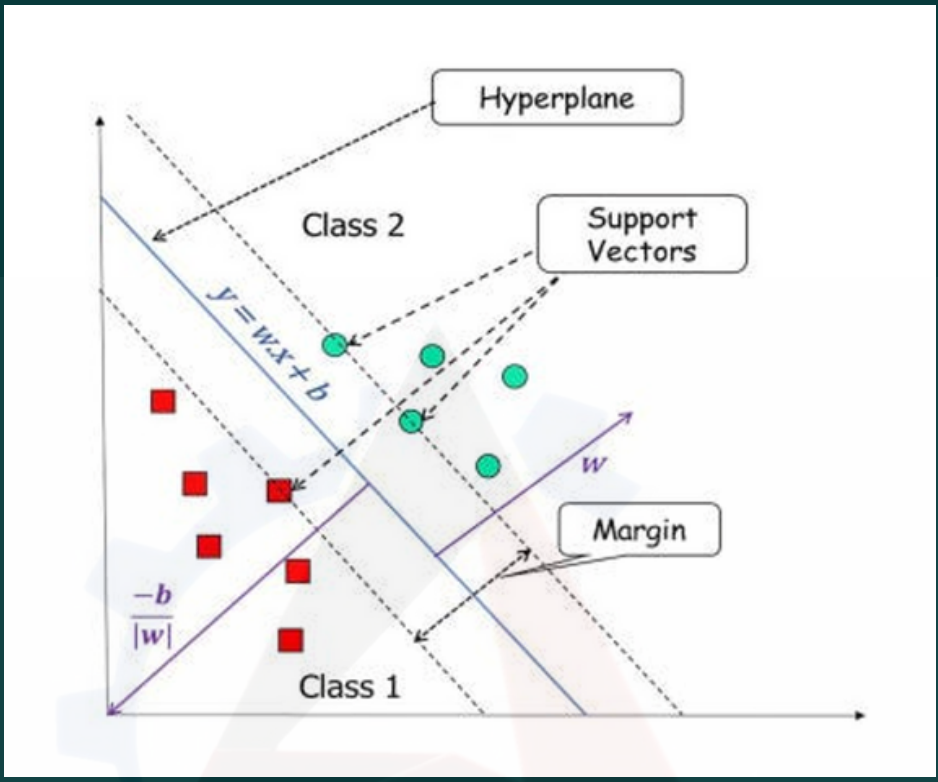
Maximal Margin is not very applicable in practice. Because it has no resistance against outliers. Imagine that we have an outlier red point that has a bluish value. In that case, the classifier will be super close to the blue points, and far from the red points.



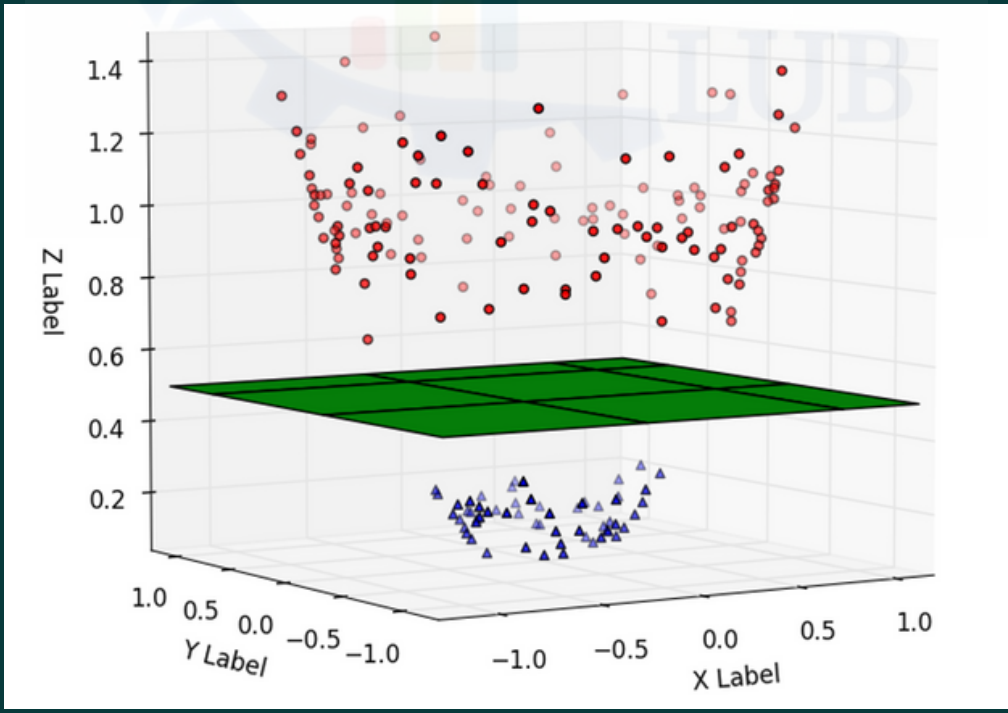
In order to improve this, we should let outliers and misclassifications. We introduce bias into the system (and decrease variance). Now, the margins are called Soft Margins. The classifiers that use Soft Margins are called Support Vector Classifiers or Soft Margin Classifiers. The data points on the edge and within the Soft Margins are called Support Vectors.



We use cross-validation to determine where the soft margins should be.
In 2D data, a Support Vector Classifier is a line.

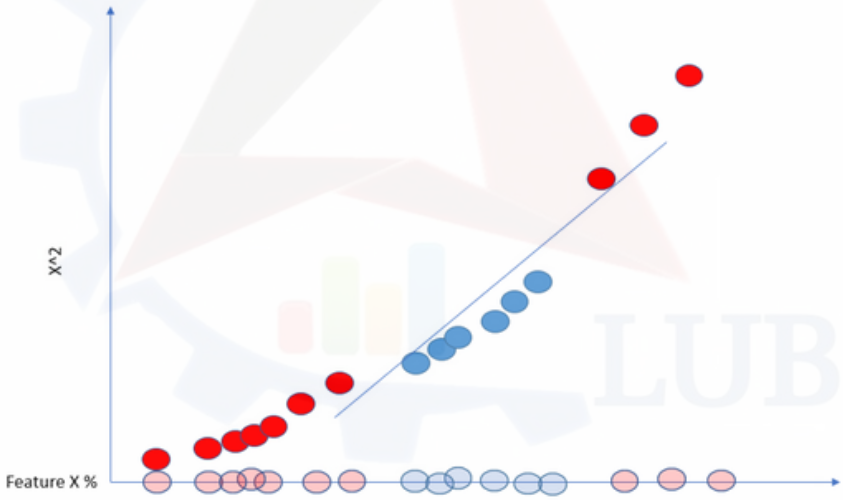


In 3D, it is a plane.

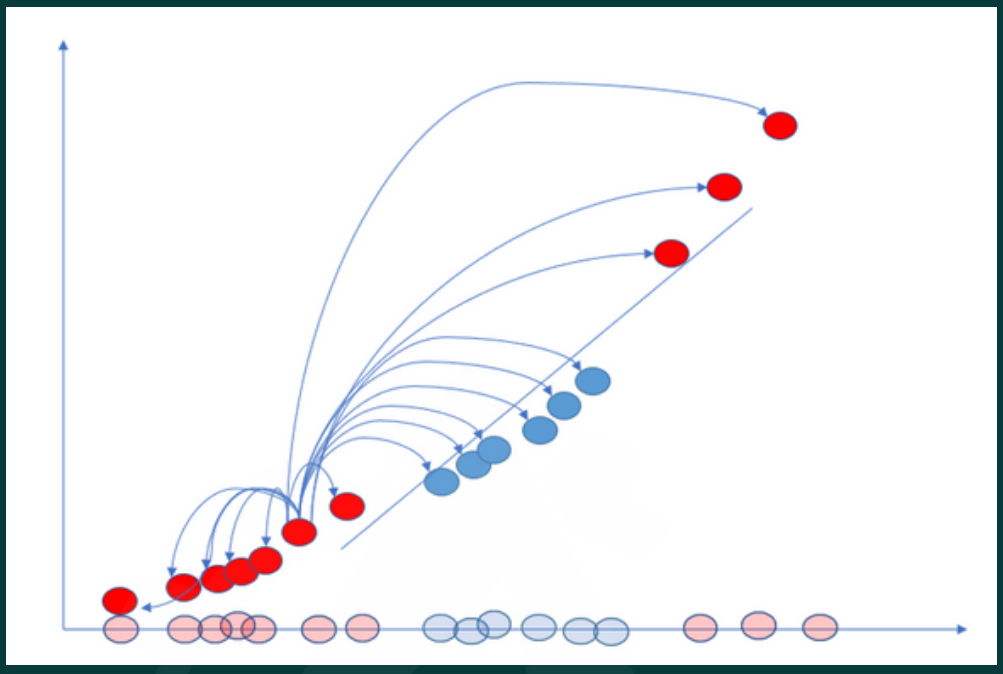


In 4 or more dimensions, Support Vector Classifier is a hyperplane. Technically all SVCs are a hyperplane, but it is easier to call them planes in the case of 2D.
As we see above, Support Vector Classifiers can handle outliers and let misclassification. But, how can we handle overlapped data as shown below?

That's where the Support Vector Machines come to play. Let's add another dimension to the problem. We have feature X and as a new dimension, we take a square of X and plot it in y-axe.
Since the data is 2D right now, we can draw a Support Vector Classifier line.



Support Vector Machines take low dimensional data, move it into a higher dimension, and find a Support Vector Classifier.
Similar to what we have done above, Support Vector Machines use Kernal Functions to find Support Vector Classifiers in higher dimensions. A kernel function is a function that takes two input data points in the original input space and calculates the inner product of their corresponding feature vectors in a transformed (higher-dimensional) feature space.



The kernel function allows SVM to operate in the transformed feature space without explicitly calculating the transformed feature vectors, which can be computationally expensive for large datasets or complex transformations. Instead, the kernel function computes the inner product between the feature vectors directly in the original input space. This is called the Kernel Trick.

The Polynomical Kernel

The polynomial kernel is used to transform the input data from a lower-dimensional space to a higher-dimensional space where it is easier to separate the classes using a linear decision boundary

$$(axb + r)^d$$

y,

NEXT STEPS

a and b are two different observations; r is the polynomial coefficient, and d is the polynomial degree. Let's say that d is 2 and r is 1/2.

$$
\begin{aligned}
(axb + r)^d &= (axb + \frac{1}{2})^2 \\
&= (axb + \frac{1}{2})(axb + \frac{1}{2}) \\
&= a^2b^2 + \frac{1}{2}ab + \frac{1}{2}ab + \frac{1}{4} \\
&= ab + a^2b^2 + \frac{1}{4} \\
&= \left(a, a^2, \frac{1}{2}\right).(b, b^2, \frac{1}{2})
\end{aligned}
$$

We get a dot product in the end. The first terms (a and b) are the x-axis, and the second terms (a^2 and b^2) are the y-axis. So, all we need to do is calculate the Dot Products between each pair of points.

The Radial Kernel (RBF)

Radial Kernel finds Support Vector Classifiers in infinite dimensions.

It assigns a higher weight to points closer to the test point and a lower weight to issues farther away (like nearest neighbors). Observations that are further away have relatively little influence on the classification of a data point.

$$e^{-\gamma(a-b)^2}$$

It calculates the squared distance between two data. Gamma is determined by cross-validation and it scales the squared distance, meaning that it scales the influence two points have on each other. In this formula, the value will approach zero as the distance between two points increases.

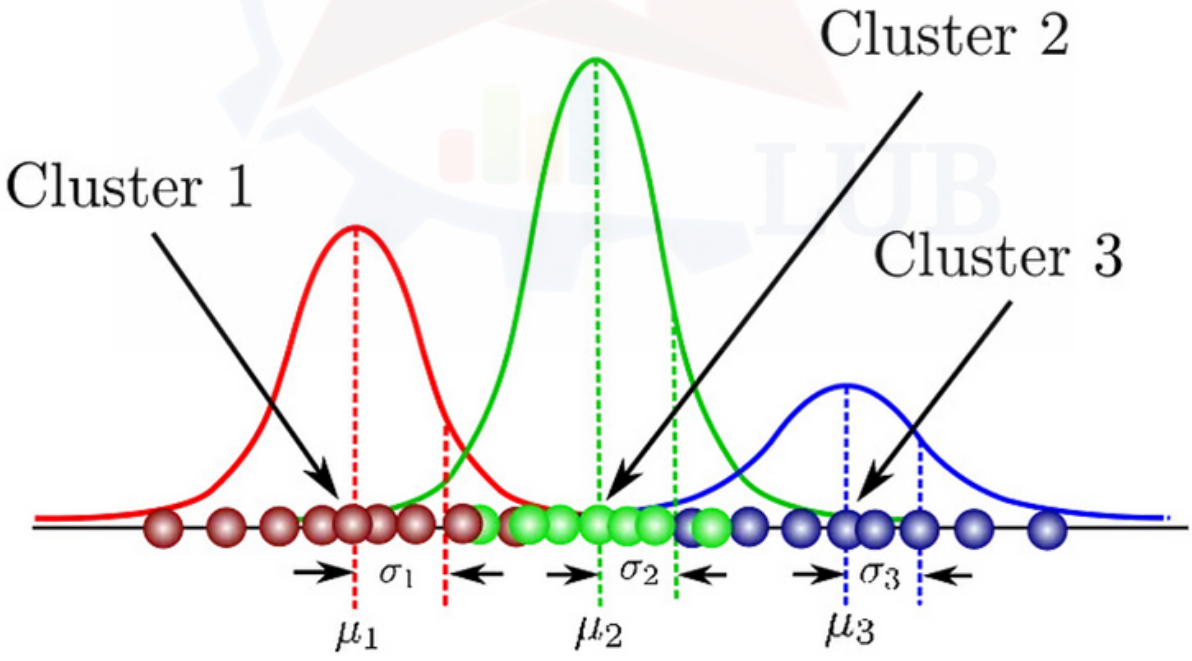
The radial kernel is particularly useful when the decision boundary between the classes is non-linear and complex, as it can capture complex relationships between the input features.

In general, SVMs are suitable for classification tasks where the number of features is relatively small compared to the number of samples, and where there is a clear margin of separation between the different classes. SVMs can also handle high-dimensional data and non-linear relationships between the features and the target variable. However, SVMs may only be suitable for some large datasets, as they can be computationally intensive and require much memory.

GAUSSIAN MIXTURE MODEL(GMM)

A Gaussian Mixture is a function comprised of several Gaussians, each identified by $k \in \{1,\ldots, K\}$, where K is the number of clusters of our dataset. Each Gaussian k in the mixture is comprised of the following parameters:

- A mean μ that defines its center.
- A covariance Σ that defines its width. This would be equivalent to the dimensions of an ellipsoid in a multivariate scenario.
- A mixing probability π that defines how big or small the Gaussian function will be.



GMM

Here, we can see three Gaussian functions, hence K = 3. Each Gaussian explains the data contained in each of the three clusters available. The mixing coefficients are themselves probabilities and must meet this condition:

$$\sum_{k=1}^{K} \pi_k = 1$$

Now, how do we determine the optimal values for these parameters? To achieve this, we must ensure that each Gaussian fits the data points belonging to each cluster. This is precisely what maximum likelihood does.
In general, the Gaussian density function is given by:

$$\mathcal{N}(\mathbf{x}|\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right)$$

Where x represents our data points, and D is the number of dimensions of each data point. μ and Σ are the mean and covariance, respectively. If we have a dataset of N = 1000 three-dimensional points (D = 3), then x will be a 1000 × 3 matrix. μ will be a 1 × 3 vector, and Σ will be a 3 × 3 matrix. For later purposes, we will also find it helpful to take the log of this equation, which is given by:

$$\ln\mathcal{N}(\mathbf{x}|\mu,\Sigma) = -\frac{D}{2}\ln 2\pi - \frac{1}{2}\ln\Sigma - \frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)$$

GAUSSIAN MIXTURE MODEL(GMM)

If we differentiate this equation concerning the mean and covariance and then equate it to zero, we can find the optimal values for these parameters, and the solutions will correspond to the Maximum Likelihood Estimates (MLE) for this setting. However, because we are dealing with not just one but many Gaussians, things will get complicated when we find the parameters for the whole mixture. In this regard, we will need to introduce some more aspects we can talk about in the next section.

Initial Derivation

We are now going to introduce some additional notation. First, suppose we want to know the probability that a data point xn comes from Gaussian k. We can express this as:

$$p(z_{nk} = 1|\mathbf{x}_n)$$

Which reads, "Given a data point x, what is the probability it came from Gaussian k?" Z is a latent variable that takes only two possible values in this case. It is one when x came from Gaussian k and zero otherwise. We don't see this z variable in reality, but knowing its probability of occurrence will help us determine the Gaussian mixture parameters, as discussed later.
Likewise, we can state the following:

GAUSSIAN MIXTURE MODEL(GMM)

$$\pi_k = p(z_k = 1)$$

This means that the overall probability of observing a point that comes from Gaussian k is equivalent to the mixing coefficient for that Gaussian. This makes sense because the more significant the Gaussian is, the higher we would expect this probability to be. Now let z be the set of all possible latent variables z, hence:

$$Z = \{z_1, z_2 \dots z_k\}$$

$$p(\mathbf{z}) = p(z_1 = 1)^{z_1} p(z_2 = 1)^{z_2} \dots p(z_K = 1)^{z_K} = \prod_{k=1}^{K} \pi_k^{z_k}$$

We know beforehand that each z occurs independently of others and that they can only take the value of one when k is equal to the cluster the point comes from. Therefore

What about finding the probability of observing our data, given that it came from Gaussian k? It turns out to be the Gaussian function itself! Following the same logic we used to define p(z), we can state:

$$p(\mathbf{x}_n|\mathbf{z}) = \prod_{k=1}^{K} \mathcal{N}(\mathbf{x}_n|\mu_k, \mathbf{\Sigma}_k)^{z_k}$$

Ok, now you may be asking, why are we doing all this? Remember our initial aim was to determine the probability of z given our observation x? The equations we have just derived and the Bayes rule will help us resolve this probability. From the product rule of possibilities, we know that.

$$p(\mathbf{x}_n, \mathbf{z}) = p(\mathbf{x}_n|\mathbf{z})p(\mathbf{z})$$

Hmm, we are getting somewhere now. The operands on the right are what we have just found. However, first, we will need p(xn), not p(xn, z). So, how do we get rid of z here? Marginalization is the answer! We need to sum up the terms on z, hence.

$$p(\mathbf{x}_n) = \sum_{k=1}^{K} p(\mathbf{x}_n|\mathbf{z})p(\mathbf{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)$$

This equation defines a Gaussian Mixture, and we can see that it depends on all the previously mentioned parameters! To determine the optimal values for these, we need to select the maximum likelihood of the model. We can find the likelihood as the joint probability of all observations xn, defined by:

$$p(\mathbf{X}) = \prod_{n=1}^{N} p(\mathbf{x}_n) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k)$$

Like we did for the original Gaussian density function, let's apply the log to each side of the equation:

$$\ln p(\mathbf{X}) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k) \qquad (3)$$

To find the optimal parameters for the Gaussian mixture, we have to differentiate this equation concerning the parameters, and we are done.

EXPECTATION MAXIMIZATION ALGORITHM

Calculating the final derivative for "ln p(X)" in the GMM model is possible, but we can see a logarithm affecting the second summation.
So, calculating the derivative of this expression and then solving for the parameters will be very hard!
What can we do? Well, we need to use an iterative method to estimate the parameters. But first, remember we were supposed to find the probability of z given x?
Well, let's do that since, at this point, we already have everything in place to define what this probability will look like.

From Bayes's rule, we know that

$$p(z_k = 1|\mathbf{x}_n) = \frac{p(\mathbf{x}_n|z_k = 1)p(z_k = 1)}{\sum_{j=1}^{K} p(\mathbf{x}_n|z_j = 1)p(z_j = 1)}$$

THUS,

$$p(z_k = 1|\mathbf{x}_n) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\mu_j, \Sigma_j)} = \gamma(z_{nk}) \qquad (4)$$

And this is what we've been looking for! Moving forward, we are going to see this expression a lot. Next, we will continue our discussion with a method to help us easily determine the parameters for the Gaussian mixture.

The iterative method used to estimate the parameters for GMM is called Expectation — Maximization, or simply EM algorithm.
It is widely used for optimization problems where the objective function has complexities, such as the one we've just encountered for the GMM case.
Let the parameters of our model be

$$\theta = \{\pi, \mu, \Sigma\}$$

Let us now define the steps that the general EM algorithm will follow:

Step 1: Initialise θ accordingly. For instance, we can use the results obtained by a previous K-Means run as a good starting point for our algorithm.
Step 2 (Expectation step): Evaluate

$$\mathcal{Q}(\theta^*, \theta) = \mathbb{E}[\ln p(\mathbf{X}, \mathbf{Z}|\theta^*)] = \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \theta) \ln p(\mathbf{X}, \mathbf{Z}|\theta^*) \qquad (5)$$

We have already found p(Z|X, θ). Remember the γ expression we ended up with in the previous section? For better visibility, let's bring our earlier equation (4) here:

$$p(z_k = 1|\mathbf{x}_n) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\mu_j, \Sigma_j)} = \gamma(z_{nk}) \qquad (4)$$

For Gaussian Mixture Models, the expectation step boils down to calculating the value of γ in (4) using the old parameter values.
Now if we replace (4) in (5), we will have:

$$\mathcal{Q}(\theta^*, \theta) = \sum_{\mathbf{Z}} \gamma(z_{nk}) \ln p(\mathbf{X}, \mathbf{Z}|\theta^*) \qquad (6)$$

But we still miss p(X, Z|θ*). How can we find it? Well, it's not that difficult. It is just the complete likelihood of the model, including both X and Z, and we can find it by using the following expression:

$$p(\mathbf{X}, \mathbf{Z}|\theta^*) = \prod_{n=1}^{N} \prod_{k=1}^{K} \pi^{z_{nk}} \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)^{z_{nk}}$$

Well, I know this is much math! But please do it a little, as we are almost there. We have finally eliminated this troublesome logarithm that affected the summation in (3). With all of this in place, estimating the parameters by maximizing Q concerning the parameters will be much easier. Still, we will deal with this in the maximization step. Besides, remember that the latent variable z will only be one once every time the summation is evaluated. With that knowledge, we can quickly eliminate it as needed for our derivations.

Finally,

$$\mathcal{Q}(\theta^*, \theta) = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) [\ln \pi_k + \ln \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)] \qquad (8)$$

In the maximization step, we will find the revised parameters of the mixture. For this purpose, we will need to make Q a restricted maximization problem, and thus, we will add a Lagrange multiplier to (8). Let's now review the maximization step.

Step 3 (Maximization step): Find the revised parameters θ* using:

$$\theta^* = \arg\max_\theta \mathcal{Q}(\theta^*, \theta)$$

However, Q should also consider the restriction that all π values should sum up to one. To do so, we will need to add a suitable Lagrange multiplier. Therefore, we should rewrite (8) in this way:

$$\mathcal{Q}(\theta^*, \theta) = \sum_{n=1}^{N}\sum_{k=1}^{K}\gamma(z_{nk})[\ln \pi_k + \ln \mathcal{N}(x_n|\mu_k, \Sigma_k)] - \lambda\left(\sum_{k=1}^{K}\pi_k - 1\right) \quad (8)$$

Now, we can easily determine the parameters by using maximum likelihood. Let's now take the derivative of Q concerning π and set it equal to zero:

$$\frac{\partial \mathcal{Q}(\theta^*, \theta)}{\partial \pi_k} = \sum_{n=1}^{N}\frac{\gamma(z_{nk})}{\pi_k} - \lambda = 0$$

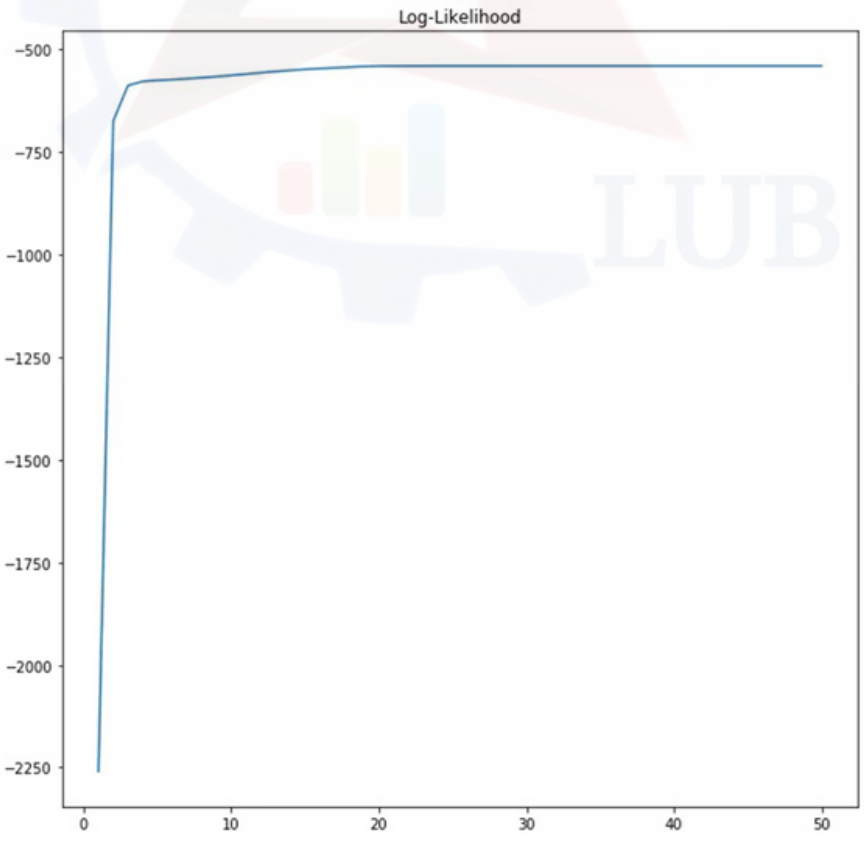
From (1), we know that the summation of all mixing coefficients π equals one. In addition, we know that summing up the probabilities γ over k will also give us 1. Thus, we get λ = N. Using this result, we can solve for π:

$$\pi_k = \frac{\sum_{n=1}^{N}\gamma(z_{nk})}{N}$$

Similarly, if we differentiate Q concerning μ and Σ, equate the derivative to zero, and then solve for the parameters by making use of the log-likelihood equation (2) we defined, we obtain:

$$\mu_k^* = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n}{\sum_{n=1}^{N} \gamma(z_{nk})}, \qquad \Sigma_k^* = \frac{\sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \mu_k)(\mathbf{x}_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma(z_{nk})}$$

And that's it! Then, we will use these revised values to determine γ in the next EM iteration until we see some convergence in the likelihood value. We can use equation (3) to monitor the log-likelihood in each step, and we are always guaranteed to reach a local maximum.



PRINCIPAL COMPONENT ANALYSIS

The central idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of many interrelated variables while retaining as much of the variation present in the data set as possible. This is achieved by transforming a new set of variables, the principal components (PCs), which are uncorrelated and ordered so that the first few retain most of the variation in the original variables.

Mathematics Behind PCA

PCA can be thought of as an unsupervised learning problem. The whole process of obtaining principle components from a raw dataset can be simplified into six parts :

- Take the whole dataset consisting of d+1 dimensions and ignore the labels so our new dataset becomes d-dimensional.
- Compute the mean for every dimension of the whole dataset.
- Compute the covariance matrix of the whole dataset.
- Compute eigenvectors and the corresponding eigenvalues.
- Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a d × k dimensional matrix W.
- This d × k eigenvector matrix transforms the samples onto the new subspace.

MATHEMATICS BEHIND PCA

So, let's unfurl the maths behind each of these individually.

1. Take the whole dataset consisting of d+1 dimensions and ignore the labels so our new dataset becomes d-dimensional.

Let's say we have a dataset which is d+1 dimensional. In the modern machine learning paradigm, d could be considered X_train, and one could be y_train (labels). So, X_train + y_train makes up our complete train dataset.
So, after we drop the labels, we are left with a d-dimensional dataset and this would be the dataset we will use to find the principal components. Also, let's assume we are left with a three-dimensional dataset after ignoring the labels, i.e., d = 3.
We will assume that the samples stem from two different classes, where one-half of the pieces of our dataset are labeled class 1 and the other half class 2.
Let our data matrix X be the score of three students :

Student	**Math**	**English**	Art
1	90	60	90
2	90	90	30
3	60	60	60
4	60	60	90
5	30	30	30

MATHEMATICS BEHIND PCA

2. Compute the mean of every dimension of the whole dataset.

The data from the above table can be represented in matrix A, where each column in the matrix shows scores on a test, and each row shows a student's score.

$$A = \begin{bmatrix} 90 & 60 & 90 \\ 90 & 90 & 30 \\ 60 & 60 & 60 \\ 60 & 60 & 90 \\ 30 & 30 & 30 \end{bmatrix}$$

Matrix A

So, The mean of matrix A would be:

$$A = [66 \quad 60 \quad 60]$$

Mean of Matrix A

3. Compute the covariance matrix of the whole dataset (sometimes also called the variance-covariance matrix)

So, we can compute the covariance of two variables, X and Y, using the following formula :

$$cov\,(X,Y) = \sum_{i=1}^{N} \frac{\left(x_i - \bar{x}\right)\left(y_i - \bar{y}\right)}{N}$$

Using the above formula, we can find the covariance matrix of A. Also, the result would be a square matrix of d ×d dimensions.

MATHEMATICS BEHIND PCA

Let's rewrite our original matrix like this (Matrix A)

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccc} Math & English & Arts \end{array} \\ \begin{bmatrix} 90 & 60 & 90 \\ 90 & 90 & 30 \\ 60 & 60 & 60 \\ 60 & 60 & 90 \\ 30 & 30 & 30 \end{bmatrix} \end{array}$$

Its covariance matrix would be (Cov(A))

$$\begin{array}{c} \\ Math \\ English \\ Arts \end{array} \begin{array}{c} \begin{array}{ccc} Math & English & Arts \end{array} \\ \begin{bmatrix} 504 & 360 & 180 \\ 360 & 360 & 0 \\ 180 & 0 & 720 \end{bmatrix} \end{array}$$

A few points that can be noted here are:

- Shown along the diagonal, we see the variance of scores for each test. The art test has the most significant variance (720), and the English test has the most minor variance (360). Art test scores have more variability than English test scores.
- The covariance is displayed in black in the off-diagonal elements of the matrix A

a) The covariance between math and English is positive (360), and the covariance between math and art is positive (180). This means the scores tend to covary positively. As scores in math go up, scores in art and English also tend to go up, and vice versa.

b) The covariance between English and art, however, is zero. This means there tends to be no predictable relationship between the movement of English and art scores.

MATHEMATICS BEHIND PCA

4. Compute Eigenvectors and corresponding Eigenvalues

Intuitively, an eigenvector is a vector whose direction remains unchanged when a linear transformation is applied to it.

We can easily compute eigenvalue and eigenvectors from the covariance matrix above.

Let A be a square matrix, v a vector, and λ a scalar that satisfies Av = λv, then λ is called the eigenvalue associated with eigenvector v of A.

The eigenvalues of A are roots of the characteristic equation.

$$det\left(A-\lambda I\right)$$

Calculating det(A-λI) first, I is an identity matrix :

$$det\left(\begin{bmatrix}504 & 360 & 180\\ 360 & 360 & 0\\ 180 & 0 & 720\end{bmatrix}-\lambda\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\right)$$

Simplifying the matrix first, we can calculate the determinant later,

$$\begin{bmatrix}504 & 360 & 180\\ 360 & 360 & 0\\ 180 & 0 & 720\end{bmatrix}-\begin{bmatrix}\lambda & 0 & 0\\ 0 & \lambda & 0\\ 0 & 0 & \lambda\end{bmatrix}$$

$$\begin{bmatrix}504-\lambda & 360 & 180\\ 360 & 360-\lambda & 0\\ 180 & 0 & 720-\lambda\end{bmatrix}$$

Now that we have our simplified matrix, we can find the determinant of the same :

$$det\left(\begin{bmatrix}504-\lambda & 360 & 180\\ 360 & 360-\lambda & 0\\ 180 & 0 & 720-\lambda\end{bmatrix}\right)=-\lambda^3+1584\lambda^2-641520\lambda+25660800$$

MATHEMATICS BEHIND PCA

We now have the equation, and we need to solve for λ to get the matrix's eigenvalue. So, equating the above equation to zero :

$$-\lambda^3 + 1584\lambda^2 - 641520\lambda + 25660800 = 0$$

After solving this equation for the value of λ, we get the following values of eigenvalues:

$$\lambda \approx 44.81966\ldots, \lambda \approx 629.11039\ldots, \lambda \approx 910.06995\ldots$$

Now, we can calculate the eigenvectors corresponding to the above eigenvalues.
So, after solving for eigenvectors, we would get the following solution for the corresponding eigenvalues:

$$\begin{bmatrix} -3.75100\ldots \\ 4.28441\ldots \\ 1 \end{bmatrix}, \begin{bmatrix} -0.50494\ldots \\ -0.67548\ldots \\ 1 \end{bmatrix}, \begin{bmatrix} 1.05594\ldots \\ 0.69108\ldots \\ 1 \end{bmatrix}$$

5. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a d × k dimensional matrix W.

We started with the goal of reducing the dimensionality of our feature space, i.e., projecting the feature space via PCA onto a smaller subspace, where the eigenvectors will form the axes of this new feature subspace. However, the eigenvectors only define the directions of the new axis since they all have the same unit length. 1.

So, to decide which eigenvector(s) we want to drop for our lower-dimensional subspace, we have to look at the eigenvectors' corresponding eigenvalues. Roughly speaking, the eigenvectors with the lowest eigenvalues bear the least information about the data distribution, and those are the ones we want to drop.

The standard approach is to rank the eigenvectors from highest to lowest corresponding eigenvalue and choose the top k eigenvectors.

MATHEMATICS BEHIND PCA

So, after sorting the eigenvalues in decreasing order, we have:

$$\begin{bmatrix} 910.06995 \\ 629.11039 \\ 44.81966 \end{bmatrix}$$

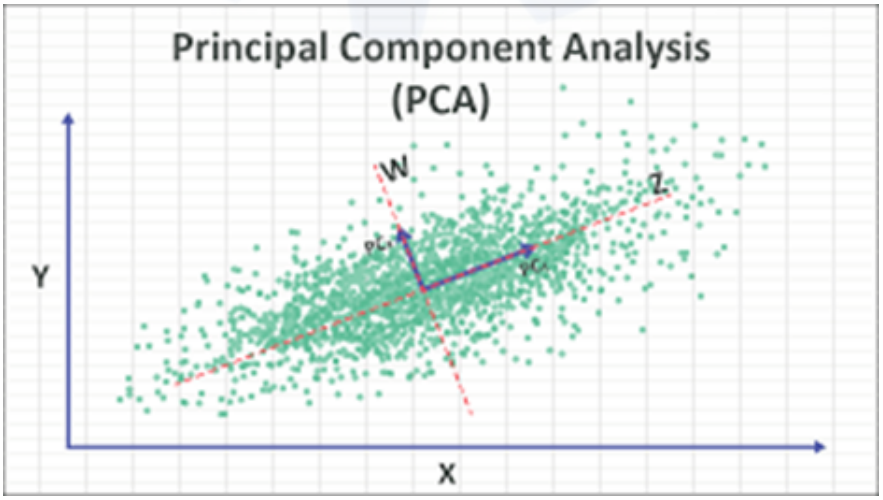
For our simple example, where we are reducing a 3-dimensional feature space to a 2-dimensional feature subspace, we combine the two eigenvectors with the highest eigenvalues to construct our d×k dimensional eigenvector matrix W.

So, eigenvectors corresponding to two maximum eigenvalues are :

$$W = \begin{bmatrix} 1.05594 & -0.50494 \\ 0.69108 & -0.67548 \\ 1 & 1 \end{bmatrix}$$

6. Transform the samples onto the new subspace

In the last step, we use the 3x2 dimensional matrix W that we just computed to transform our samples onto the new subspace via the equation y = W′ × x where W′ is the transpose of the matrix W.
Lastly, we have computed our two principal components and projected the data points onto the new subspace.



ACKNOWLEDGEMENTS

Pradyuman Agarwal
Aansh Samyani
Aditi Agrawal
Keshav Maheshwari
Yash Shah
Siddharth Acharya
Kalash Shah
Sharayu Korade
Unnati Agarwal

We thank you for your contribution to the making of Machine Learning Booklet

Contact

Analytics Club

Aansh Samyani
7021068263

Aditi Agrawal
9977104908

analytics.convener@gmail.com