

Machine Learning

Machine learning is a subset of artificial intelligence (AI) that focuses on developing computer systems capable of learning and improving from data without being explicitly programmed. Machine learning is teaching computers to learn from data and make decisions or predictions based on that learning.

Machine learning is widely used in various fields and applications such as:

- Face Recognition
- Object Detection
- Chatbots
- Recommendation Systems
- Autonomous Vehicles
- Disease Diagnosis
- Fraud detection

And many more it is nowadays being used in almost all sectors including healthcare, education, business, construction, astronomy, etc.

Types of Machine Learning:

There are many types of machine learning but the most famous types are:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Supervised Learning

In supervised learning the algorithm is trained on labeled data which means training data includes the input and output pairs. The goal is to learn the mapping from input to output.

$$X \rightarrow Y$$

Some examples of supervised learning include image classification, email spam detection, and predicting software.

Types of supervised learning tasks:

1. Regression
2. Classification

Regression:

Regression is a type of supervised learning task where the algorithm's goal is to predict a continuous numerical output or target variable. In regression, the output is a real-valued number, and the algorithm's objective is to learn a mapping from input features to this continuous output.

Examples of regression tasks include home price prediction, black hole mass prediction, stock price prediction, age estimation, etc.

Algorithms:

❖ Linear Regression:

Linear Regression is a fundamental statistical and machine-learning technique for solving regression problems used for modeling the relationship between a dependent variable (or target) and one or more

independent variables (or features). It assumes that this relationship is approximately linear, meaning that changes in the independent variables have a linear effect on the dependent variable.

➤ **Simple Linear Regression:**

In simple linear regression, there is only one input feature.

$$Y = wx + b$$

Y = target variable

W = slope of straight line

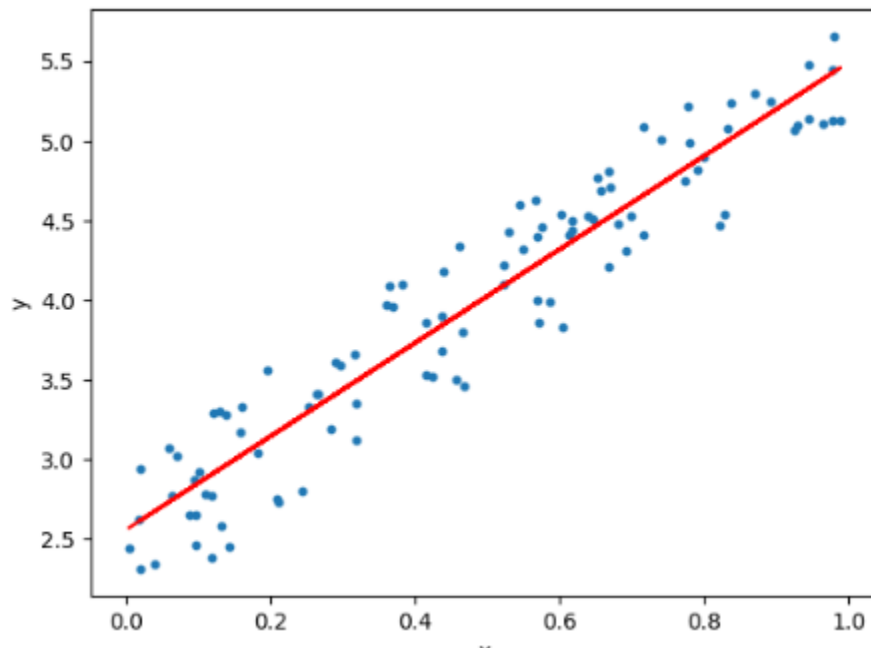
X = input features

b = y-intercept

➤ **Multiple Linear Regression:**

In multiple linear regression, there is more than one input feature.

$$Y = w_1x_1 + w_2x_2 + w_3x_3 + \dots + b$$



Note: There are 2 common issues in machine learning and statistical modeling that are **underfitting** and **overfitting**.

Underfitting:

Underfitting occurs when a model is too simple to capture the underlying patterns in the data, both in the training set and unseen data. It essentially means the model is not complex enough to represent the relationships between the input features and the target variable. This means it is not predicting the output correctly and the accuracy of the model is very low. Solutions to overcome this problem are:

- 1) Increase model complexity.
- 2) Add more features.
- 3) Feature engineering (It is basically creating the new input feature using the existing input features For example for home price prediction you are given the width and length of a home you can simply create another feature named area of home by multiplying the width and length of home you already know).
- 4) Remove noise (clean data to remove outliers).
- 5) Increase training data.
- 6) Increase training data.
- 7) Reduce regularization.
- 8) Iterating the optimization algorithms like gradient descent for more times.

Overfitting:

Overfitting occurs when a model is excessively complex and fits the training data noise rather than the underlying patterns. It means the model is too flexible and essentially memorizes the training data rather than generalizing it. Simply we can say that in this case the cost function (The cost function quantifies the error between predicted and expected values and presents that error in the form of a single real number.) is very low close to 0 and the accuracy of the model is around 100%. Solutions to overcome this problem are:

- 1) Collect more training data
- 2) Feature selection
- 3) Feature engineering

4) Do regularization.

Regularization:

Regularization is a technique used in machine learning and statistics to prevent overfitting, which occurs when a model fits the training data too closely, capturing noise and making it less effective at making predictions on new, unseen data. Regularization adds a penalty term to the model's loss function, discouraging it from fitting the training data too precisely and encouraging it to find a simpler, more generalizable solution. There are 2 of its common techniques.

A. L1 regularization (lasso):

L1 regularization adds the absolute values of the model's parameters to the loss function. Lasso is useful for feature selection and simplifying models.

Transforming the Loss function into Lasso Regression

$$\sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \Rightarrow \sum_{i=1}^M \left(y_i - \sum_{j=0}^p w_j \times x_{ij} \right)^2 + \lambda \sum_{j=0}^p |w_j|$$

Loss function Loss function + Regularized term

B. L2 regularization (ridge):

L2 regularization adds the squared values of the model's parameters to the loss function. Ridge helps reduce model complexity and is effective when all features are potentially relevant.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{\theta}(x^{(i)}), y^{(i)}) + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

Note: **Learning rate** is a hyperparameter in machine learning algorithms, particularly in optimization algorithms like gradient descent. It determines the size of steps that the algorithm takes when adjusting the model's parameters during

training. Choosing a small learning rate can lead you to converge to the solution slowly and by choosing a large value of learning rate the solution does not converge to a point.

Validation set in Machine learning:

A validation set is a portion of the dataset used in machine learning to assess the performance of a trained model. It's like a practice exam for the model before it faces the real test (the test set). The validation set is not part of the training data, nor is it part of the final evaluation data (the test set). Instead, it serves as a middle ground for testing the model's performance during training. It helps you make decisions about how well your model is learning and whether it's overfitting or underfitting. During the training process, after each training epoch (iteration), the model's performance is evaluated on the validation set. The model's predictions on the validation set are compared to the actual target values, and a performance metric (like accuracy, loss, or other relevant metrics) is computed. This metric helps you understand how well the model is doing on data it hasn't seen during training. You can adjust hyperparameters (like learning rate, model architecture, or regularization strength) based on the model's performance on the validation set. For example, if the model's performance on the validation set starts to degrade, you may want to reduce the learning rate or simplify the model.

Feature Scaling in Machine Learning:

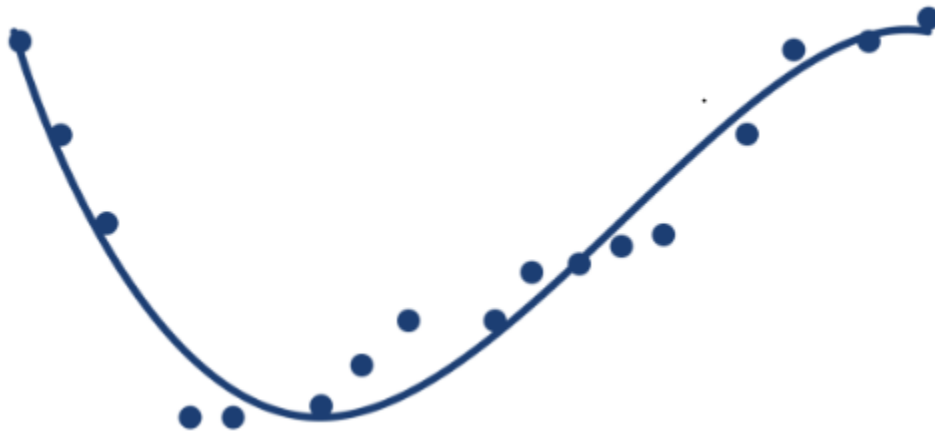
Feature scaling is a preprocessing technique in machine learning that helps bring all the features (variables) of your dataset onto a similar scale. It ensures that no single feature dominates the learning process because of its larger magnitude. Different features in your dataset may have values on different scales. For example, one feature might range from 0 to 1, while another might range from 0 to 1000. Feature scaling is used to make sure that these varying scales do not affect the performance of machine learning algorithms. Feature scaling involves transforming the values of each feature so that they fall within a similar numerical range. The common techniques used to do feature scaling are min-max scaling and z-score. Feature scaling ensures that all features contribute equally to the learning process, preventing some features from having undue influence due to

their larger values. It can lead to faster convergence of gradient-based optimization algorithms (like gradient descent).

❖ **Polynomial Regression:**

Polynomial regression is a type of regression analysis used in machine learning and statistics to model relationships between variables when the traditional linear regression model is insufficient. It extends the concept of linear regression by allowing for more complex, nonlinear relationships between the independent and dependent variables.

$$y = b_0 + b_1x + b_2x^2 + \dots + b_nx^n$$



The degree of the polynomial (n) is a hyperparameter that you can choose based on the complexity of the relationship you want to capture.

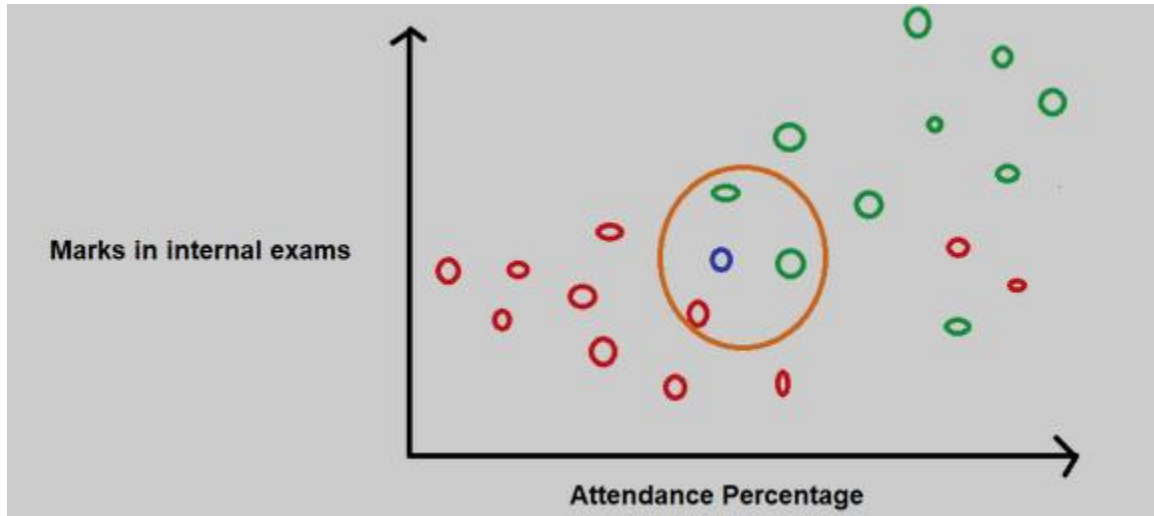
A higher degree allows the model to fit the data more closely but may also lead to overfitting if not chosen carefully.

In order if overfitting occurs then reduce the degree of polynomial, collect more data do regularization.

❖ **KNN regression:**

K-Nearest Neighbors (KNN) regression is a simple and intuitive machine learning algorithm used for regression tasks. It's an instance-based learning method that makes predictions by considering the average (or weighted

average) of the target values of its k-nearest neighbors in the training data. KNN regression is used to predict a continuous target variable (numeric value) based on the values of its neighboring data points. During training, the KNN algorithm stores the entire training dataset.

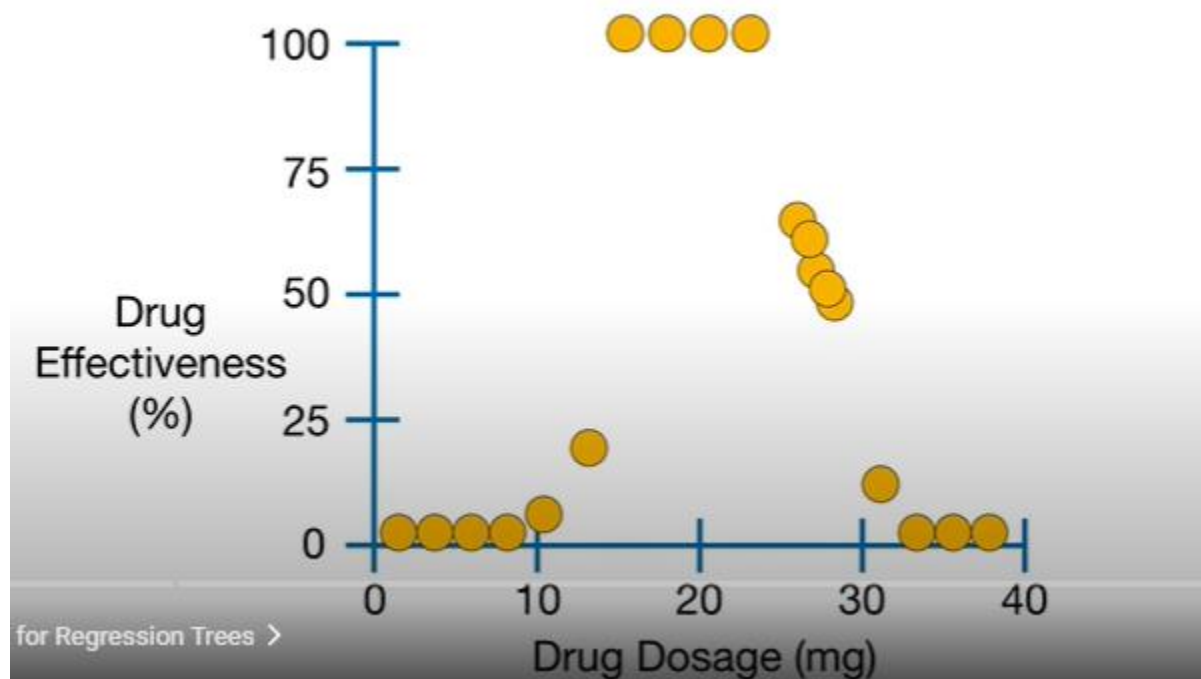


To make a prediction for a new data point, the algorithm identifies the k-nearest data points (neighbors) in the training set based on a distance metric (usually Euclidean distance). K is a hyperparameter here. The distance is calculated between the entered point from all points in the whole data set then the data set is sorted based on distance in ascending order and picked the first k rows and the mean of target variables are calculated which is the answer. The value of k is a hyperparameter that you need to specify when using KNN regression. A smaller k (e.g., 1 or 3) makes the model sensitive to noise in the data and can result in a more variable prediction. A larger k (e.g., 10 or 20) provides a smoother prediction but might not capture local patterns as effectively. KNN regression is simple to understand and implement. It can capture complex and nonlinear relationships between features and the target variable. Choosing the appropriate value of k is crucial and can be challenging. KNN can be computationally expensive when the dataset is large, as it requires calculating distances to all data points during prediction. KNN regression works well for small to moderately-sized datasets with a reasonable number of features. For large datasets, the computational cost of finding nearest neighbors can become prohibitive. If overfitting occurs in KNN regression

then do better feature selection, adjust the value of k, do feature selection etc.

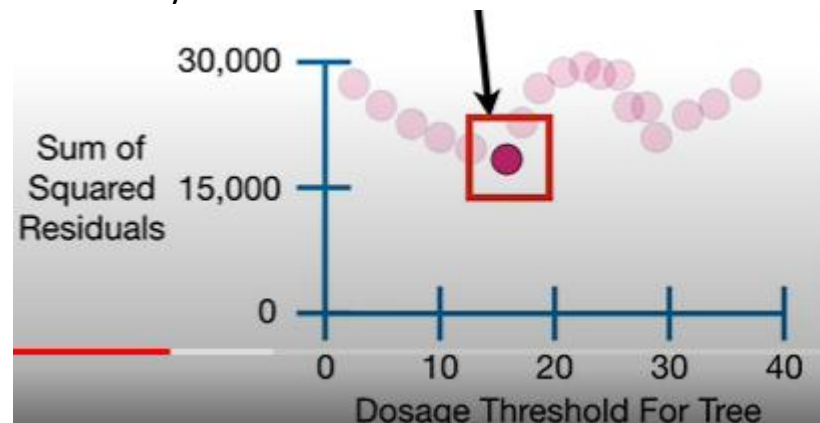
❖ **Decision Tree Regressor/ Regression tree:**

A regression tree is a type of decision tree used in machine learning for regression tasks. It's a tree-like structure that makes predictions about continuous numeric values. Consider a scenario in which you are trying to predict the drug usage effectiveness prediction.

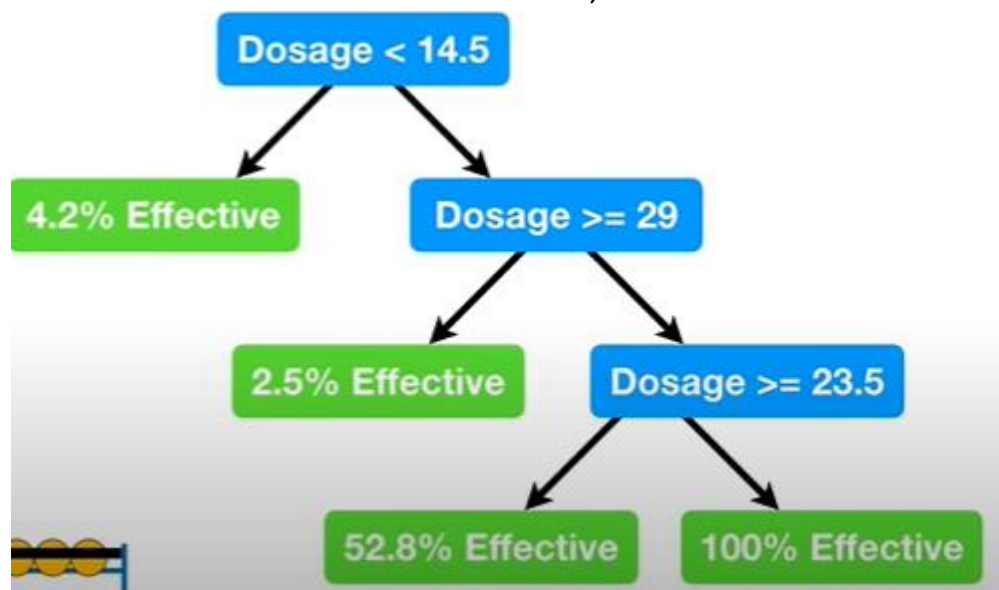


Now we have to build a tree and how should be built. Like what value of drug dosage should be the root node here is what we should know about the square sum residual. In the diagram let's focus on small 2 values and take an average of it that will be 3. Now imagine the node has a drug dosage of less than 3. Now for all data below 3 calculate the average value of effectiveness of all and also calculate the average value of effectiveness on data set above 3. Calculate the Squared sum residual in case of saving it. Now focus on the next 2 data points calculate the average of and consider the root to be valued less than 5 and do the same that we did for case of

drug dosage less than 3 after that take the next data sets and do same. Now check for the residuals you stored.



As seen select the dosage value with less residual value that is 14.5. Now root node will have a drug dosage of less than 14.5. Now after that, if the data sets less than 14.5 are really less values then no need to further split it just calculate their average value and make it a leaf node. For the data set above 14.5 split it into other nodes using the same concept that we did earlier selecting the threshold value calculating the residuals and selecting one with less residual. The final tree will be like,



Here the scenario we discussed has only one input feature. If there are multiple features, calculate the residuals for each feature and then select one with less value of residual and make the tree. They are easy to

understand and able due to their tree-like structure. It can model complex, nonlinear relationships in data. Outliers have minimal impact on model performance. But it is Sensitive to small changes in the data. If overfitting happens in the regression tree then reduce the depth of the tree and do pruning.

Pruning:

Pruning is a technique used in decision tree-based models, including regression trees, to prevent overfitting and improve model generalization. Pruning involves cutting back or removing some branches (subtrees) of a decision tree after it has been fully grown. It aims to simplify the tree by removing branches that capture noise or fine-grained details in the training data. Pruning techniques consider a cost-complexity trade-off: They evaluate the cost (error) associated with keeping or removing each subtree and select the option that minimizes this cost. Pruned subtrees are replaced with a single leaf node, often representing the average or majority class (for classification) or the average value (for regression) of the training data in that subtree.

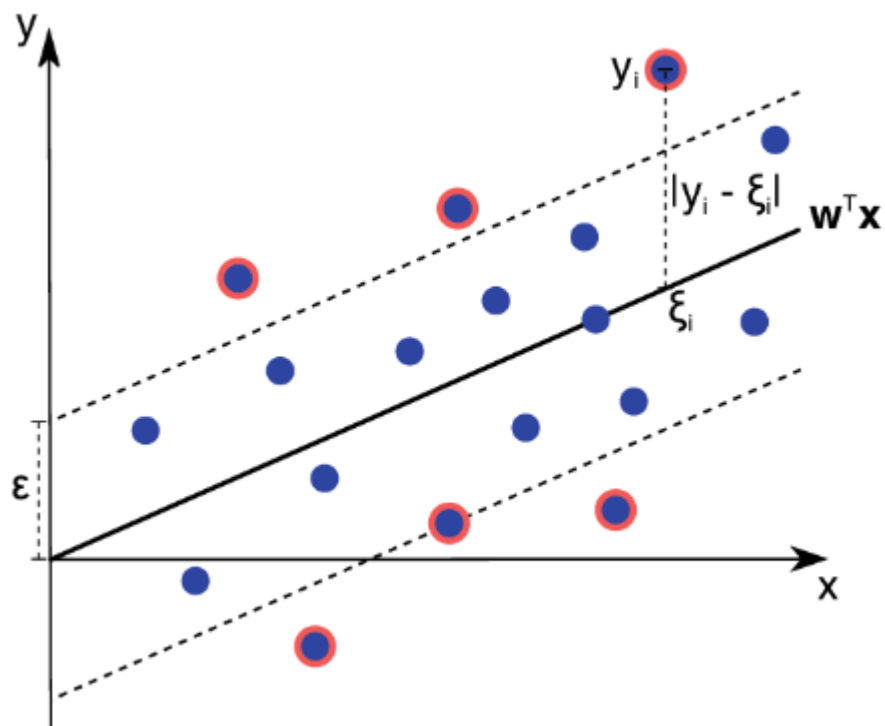
❖ Random Forest Regression:

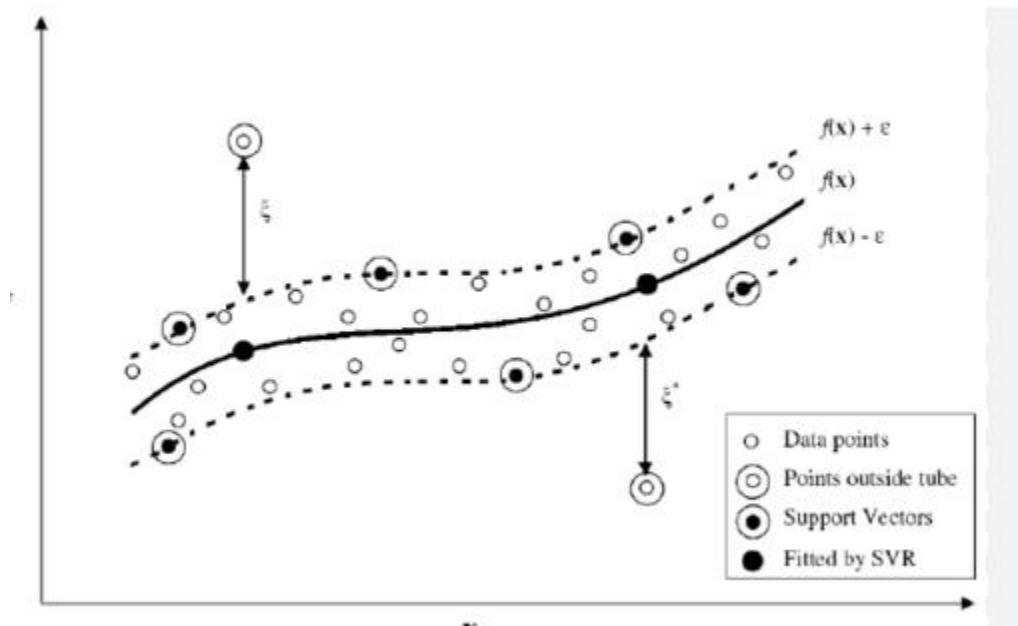
Random Forest Regression is a machine learning method for predicting continuous values, like house prices or temperatures. Random Forest Regression is like having a group of decision trees who collaborate to make predictions. When making a prediction, the Random Forest collects opinions from all the decision trees. It combines these opinions by averaging their predictions to arrive at a final, more accurate prediction. If overfitting happens in random forest regression then reduce the number of trees, feature selection, pruning, etc. Random Forest Regression is less prone to overfitting compared to a single Decision Tree. Random Forests often provide higher predictive accuracy compared to a single Decision Tree. Random Forests are more stable and less sensitive to small variations in the training data. While on the other hand, Random Forests are more complex than individual Decision Trees. Random Forests typically require

more computational resources and training time than a single Decision Tree.

❖ Support Vector Regressor:

SVR is a machine learning algorithm used for regression tasks. It's an extension of Support Vector Machines (SVMs) that were originally designed for classification. The data points outside the tube of ϵ (eta) to the regression line. They have the most influence on the positioning of the regression line these data points are called support vectors. So, in SVR we also draw a marginal line and our goal is to maximize the marginal line so maximum data points lie inside the marginal lines.





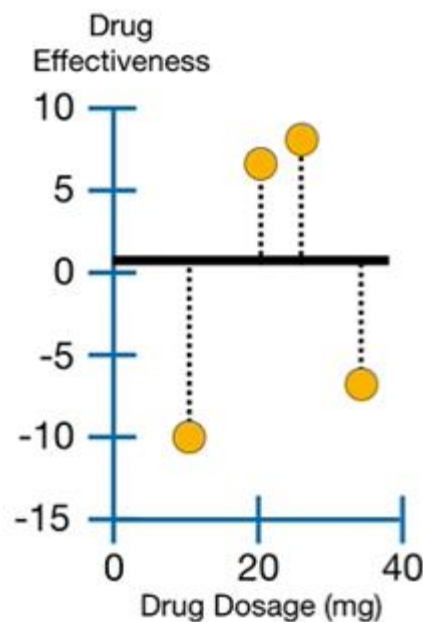
So the cost function will be

$$\frac{\|w\|^2}{2} + C \sum_{i=1}^n |\xi_i|$$

And our aim is to minimize the cost function. Where c controls how many data points can be allowed to exist outside the margin (the region around the regression line). so a smaller value of c means that there are fewer values outside the margin line and model accuracy will be high and a high value of c that there will be more points outside the margin line and accuracy will be low. While other parameter tells us about the distance between the data point outside the marginal line and the marginal line. So here is the scenario we reduce the cost function in the way and make a model. So in SVR kernel is basically a function that transforms the data into higher dimensions in order to build a model SVR that can handle outliers effectively. Training an SVR model can be computationally expensive, especially on very large datasets. In order to avoid overfitting in the Support vector regressor we select the proper value of hyperparameters, feature selection, feature scaling, regularization, and kernel selection.

❖ XGBoost Regression:

XGBoost (eXtreme Gradient Boosting) is a powerful and popular machine-learning algorithm known for its efficiency and high performance across a wide range of tasks. It belongs to the ensemble learning category, which means it combines the predictions of multiple base models (often decision trees) to improve overall performance. Suppose we have this data, and we want to predict the drug effectiveness.



The first step in fitting XGBoost to training data is to make an initial prediction which is basically the mean of the target variable in the training data set. In the above example, the initial prediction is 0.5. Then we calculate the residuals which is the difference between the observed and predicted value. So according to the above data set the residual values are:

-10.5, 6.5, 7.5, -7.5

Now we do calculations for similarity gain and the formula for this is,

$$\text{Similarity Score} = \frac{\text{Sum of Residuals, Squared}}{\text{Number of Residuals} + \lambda}$$

Where lambda is a hyperparameter and we are assuming it is 0 here. Now we calculate the similarity gain of the root that is 0. Now for splitting checks which will be better. So, for this, we will take an average of the first 2 data sets and that is the average whose value is 15. Now make dosage less than 15 a root node and we will calculate a gain first we will calculate the similarity score of each node.



$$\text{Similarity Score} = \frac{-10.5^2}{1 + 0}$$

...and the **Similarity Score** for the leaf on the left = **110.25**.

$$\text{Similarity Score} = \frac{6.5^2}{3 + 0}$$

Thus, the **Similarity Score** for the **Residuals** in the leaf on the right = **14.08**.

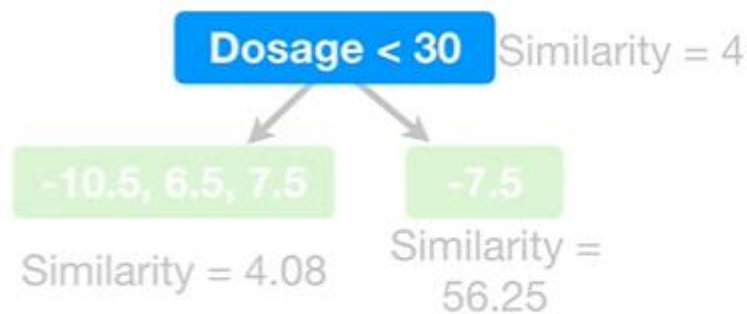
We calculated the similarity score now we will calculate the gain.

$$\text{Gain} = \text{Left}_{\text{Similarity}} + \text{Right}_{\text{Similarity}} - \text{Root}_{\text{Similarity}}$$

So the gain is 120.33. We calculated the gain for a threshold dosage of less than 15. Now we will take the other 2 data sets and calculate an average that is 22.5 Now we will calculate the gain for 22.5.



The gain of this node is 4. Now we will take other values and compute the average for new threshold value that is dosage less than 30.

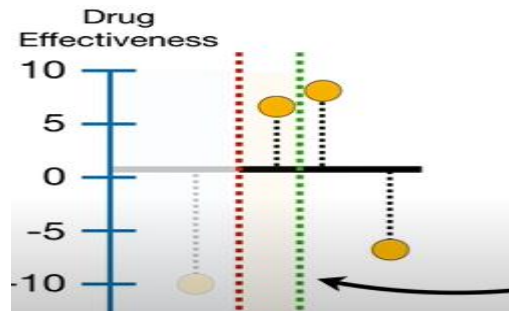


$$\text{Gain} = 4.08 + 56.25 - 4 = 56.33$$

Now we will select the threshold that will calculate the largest gain that is dosage less than 15.



Now as in left node there is only one residual we cannot split it further but we can split the right node. Now we will again do the same procedure as explained above. Now we will look into data for values greater than 15. The next node is less than 22.5.

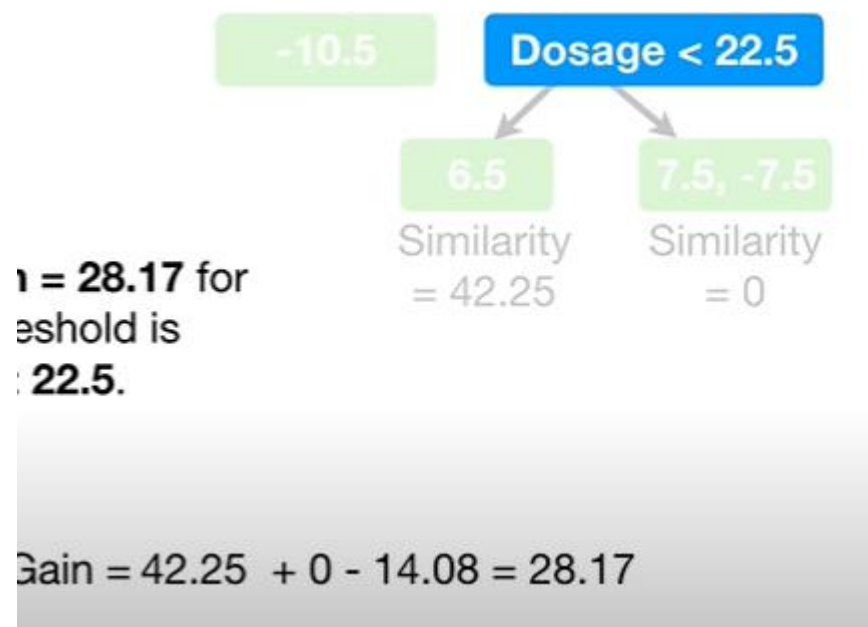


We calculate the similarity score of node.

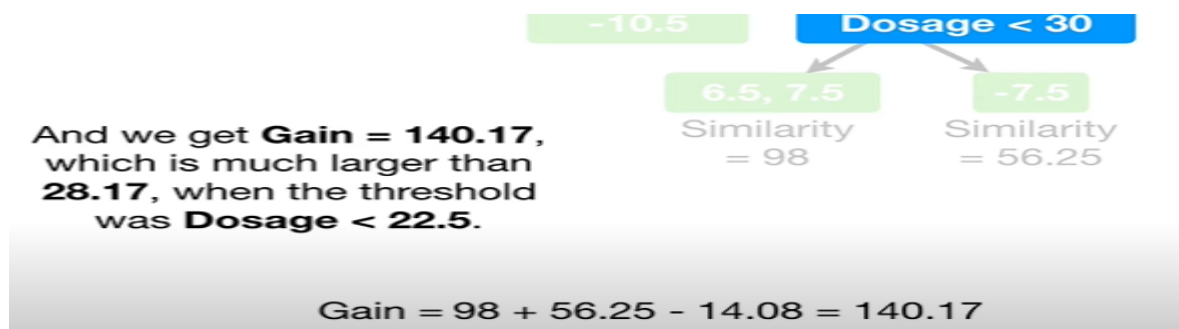
< 15 Similarity = 14.08

6.5, 7.5, -7.5

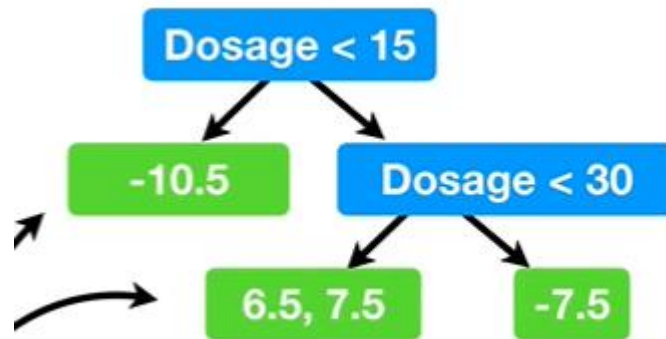
And gain for the node is below.



Now we calculate the gain of next threshold. That is,



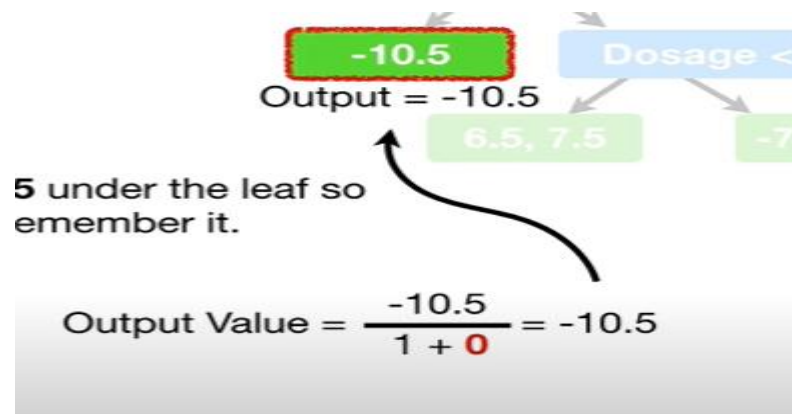
As a dosage less than 30 has a high value of gain we will select this as a node.

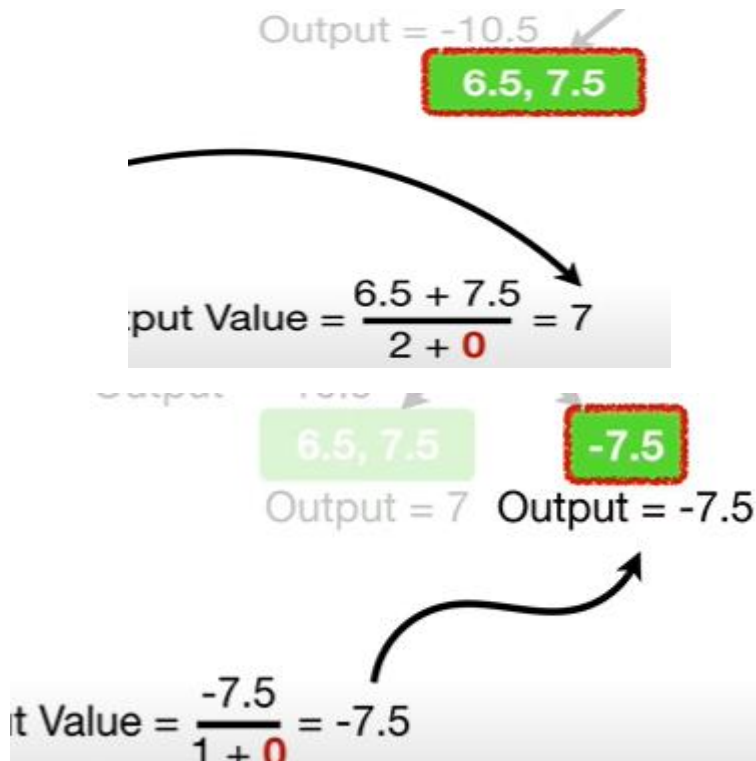


Now let's discuss the pruning of a tree for that we use a hyperparameter called gamma. Pruning is done based on the value of gain whether the subtree has to be removed or not. So if the difference between gain and gamma is negative we remove the subtree we do not remove it. Now if we consider the value of gamma as 130 initially and do $140.17 - 130$ answer will be positive so we will not remove the subtree. For the root node, the difference is negative but we will not remove it because we haven't removed the child nodes of the root node. Here is how pruning is done. We can also set the limit of the depth of a tree. Now we will calculate the output value of each leaf node.

$$\text{Output Value} = \frac{\text{Sum of Residuals}}{\text{Number of Residuals} + \lambda}$$

let $\lambda = 0$.



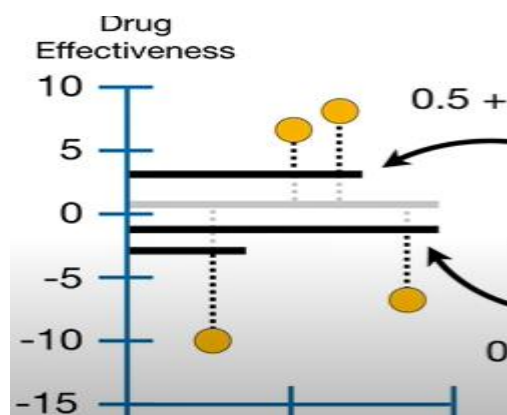


Now we can use the above tree to make new predictions. For that the formula is,
 Predicted value initial prediction + (learning rate * output value)

So for data 13 dosages, the predicted value is,

$$0.5 + (0.3 \times -10.5) = -2.65$$

For a dosage equal to 20, the predicted value is 2.6. Now we will repeat the steps with all values.



Now we will build another tree based on new residuals and make new predictions that give us even smaller residuals. Values of predicted values that we got from the first tree then we used these values and calculated their mean which is used as the initial predictor for building the second tree and so on. This algorithm predicts highly accurate values. Training an XGBoost model with a large number of trees and deep trees can be resource-intensive in terms of memory and computation. It is applicable to a wide range of data sets. If overfitting occurs then we do limit the number of trees, limit the depth of trees, etc.

Classification:

Classification in supervised machine learning is like teaching a computer to recognize and sort things into different groups based on their unique characteristics. It's like how we classify objects in our daily lives.

Example: Disease detection etc.

Algorithms:

❖ Logistic Regression:

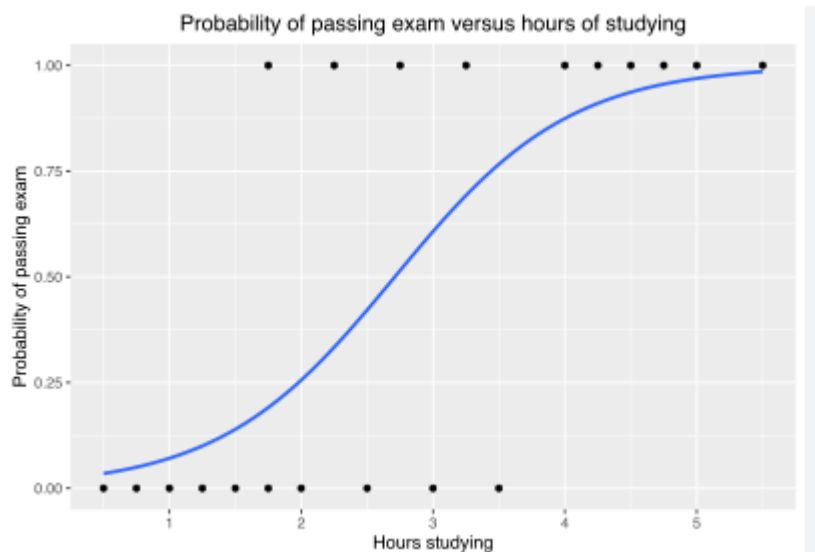
Logistic regression is a type of statistical model used for classification tasks in machine learning. It's particularly useful when the target variable (what you're trying to predict) is categorical. This means it can have only two possible outcomes, such as "yes" or "no", "spam" or "not spam", etc. Unlike linear regression, where the output can be any real number, logistic regression outputs probabilities. These probabilities are constrained to be between 0 and 1. Logistic regression uses the logistic function (also known as the sigmoid function) to model the relationship between the features and the probability of a specific outcome. The decision boundary is a threshold value that separates the classes. If the predicted probability is greater than the threshold, it assigns the data point to one class, otherwise to the other. The most common loss function used in logistic regression is

the log-likelihood loss, which measures the difference between predicted probabilities and actual outcomes.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{\theta}(x^{(i)}), y^{(i)})$$

$$J(\theta) = \frac{1}{m} \left[\sum_{i=1}^m -y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right]$$

m = number of samples

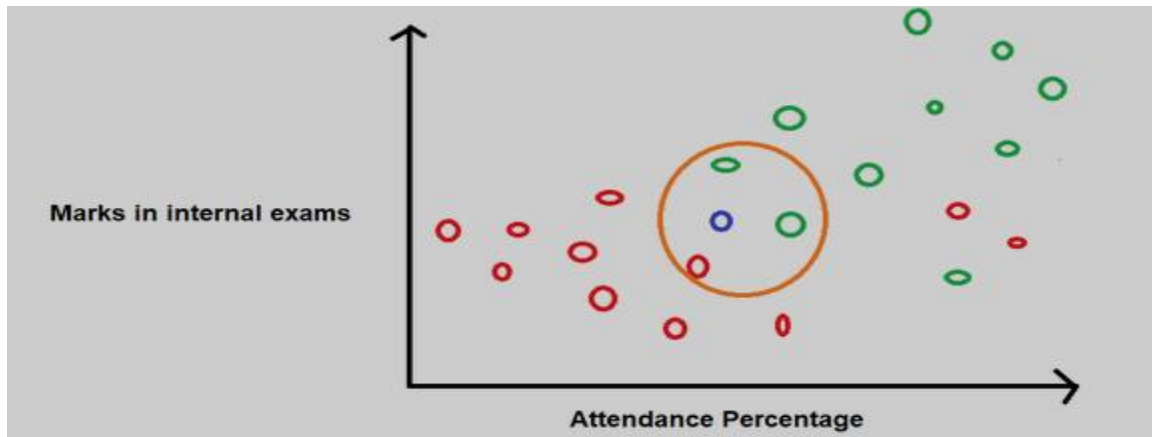


This model is well used for binary classification. So in order to avoid overfitting in logistic regression we do regularization.

❖ **KNN:**

K-Nearest Neighbors (KNN) regression is a simple and intuitive machine learning algorithm used for classification tasks. It's an instance-based learning method that makes predictions by considering the average (or weighted average) of the target values of its k-nearest neighbors in the

training data. During training, the KNN algorithm stores the entire training dataset.



To make a prediction for a new data point, the algorithm identifies the k-nearest data points (neighbors) in the training set based on a distance metric (usually Euclidean distance). K is a hyperparameter here. The distance is calculated between the entered point from all points in the whole data set then the data set is sorted based on distance in ascending order and picked the first k rows and the mode of target variables are calculated which is the answer. The value of k is a hyperparameter that you need to specify. A smaller k (e.g., 1 or 3) makes the model sensitive to noise in the data and can result in a more variable prediction. A larger k (e.g., 10 or 20) provides a smoother prediction but might not capture local patterns as effectively. KNN classifier is simple to understand and implement. It can capture complex and nonlinear relationships between features and the target variable. Choosing the appropriate value of k is crucial and can be challenging. KNN can be computationally expensive when the dataset is large, as it requires calculating distances to all data points during prediction. KNN classifier works well for small to moderately-sized datasets with a reasonable number of features. For large datasets, the computational cost of finding nearest neighbors can become prohibitive. If overfitting occurs in KNN regression then do better feature selection, adjust the value of k, do feature selection etc.

❖ Naïve Bayes Algorithm:

Naive Bayes is a simple and intuitive machine learning algorithm used for classification tasks. It's based on Bayes' theorem and assumes that the features used to make predictions are independent of each other (which is why it's called "naive"). Naive Bayes relies on Bayes' theorem.

| Day | Outlook | Temperature | Humidity | Wind | PlayTennis |
|-----|----------|-------------|----------|--------|------------|
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | Hot | High | Weak | Yes |
| D4 | Rain | Mild | High | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | Yes |
| D6 | Rain | Cool | Normal | Strong | No |
| D7 | Overcast | Cool | Normal | Strong | Yes |
| D8 | Sunny | Mild | High | Weak | No |
| D9 | Sunny | Cool | Normal | Weak | Yes |
| D10 | Rain | Mild | Normal | Weak | Yes |
| D11 | Sunny | Mild | Normal | Strong | Yes |
| D12 | Overcast | Mild | High | Strong | Yes |
| D13 | Overcast | Hot | Normal | Weak | Yes |
| D14 | Rain | Mild | High | Strong | No |

Consider above is a data set and we have to use naïve Bayes to predict whether we can play tennis or not. So we first calculate the prior probabilities based on the given data what is the probability that yes we can play tennis and no we cannot? After that, we calculate the conditional probabilities of all input features as shown below.

$$P(\text{PlayTennis} = \text{yes}) = 9/14 = .64$$

$$P(\text{PlayTennis} = \text{no}) = 5/14 = .36$$

| Outlook | Y | N | Humidity | Y | N |
|-------------|-----|-----|----------|-----|-----|
| sunny | 2/9 | 3/5 | high | 3/9 | 4/5 |
| overcast | 4/9 | 0 | normal | 6/9 | 1/5 |
| rain | 3/9 | 2/5 | | | |
| Temperature | | | Windy | | |
| hot | 2/9 | 2/5 | Strong | 3/9 | 3/5 |
| mild | 4/9 | 2/5 | Weak | 6/9 | 2/5 |
| cool | 3/9 | 1/5 | | | |

Then using the above data we do calculations for the testing data. So we have to do a prediction for the following data.

{Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong}

$$v_{NB}(yes) = P(yes) P(sunny|yes) P(cool|yes) P(high|yes) P(strong|yes) = .0053$$

$$v_{NB}(no) = P(no) P(sunny|no) P(cool|no) P(high|no) P(strong|no) = .0206$$

As the probability of no is higher than yes we will predict that tennis cannot be played. This is an example of multinomial naïve bayes.

Now in the above example, all input features were discrete what if the input features have continuous values then we cannot calculate the conditional probability We will use Gaussian Naïve Bayes.

| Person | Height (ft) | Weight (lbs) | Foot size (inches) |
|--------|-------------|--------------|--------------------|
| Male | 6.00 | 180 | 12 |
| Male | 5.92 | 190 | 11 |
| Male | 5.58 | 170 | 12 |
| Male | 5.92 | 165 | 10 |
| Female | 5.00 | 100 | 6 |
| Female | 5.50 | 150 | 8 |
| Female | 5.42 | 130 | 7 |
| Female | 5.75 | 150 | 9 |

We have to predict using data that the person is male or female so we will calculate the mean and standard deviation of each input feature for male and female and first we calculate the prior probabilities.

$$P(Male) = 4/8 = 0.5$$

$$P(Female) = 4/8 = 0.5$$

Male:

$$\text{Mean (Height)} = \frac{(6+5.92+5.58+5.92)}{4} = 5.855$$

$$\begin{aligned} \text{Variance (Height)} &= \frac{\sum (x_i - \bar{x})^2}{n-1} \\ &= \frac{(6-5.855)^2 + (5.92-5.855)^2 + (5.58-5.855)^2 + (5.92-5.855)^2}{4-1} \\ &= 0.035055 \end{aligned}$$

Now we will calculate for all

| Sex | Mean (height) | Variance (height) | Mean (weight) | Variance (weight) | Mean (foot size) | Variance (foot size) |
|--------|---------------|-------------------|---------------|-------------------|------------------|----------------------|
| Male | 5.855 | 0.035033 | 176.25 | 122.92 | 11.25 | 0.91667 |
| Female | 5.4175 | 0.097225 | 132.5 | 0558.33 | 7.5 | 1.6667 |

Now we have to predict whether a person is male or female based on the following data

| Sex | Height(ft) | Weight(lbs) | Foot size(inch) |
|--------|------------|-------------|-----------------|
| Sample | 6 | 130 | 8 |

For that, we can use the Gaussian distribution equation.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Using Gaussian distribution we calculate the following values

$$\begin{aligned} P(H|M) &= \frac{1}{\sqrt{2 * 3.142 * 0.035033}} * e^{-\frac{(6-5.855)^2}{2 * 0.035033}} = 1.5789 & P(H|F) &= 2.2346e^{-1} \\ P(W|M) &= 5.9881e^{-6} & P(W|F) &= 1.6789e^{-2} \\ P(FS|M) &= 1.3112e^{-3} & P(FS|F) &= 2.8669e^{-1} \\ \text{Posterior (Male)} &= \frac{P(M) * P(H|M) * P(W|M) * P(FS|M)}{\text{Evidence}} = 0.5 * 1.5789 * 5.9881e^{-6} * 1.3112e^{-3} = 6.1984e^{-9} \\ \text{Posterior (Female)} &= \frac{P(F) * P(H|F) * P(W|F) * P(FS|F)}{\text{Evidence}} = 0.5 * 2.2346e^{-1} * 1.6789e^{-2} * 2.8669e^{-1} = 5.377e^{-4} \end{aligned}$$

As the posterior probability of female is more so we will predict that person will be female. Naive Bayes is computationally efficient. It can handle a large number of features, making it suitable for high-dimensional

data sets. Sometimes we face an issue called zero probability issue and to deal with that we use Laplace smoothing.

| Outlook | Yes | No |
|----------|-----|-----|
| Sunny | 2/9 | 3/5 |
| Overcast | 4/9 | 0/5 |
| Rainy | 3/9 | 3/5 |

You can see for overcast | No the probability is zero.

$$P_{LAP,k}(x|y) = \frac{c(x, y) + k}{c(y) + k|X|}$$

Here,

k represents the smoothing parameter (greater than zero),

X represents the number of dimensions (features) in the data

Now probability will change from 0 to some positive value.

$$p(\text{outlook} = \text{overcast}|\text{no}) = \frac{0 + 1}{5 + 1 * 3} = \frac{1}{8}$$

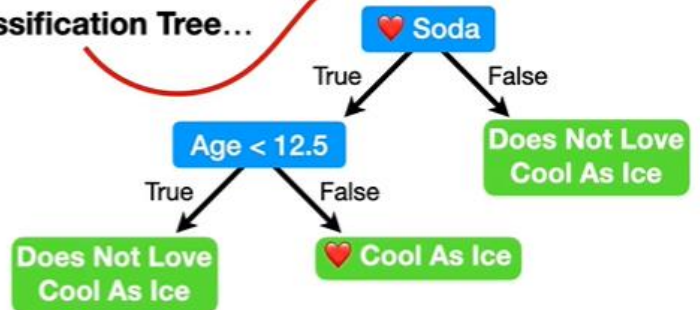
If overfitting occurs in naïve Bayes use feature selection, Laplace smoothing, etc.

❖ Decision Tree:

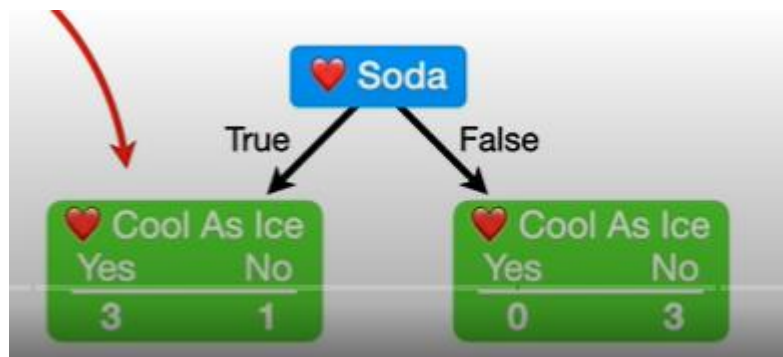
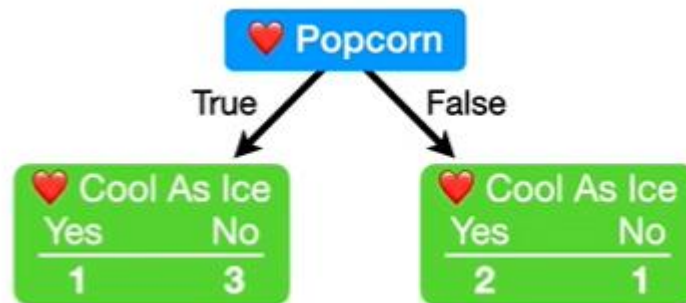
Imagine you have a dataset with different types of fruits, and you want to classify them as either apples, oranges, or bananas. A decision tree algorithm helps make these classifications based on features like color, size, and texture. The goal of decision trees, is to partition the data into subsets based on the input features, leading to decisions or predictions. There is a root node that is the first node of a decision tree also you can say it's a node that does not have a parent. Then there is a leaf node which is the node who do not have child nodes. The other are decision nodes that are in between these 2 nodes. Below is the data and its corresponding tree.

| Loves Popcorn | Loves Soda | Age | Loves Cool As Ice |
|---------------|------------|-----|-------------------|
| Yes | Yes | 7 | No |
| Yes | No | 12 | No |
| No | Yes | 18 | Yes |
| No | Yes | 35 | Yes |
| Yes | Yes | 38 | Yes |
| Yes | No | 50 | No |
| No | No | 83 | No |

Classification Tree...



Now we must select the root node among the input features which one will be a root node. So first we will keep all inputs as root and will do calculation like if the root node is popcorn then if someone likes popcorn then will it loves cool as ice or not or if someone who does not love popcorn will someone love cool as ice or not.



You have seen that 2 leaves of popcorn and one leaf of soda contain a mixture of people who love as cool as ice, and some do not it is called

impure. While the right leaf of soda does not contain the mixture it is called pure. There are several ways to quantify the impurity known as entropy, information gain, and Gini impurity. We will calculate the Gini impurity of nodes. We calculate it as below. First we calculate the Gini impurity of leaves. For the left leaf:

Gini Impurity for a Leaf = $1 - (\text{the probability of "Yes"})^2 - (\text{the probability of "No"})^2$

$$= 1 - \left(\frac{1}{1+3}\right)^2 - \left(\frac{3}{1+3}\right)^2$$

$$= 0.375$$

And when we do the math, we get **0.375**.

For right leaf:

Gini Impurity for a Leaf = $1 - (\text{the probability of "Yes"})^2 - (\text{the probability of "No"})^2$

$$= 1 - \left(\frac{2}{2+1}\right)^2 - \left(\frac{1}{2+1}\right)^2$$

$$= 0.444$$

And when we do the math we get **0.444**.

Now the total Gini impurity of a node is calculated as,

Total Gini Impurity = weighted average of **Gini Impurities** for the **Leaves**

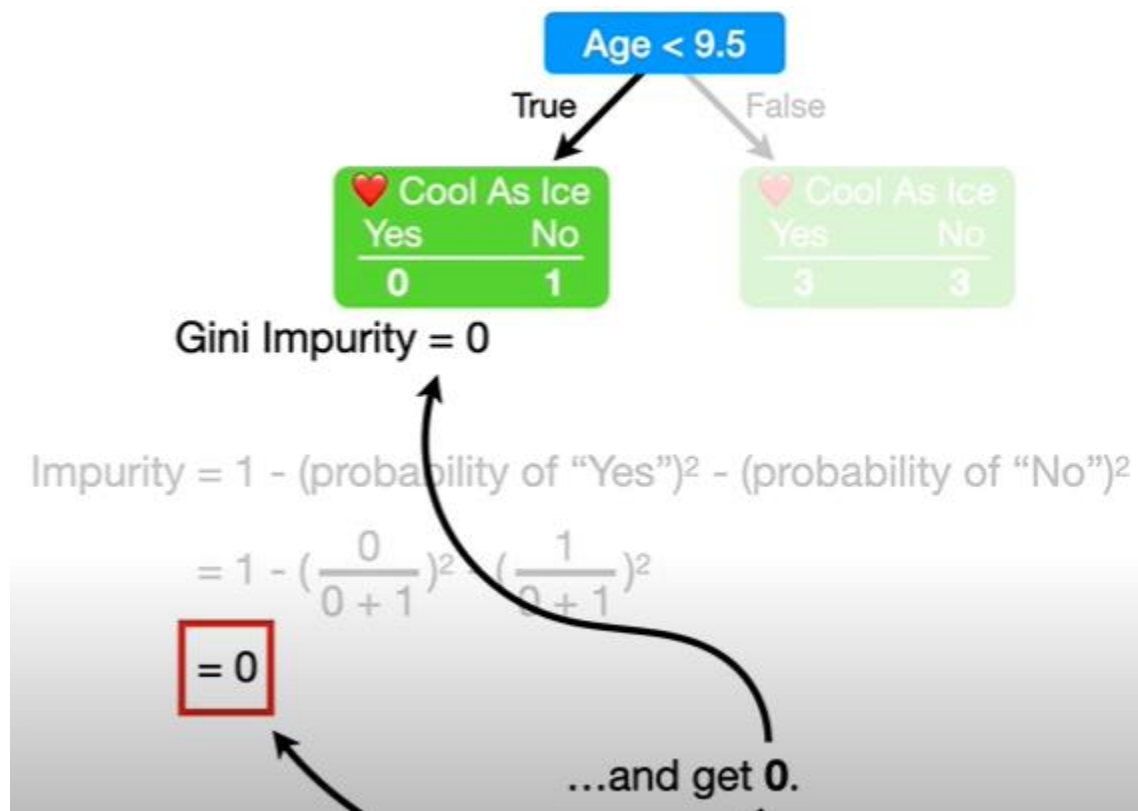
$$= \left(\frac{4}{4+3}\right) 0.375 + \left(\frac{3}{4+3}\right) 0.444$$

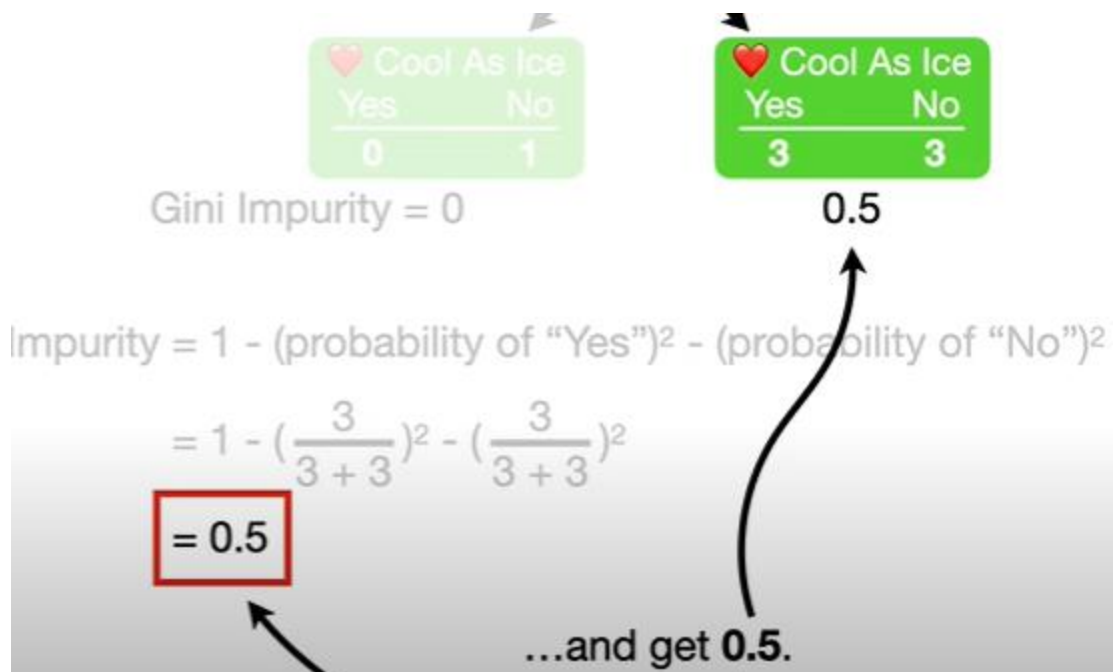
$$= 0.405$$

We do the same calculations for soda. For input feature age as it contains the continuous value, we first sort it in ascending order and calculate the average of 2 adjacent age values and then calculate the Gini impurity value of each average age.

| | | Ice | |
|------|----|-----|-----------------------|
| | 7 | No | |
| 9.5 | | | Gini Impurity = 0.429 |
| | 12 | No | |
| 15 | | | Gini Impurity = 0.343 |
| | 18 | Yes | |
| 26.5 | | | Gini Impurity = 0.476 |
| | 35 | Yes | |
| 36.5 | | | Gini Impurity = 0.476 |
| | 38 | Yes | |
| 44 | | | Gini Impurity = 0.343 |
| | 50 | No | |
| 66.5 | | | Gini Impurity = 0.429 |
| | 83 | No | |

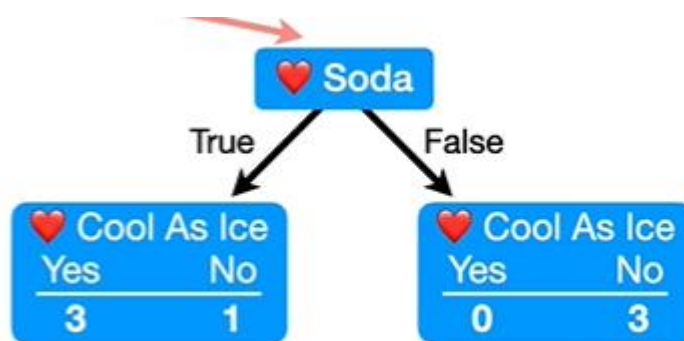
Now here is an example of how we calculate the Gini impurity value of each average values.





$$\text{Total Gini Impurity} = (\frac{1}{1+6}) 0 + (\frac{6}{1+6}) 0.5 = \mathbf{0.429}$$

In the same way, we calculate the Gini impurity for all average values and select the minimum one. Now we see among all the gini impurity value of soda was minimal all so we selected soda as the root node.



Now we see the left node is n=impure so will split it in order to reduce the impurity. We will follow the same steps as explained earlier and will select the node with the minimum value. As by calculations we observed age less than 12.5 has less gini impurity so we will select it as a node. so in the decision tree which depth tree would be good with a depth of 5 or 7, we built different trees to check their accuracy and selected the one with the

best accuracy and best performed on the training set. Decision trees are easy to interpret and visualize. Can handle both categorical and numerical data. Can capture complex relationships in data. But they may be sensitive to small variations in data and in order to avoid overfitting in the decision tree we do pruning.

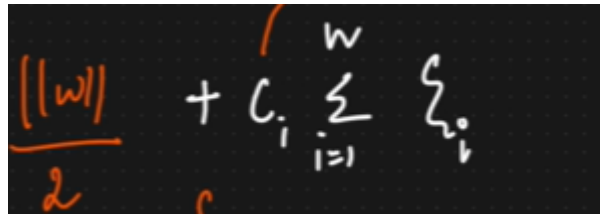
❖ **Random Forest Classifier:**

Random Forest is an ensemble learning method used for both classification and regression tasks in machine learning. It builds multiple decision trees during training and merges their outputs to get a more accurate and stable prediction. Each tree in a Random Forest is trained on a different random sample of the training data, allowing it to learn different aspects of the data. Randomly select a subset of data (with replacement) for training each tree. Some data points may be included multiple times, while others may not be included at all called out of the bag data set. So using this data set we can check the accuracy of random forest. The portion of out of bag data set that is incorrectly classified is called out of bag error. This is usually known as bootstrapping. The data set is called a bootstrap data set. Then we create the decision tree for each data set using above explained information. Random forest can handle large data and its accuracy is better than decision tree. But it requires more computational resources. In order to avoid overfitting do pruning (limit the depth of a tree), etc.

❖ **Support Vector Machine:**

Support Vector Machines, often abbreviated as SVM, are a powerful class of supervised machine learning algorithms used for classification and regression tasks. They are particularly effective when dealing with complex datasets where there is no clear linear separation between different classes or groups. The primary goal of SVM is to find the best possible boundary (or hyperplane) that can effectively separate different classes in the dataset. This boundary is known as the "decision boundary" or "hyperplane." SVM aims to maximize the margin between the decision boundary and the nearest data points from each class. These nearest data points are called

"support vectors." Maximizing the margin not only ensures a good separation but also enhances the model's generalization to unseen data. In cases where the data isn't linearly separable in its original feature space, SVM uses a "kernel trick" to map the data into a higher-dimensional space where linear separation becomes possible. Common kernel functions include polynomial kernels and radial basis function (RBF) kernels. Training an SVM involves minimizing a cost function. This function tries to find the optimal hyperplane that maximizes the margin while minimizing classification errors. To classify a new data point, you apply the learned decision boundary. You check which side of the hyperplane it falls on. Like in an equation of decision boundary, the values are entered if its positive it's classified into one class otherwise, if negative then classified to another class. The cost function is

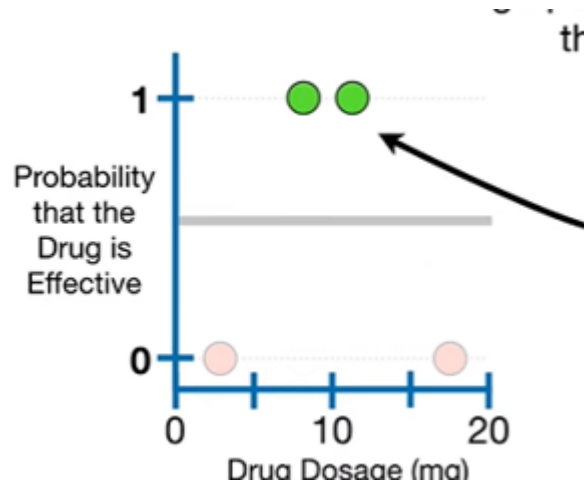


$$\frac{||w||}{2} + C \sum_{i=1}^n \xi_i$$

Where c is a hyperparameter that tells how many data points can be misclassified. While the other parameter tells the distance between misclassified data points and marginal planes.

❖ XGBoost Classifier:

XGBoost (eXtreme Gradient Boosting) is a powerful ensemble learning algorithm known for its efficiency and high performance across a wide range of tasks, including classification. XGBoost is an ensemble learning method that combines the predictions of multiple weak learners (usually decision trees) to create a stronger, more accurate model. It does this by iteratively building trees and then combining their predictions. We basically use the initial prediction for building an initial tree that is 0.5 means 50% chance of happening. The data set is below,



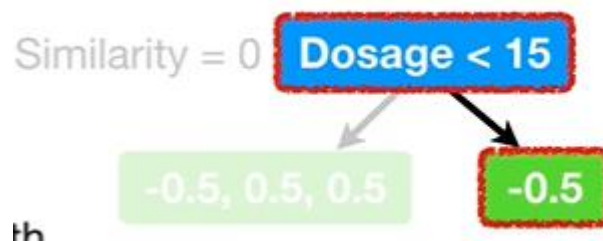
Now we calculate the residual values by subtracting the training data set and the initial predicted value.

-0.5, 0.5, 0.5, -0.5

The formula for the similarity score is,

$$\frac{(\sum \text{Residual}_i)^2}{\sum [\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)] + \lambda}$$

Now the similarity score for root is 0 which is of residual values. Now for building a tree, we first have to choose the root node that has a high value of gain. For this, we took the last 2 data points and calculated its average which is 15 which is the threshold value.



$$\frac{(-0.5 + 0.5 + 0.5)^2}{(0.5 \times (1-0.5)) + (0.5 \times (1-0.5)) + (0.5 \times (1-0.5)) + 0}$$

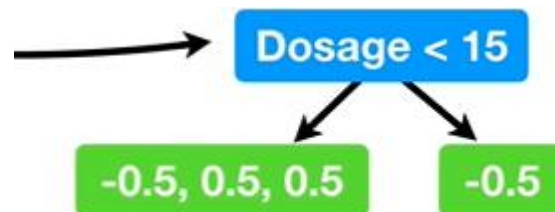
So similarity score of a left node is 0.33.

$$\frac{(-0.5)^2}{0.5 \times (1 - 0.5) + \lambda} = 1, \text{ when } \lambda = 0.$$

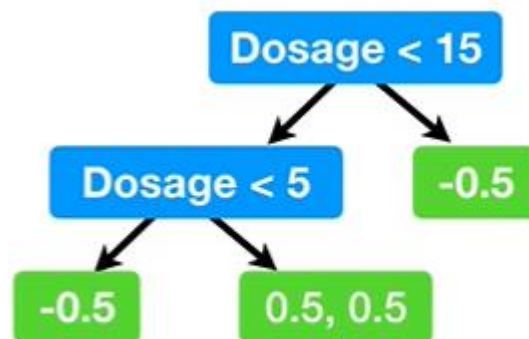
The similarity score of the right node. Now calculate the gain the formula is

$$\text{Gain} = \text{Left}_{\text{Similarity}} + \text{Right}_{\text{Similarity}} - \text{Root}_{\text{Similarity}} \quad \text{That is } 1.33.$$

Now check the same gain value for other threshold values and calculate the gain. And by calculation dosage less than 15 has more gain so it will be root node.



Now as there is only one residual in right node so we will not split it but will split the left node. It has 3 residual values so select the threshold values and check which threshold gives the high value of gain then select it. So dosage less than 5 will be selected.



The minimum number of residuals in each leaf is determined by calculating the cover. When XGBoost is used for classification the cover is equal to

$$\frac{(\sum \text{Residual}_i)^2}{\sum [\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)] + \lambda}$$

For regression the cover equals to

$$\text{Similarity Score} = \frac{\text{Sum of Residuals Squared}}{\text{Number of Residuals} + \lambda}$$

In order to avoid overfitting we do pruning we prune by calculating the difference between gain and gamma. If the answer is positive we do not remove the node but if the difference is negative we prune the node. Now for classification the output value for leaves are :

$$\frac{(\sum \text{Residual}_i)}{\sum [\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)] + \lambda}$$

-0.5

0.5, 0.5

For now, we'll let $\lambda = 0$,
because this is the default
value...

$$\frac{-0.5}{0.5 \times (1 - 0.5) + 0} = -2$$

is 2.

Output = -2

$$\frac{0.5 + 0.5}{0.5 \times (1 - 0.5) + 0.5 \times (1 - 0.5) + 0} = 2$$

So for making a prediction, we use the initial prediction value. We need to convert the probability to log(odds) value.

$$\log\left(\frac{p}{1-p}\right) = \log(\text{odds})$$

$$\log\left(\frac{0.5}{1-0.5}\right) = \log(\text{odds})$$

The value of log(odds) is 0. So the predictions are made using,

...and that gives us a **log(odds)** value = -0.6.

Output = -2

Output = 2

Output = -2

$$\text{log(odds) Prediction} = 0 + (0.3 \times -2) = -0.6$$

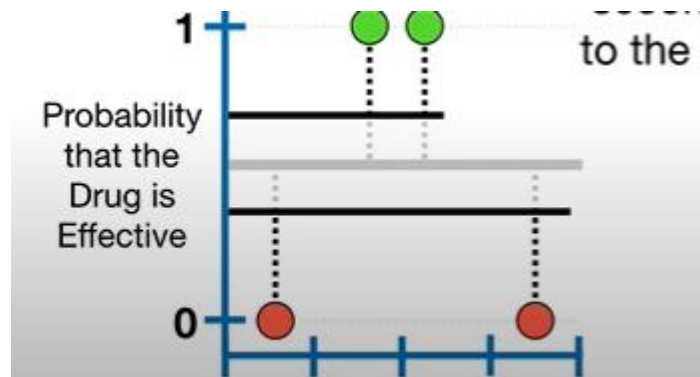
0.3 is

learning rate.

To convert the log(odds) value into probability we plug it into the logistic function.

$$\text{Probability} = \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}$$

$$\text{Probability} = \frac{e^{-0.6}}{1 + e^{-0.6}} = 0.35$$



The residual value gets shorter now. Now we built the second tree using new residuals and so on. For the second tree the residuals are

-0.35, 0.35, 0.35, -0.35

The similarity score will look like,

$$\text{Similarity Score} = \frac{(\sum \text{Residual}_i)^2}{(0.35 \times (1-0.35)) + (0.65 \times (1-0.65)) + (0.65 \times (1-0.65)) + (0.35 \times (1-0.35)) + \lambda}$$

It is known for its high accuracy.

Un-Supervised Learning:

Unsupervised learning is a type of machine learning where the algorithm learns from unlabeled data. Unlike supervised learning, there are no target labels provided. Instead, the algorithm identifies patterns, relationships, and structures in the data on its own.

Some examples of unsupervised learning are anomaly detection, pattern recognition, audience segmentation, etc.

Types of unsupervised learning tasks:

1. Clustering
2. association

Clustering:

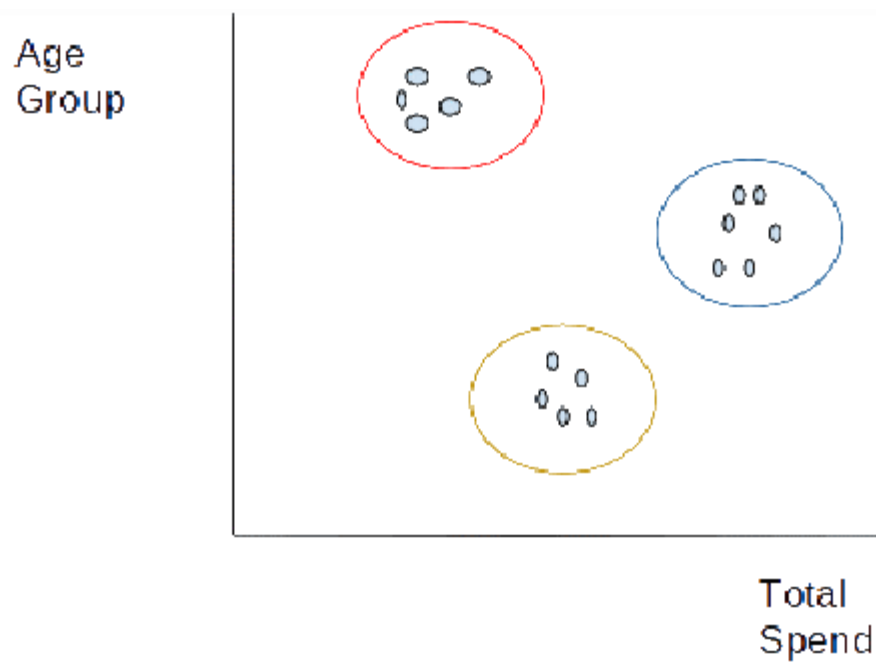
Unsupervised clustering is a type of machine learning where the algorithm tries to find natural groupings or clusters within a dataset without being provided with any specific labels or target information.

Examples of clustering include image segmentation, anomaly detection, etc.

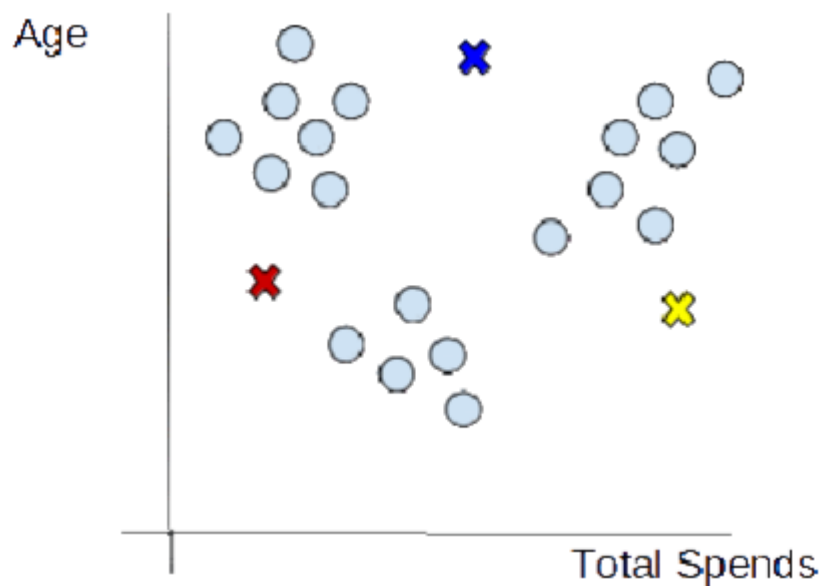
Algorithms:

❖ K-means:

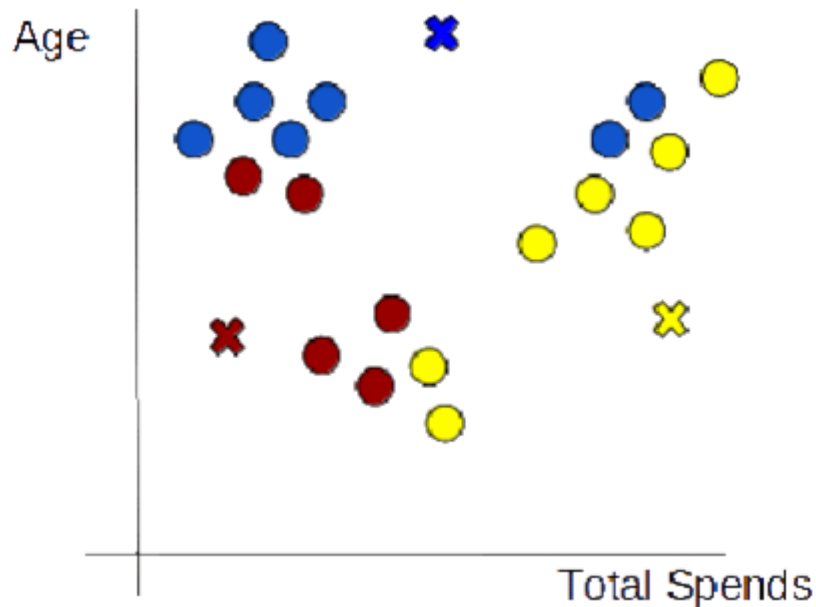
K-means is a centroid-based clustering algorithm, where we calculate the distance between each data point and a centroid to assign it to a cluster. The goal is to identify the K number of groups in the dataset. It is an iterative process of assigning each data point to the groups and slowly data points get clustered based on similar features. Here, we divide a data space into K clusters and assign a mean value to each. The data points are placed in the clusters closest to the mean value of that cluster. Let consider the data we have given below,



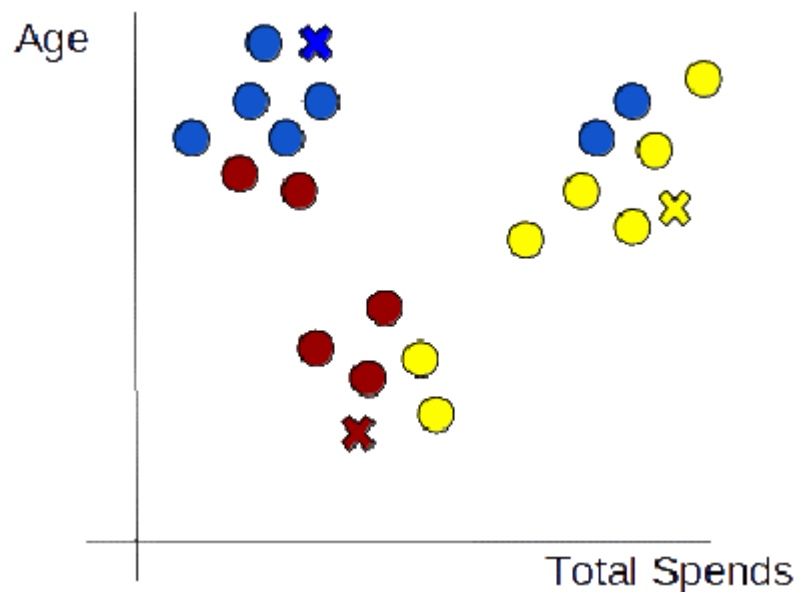
The first step is to define the K number of clusters in which we will group the data. Let's select $K=3$. The centroid is the center of a cluster but initially, the exact center of data points will be unknown so, we select random data points and define them as centroids for each cluster. We will initialize 3 centroids in the dataset.



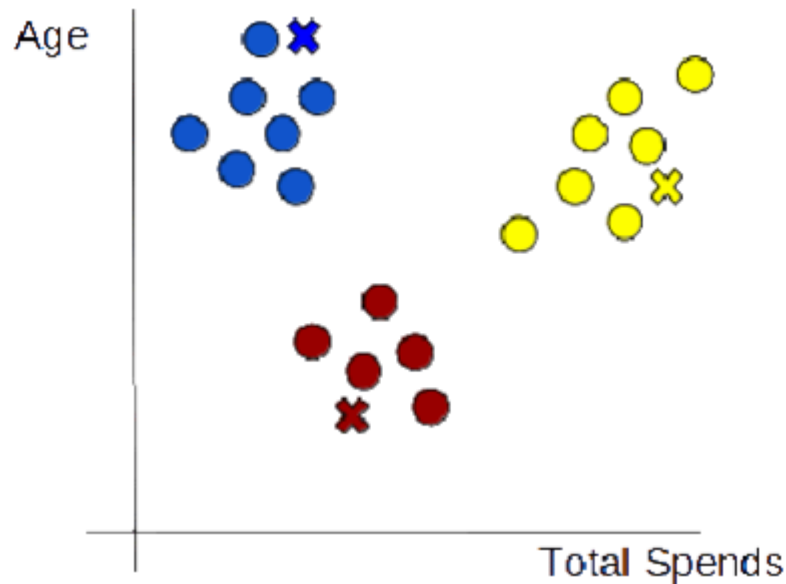
Now that centroids are initialized, the next step is to assign data points X_n to their closest cluster centroid C_k this is done by calculating the Euclidean distance between the data set and centroid and assigning a data point to a cluster where the distance between data point and a particular cluster's centroid is less.



Next, we will re-initialize the centroids by calculating the average of all data points of that cluster.

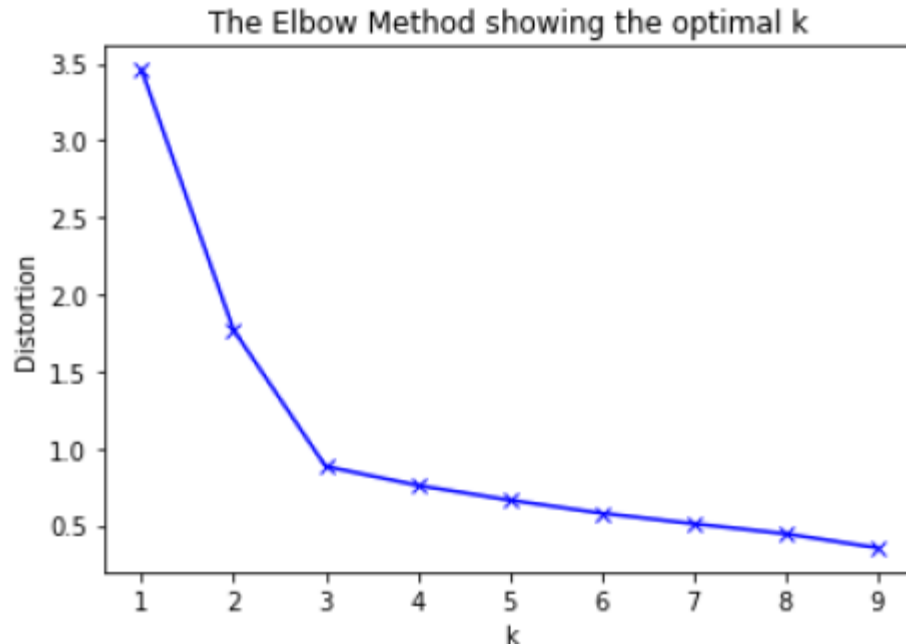


We will keep repeating steps 3 and 4 (finding the distance between data points and centroids and after that reinitializing the centroids) until we have optimal centroids and the assignments of data points to correct clusters are not changing anymore.



Here we clustered the data set. Now for initializing the value of k , there are many ways we can also select the k centroids randomly or we can use the most appropriate method k -means++. K -means++ is a smart centroid initialization method for the K -mean algorithm. There are simple steps first randomly pick the first centroid then calculate the distance between all data points and the selected centroid, and select the one with the farthest distance from the first centroid. Repeat the steps until you get the k centroids.

Now discuss how to know the value of k for that we use the famous elbow method. We select a range of k values like 1,2,3,4 and so on. Then find the distance between data points and the centroid and find the squared sum distance. Create a plot where the x-axis represents the number of clusters (K) and the y-axis represents the corresponding SSD values. Repeat the steps for other values of k and then select the elbow point.



The distortion (or inertia) in k-means clustering measures how spread out the data points are within each cluster. It's calculated as the sum of the squared distances between each data point and its corresponding cluster centroid. As you increase the number of clusters (k), the distortion tends to decrease. This is because as you add more clusters, each data point tends to be closer to its nearest centroid. So, initially, adding more clusters leads to a significant reduction in distortion. However, after a certain point, adding more clusters doesn't result in a significant reduction in distortion. The "elbow point" is where the distortion starts to flatten out. It's the point where the rate of decrease sharply changes, forming an "elbow" shape in the distortion vs. k plot.

K-Means is easy to understand and implement, making it a quick and efficient clustering method. It can handle large datasets efficiently, making it suitable for big data applications. K-Means is computationally faster compared to other clustering algorithms, making it suitable for real-time applications. But, You need to specify the number of clusters in advance, which can be challenging in some cases. Outliers can significantly impact the clustering results, potentially leading to inaccurate cluster assignments. Here are some steps you can take to address overfitting in K-Means:

Reduce the number of clusters(k), address outliers, etc.

❖ Hierarchal Clustering:

Hierarchical clustering is a method used to group together similar data points based on their features. It creates a tree-like diagram called a dendrogram, which shows the arrangement of clusters.

Hierarchal clustering is of two types:

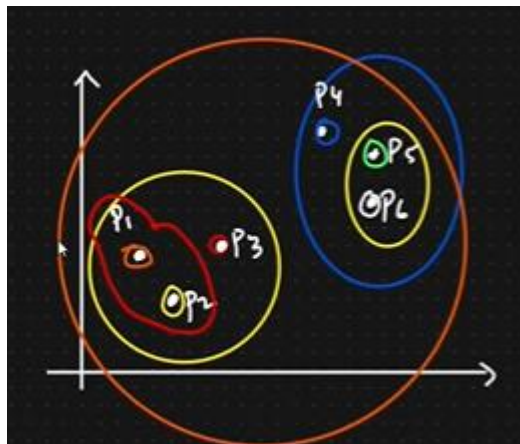
- Agglomerative clustering
- Divisive clustering

In Agglomerative Hierarchical Clustering each data or observation is treated as its cluster. A pair of clusters are combined until all clusters are merged into one big cluster that contains all the data.

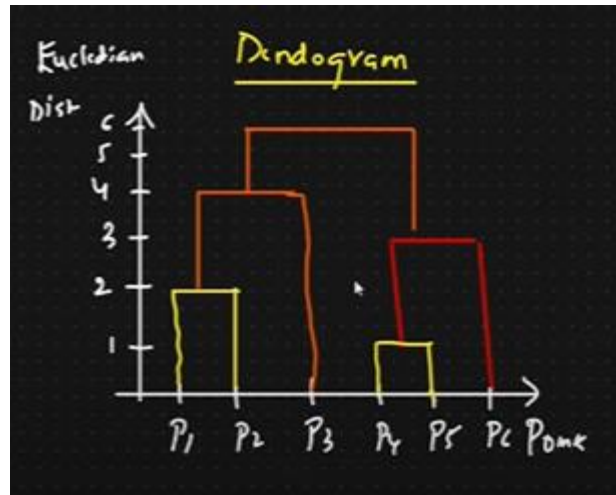
In Divisive Hierarchical Clustering, entire data or observation is assigned to a single cluster. The cluster is further split until there is one cluster for each data or observation.

So, both are the reverse of each other here I will explain agglomerative clustering.

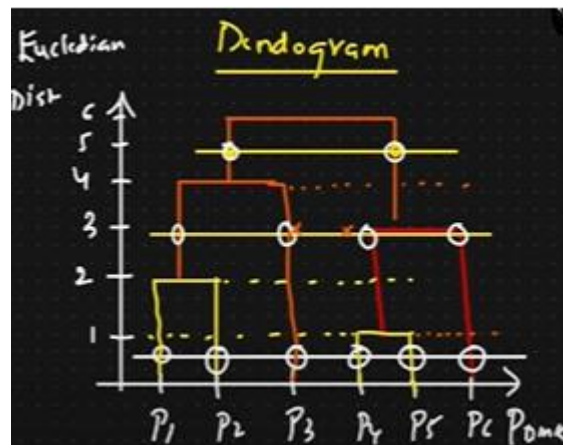
So, in agglomerative clustering for each point initially, we will consider it as a separate cluster. Next, we will find the nearest point and create a new cluster. So, we keep on repeating the steps until we get a single cluster.



Now to check how many clusters are there we use a dendrogram.



Like the first clusters of P_1 and P_2 are formed the P_4 and P_5 and we keep continuing steps to check for the nearest points and form clusters now the thing is what will be the value of k ? For we take a threshold value and for a threshold, we take the longest vertical line where no horizontal line passes through it. Like threshold may be of distance 5, 3 or 1.



So the threshold found at a distance equals 5 and will draw a horizontal line and will check how many points line has cut in the above case 2 points are formed by cutting the vertical line so value of k will be 2. Here is how we choose threshold value.

So, divisive clustering starts with all data points in a single cluster and then recursively divides them into smaller clusters. In divisive clustering, we start with all data points belonging to one large cluster. We look for the cluster that is the least cohesive (i.e., it contains data points that are less similar to each other). This cluster is then split into two smaller clusters. This process

of identifying the least cohesive cluster and splitting it continues recursively until each data point is in its own cluster. Similar to agglomerative clustering, you can create a dendrogram to visualize the process. To decide on the number of clusters, you can use a similar approach as in agglomerative clustering. Look for the threshold where you get the desired number of clusters.

The algorithm is used well if the data set is small if we have a large data set then k means will be good. Hierarchical clustering is less sensitive to outliers compared to K-means. Hierarchical clustering can be more computationally demanding, especially for large datasets

Anomaly Detection:

Anomaly detection, also known as outlier detection, is the process of identifying data points or patterns that deviate significantly from the norm in a dataset. These deviations are often indicative of unusual or unexpected behavior, which may warrant further investigation. Anomaly detection is widely used across various domains, including cybersecurity, fraud detection, healthcare, manufacturing, finance, and more. It helps identify rare events or irregularities that may have significant implications. In a Gaussian (Normal) Distribution, the probability density is highest at the mean and decreases symmetrically as you move away from the mean in both directions. This is what creates the characteristic bell-shaped curve.

Algorithm:

- Choose n features x_i that you think might be indicative of anomalous examples (Create a histogram for the feature. This shows the distribution of values. In a Gaussian Distribution, you'll see a bell-shaped curve. Does it resemble a bell curve? If so, it's an indicator that the data might follow a Gaussian Distribution. If the curve is heavily skewed, the data might not follow a Gaussian Distribution. So, you can apply mathematical transformations to the data to make it more Gaussian-like.)



- Find the mean and variance of each input feature.

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)} \quad \sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

- Given new example x compute $p(x)$,

$$p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

- Anomaly if $p(x) < \epsilon$ (a threshold value)

Now anomaly detection can be used over supervised learning in many cases like, When you have limited or no labeled data for anomalies. When you don't have a clear understanding of all possible types of anomalies. Anomaly detection can discover unknown anomalies without prior knowledge.

Recommendation Systems:

Recommendation systems are intelligent algorithms that assist users in discovering relevant and personalized content, products, or services. They play a vital role in enhancing user experiences across various platforms, from e-commerce websites to streaming services. Recommendation systems have revolutionized the way users interact with platforms and services. They not only improve user engagement but also drive sales, increase customer satisfaction, and foster loyalty.

Example: Amazon items recommendation, Netflix movie recommendation.

There are 2 common techniques used in recommendation systems explained below.

Techniques:

❖ **Collaborative Filtering:**

“Linear regression is used to predict the ratings that a user might give to a movie they haven't seen yet. The idea is to find a linear relationship between the features (such as user behavior, movie characteristics, etc.) and the ratings. This allows us to estimate what rating a user might give to a movie based on their historical preferences and characteristics of the movie.”

“Logistic regression, on the other hand, is used in a different part of the recommendation system. It can be used to answer binary questions like Did user j watch the movie i after being shown? This involves a yes/no prediction, which is a classification problem. Logistic regression is well-suited for this because it models the probability of a binary outcome. In a recommendation system, this could be used to predict whether a user will watch a recommended movie or not based on their behavior and preferences.”

For working on movie recommendations, the model is the same as the linear regression:

For user j : Predict user j 's rating for movie i as $w^{(j)} \cdot x^{(i)} + b^{(j)}$

We have movie and user matrix data and entries are rated and there are many missing values so our goal is to build a model that predicts it well.

| | User 1 | User 2 | User 3 | ... | User n |
|---------|--------|--------|--------|-----|--------|
| Movie 1 | Rating | Rating | Rating | | Rating |
| Movie 2 | Rating | Rating | Rating | | Rating |
| Movie 3 | Rating | Rating | Rating | | Rating |
| ... | ... | ... | ... | | ... |
| Movie m | Rating | Rating | Rating | | Rating |

so, for user and item parameters are trained and cost function is reduced using gradient descent, and the correct value of parameters is obtained. First of all, we will calculate the parameters of movies and after that, we will calculate for parameters of users to predict the ratings for missing values for choosing the number of parameters of movies we use cross-validation to use different values and then select which gives high accuracy. The cost to learn parameters for user j is,

$$J(w^{(j)}, b^{(j)}) = \frac{1}{2} \sum_{i:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^n (w_k^{(j)})^2$$

To learn parameters for all users the overall cost function is below,

$$\frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2$$

$n_u \Rightarrow$ number of users

Now to learn the parameters for the movie the cost function is,

$$\frac{1}{2} \sum_{j:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^n (x_k^{(i)})^2$$

To learn the parameters for all movies the overall cost function is,

$$J(x^{(1)}, x^{(2)}, \dots, x^{(n_m)}) = \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

$n_m \Rightarrow$ total number of movies.

Now, the overall cost function to learn the parameters for users and movies is given below,

$$J(w, b, x) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

To reduce the cost function and get the best parameters we may use gradient descent.

$$\begin{aligned} w_i^{(j)} &= w_i^{(j)} - \alpha \frac{\partial}{\partial w_i^{(j)}} J(w, b, x) \\ b^{(j)} &= b^{(j)} - \alpha \frac{\partial}{\partial b^{(j)}} J(w, b, x) \\ x_k^{(i)} &= x_k^{(i)} - \alpha \frac{\partial}{\partial x_k^{(i)}} J(w, b, x) \end{aligned}$$

Now using the model we fill in the missing values so we recommend the movies user j has liked the sci-fiction movies so we recommend other sci-fiction movies highly rated by other people to user j predicting that user j

may like this. In collaborative filtering, you don't have to explicitly choose whether to find the best parameters for users or movies first. The process involves updating both sets of parameters (users and movies) in an iterative manner.

Here's a simplified step-by-step process:

Start with initial parameter values for both users and movies.

Update movie parameters while keeping user parameters fixed.

Then, update user parameters while keeping movie parameters fixed.

Repeat steps 2 and 3 in an iterative manner.

Continue this process until the model converges (meaning the cost function stabilizes or changes very slowly).

This iterative process allows the model to gradually learn the best parameters for both users and movies simultaneously. So, there's no specific order to follow; you update both sets of parameters in a back-and-forth manner until the model learns the relationships between users and movies.

Now for binary classification cases used in recommendation system, the equations are below,

For binary labels:

**Predict that the probability of $y^{(i,j)} = 1$
is given by $g(w^{(j)} \cdot x^{(i)} + b^{(j)})$**

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

Loss for binary labels $y^{(i,j)}$: $f_{(w,b,x)}(x) = g(w^{(j)} \cdot x^{(i)} + b^{(j)})$

$$L(f_{(w,b,x)}(x), y^{(i,j)}) = -y^{(i,j)} \log(f_{(w,b,x)}(x)) - (1 - y^{(i,j)}) \log(1 - f_{(w,b,x)}(x))$$

Loss for
single
example

$$J(w, b, x) = \sum_{(i,j): r(i,j)=1} L(f_{(w,b,x)}(x), y^{(i,j)})$$

❖ Content-based Filtering:

Collaborative filtering:

Recommend items to you based on ratings of users who gave similar ratings as you

Content-based filtering:

Recommend items to you based on features of user and item to find good match

Content-based filtering is another important technique used in recommendation systems. Unlike collaborative filtering, which relies on user-item interactions and similarities, content-based filtering focuses on the attributes or features of items and users.

For example if we are working on a movie recommendation system,

User features:

- Age
- Gender
- Country
- Movies watched
- Average rating per genre
- ...

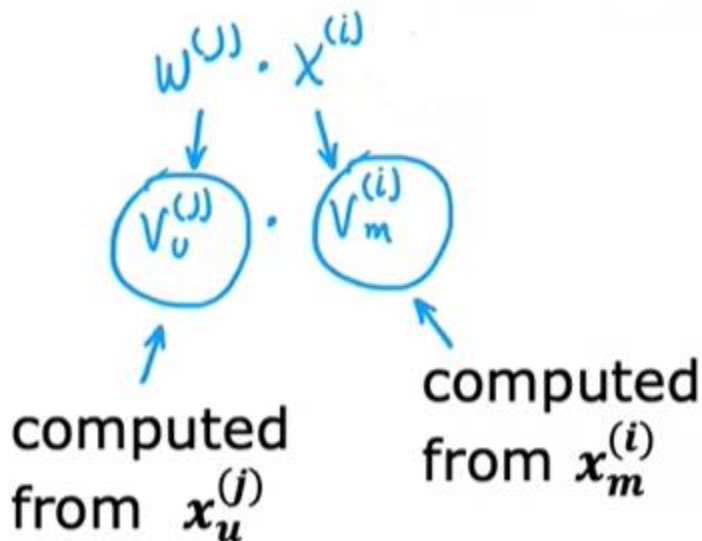
$\mathbf{x}_u^{(j)}$ for user j

Movie features:

- Year
- Genre/Genres
- Reviews
- Average rating
- ...

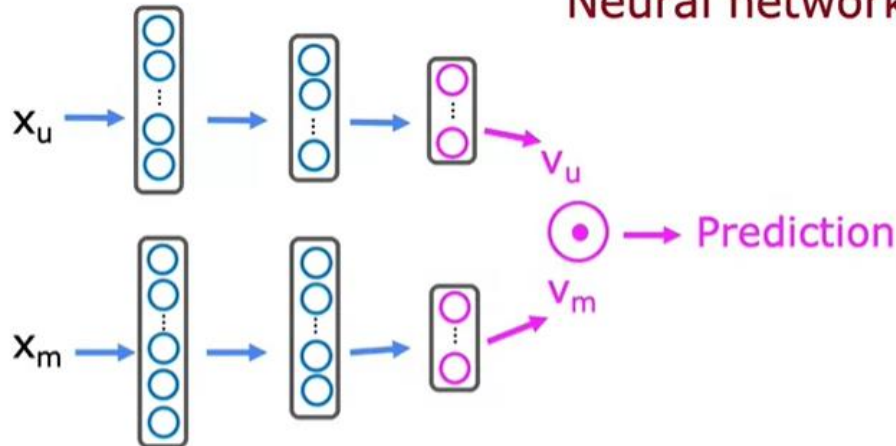
$\mathbf{x}_m^{(i)}$ for movie i

Predict rating of user j on movie i as



For finding these vectors we will use neural network.

Neural network architecture



The cost function will be,

$$\text{Cost function } J = \sum_{(i,j):r(i,j)=1} (v_u^{(j)} \cdot v_m^{(i)} - y^{(i,j)})^2 + \text{NN regularization term}$$

To find movies similar to movie i:

$$\|v_m^{(k)} - v_m^{(i)}\|^2 \text{ small}$$

If the dot product between a user's preference vector and the feature vector of a sci-fi movie is large, it indicates a high level of alignment or similarity. This suggests that the user's preferences are well-matched with the features associated with sci-fi movies.

If the distance above is small then it means movies are similar if we find the distance between the vectors of Interstellar and The Martian the distance will be small. So if a user likes science fiction movies such movies will be recommended to users which are more similar to science fiction.

Summary:

Both collaborative filtering and content-based filtering have their strengths and weaknesses. The choice between them depends on the specific context and requirements of the recommendation system.

Collaborative filtering doesn't require explicit feature extraction. It learns preferences directly from user-item interactions, making it suitable for a wide range of domains. It requires a sufficient amount of user-item interaction data to make accurate recommendations, which can be an issue in sparse datasets. Content-based filtering offers clear interpretability, as recommendations are based on explicit item features (e.g., genre, director). It requires explicit feature extraction and engineering, which can be time-consuming and may require domain knowledge.

If you have a large dataset with significant user-item interactions, collaborative filtering can be very effective. When you have detailed and explicit features for items that can be used for recommendation (e.g., genre, tags, attributes) content-based filtering will be best.

Combining collaborative and content-based filtering can often yield the best of both worlds. Hybrid models can mitigate the limitations of individual methods.

Dimensionality Reduction:

Techniques:

❖ PCA (Principal Component Analysis):

“Process of figuring out the most important features that has the most impact on the target variable.”

Principal Component Analysis (PCA) is a technique used in data analysis and dimensionality reduction. It helps find the underlying patterns in complex data by transforming it into a new coordinate system where the data's variability is maximized along the new axes, called principal components. This makes it easier to visualize and analyze the data while retaining as much relevant information as possible.

When you have a high-dimensional dataset with many features (like 10), visualizing it directly becomes challenging because our visual perception is

limited to three dimensions at most. PCA helps by transforming the data into a lower-dimensional space (like 2D or 3D) while retaining as much of the important information as possible. This makes it much easier to visualize and understand the relationships between data points.

| Country | GDP (trillions of US\$) | Per capita GDP (thousands of intl. \$) | Human Development Index | Life expectancy | what if 50 features |
|-----------|-------------------------|--|-------------------------|-----------------|---------------------|
| Canada | 1.577 | 39.17 | 0.908 | 80.7 | |
| China | 5.878 | 7.54 | 0.687 | 73 | |
| India | 1.632 | 3.41 | 0.547 | 64.7 | |
| Russia | 1.48 | 19.84 | 0.755 | 65.5 | |
| Singapore | 0.223 | 56.69 | 0.866 | 80 | |
| USA | 14.527 | 46.86 | 0.91 | 78.3 | |
| ... | ... | ... | ... | ... | |

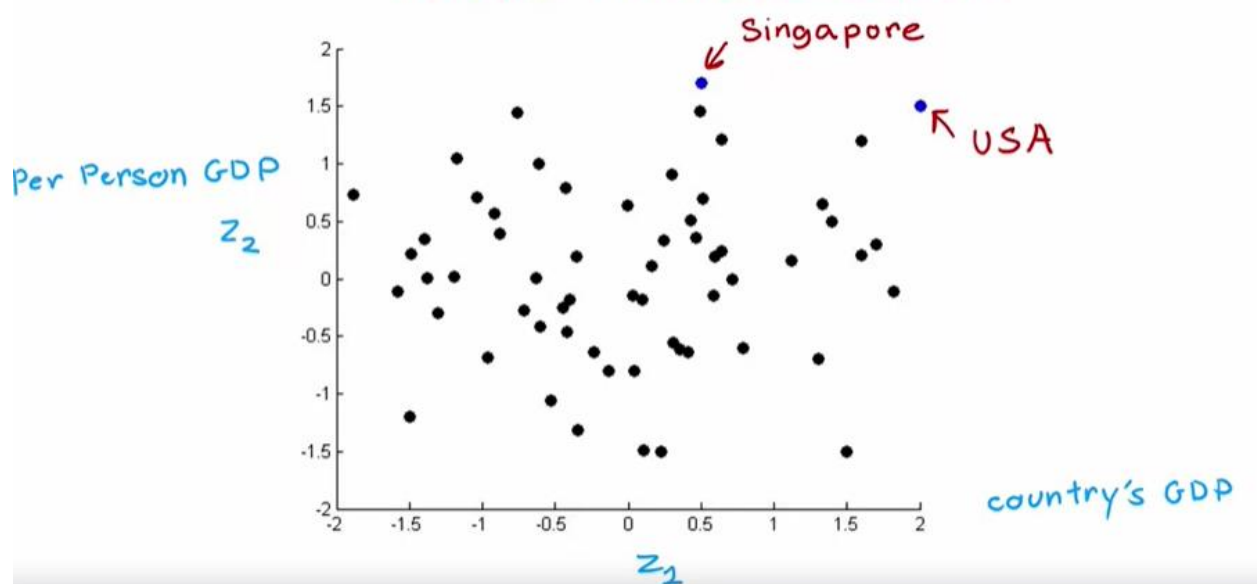
Before applying PCA first you should do feature scaling.

50 features

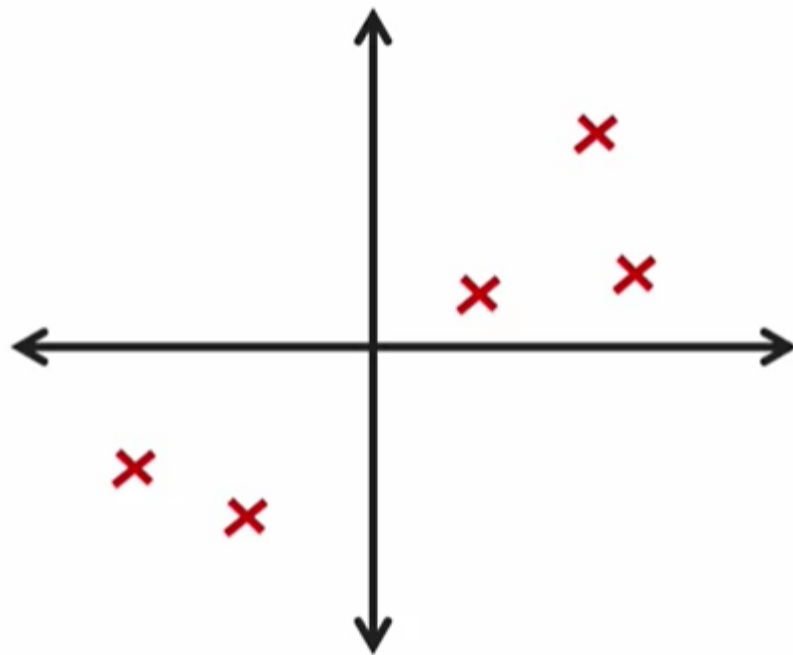
| Country | x_1 GDP (trillions of US\$) | x_2 Per capita GDP (thousands of intl. \$) | x_3 Human Development Index | x_4 Life expectancy | ... | Poverty Index (Gini as percentage) | Mean household income (thousands of US\$) | ... |
|---------|----------------------------------|---|----------------------------------|--------------------------|-----|------------------------------------|---|-----|
| Canada | 1.577 | 39.17 | 0.908 | 80.7 | | 32.6 | 67.293 | ... |
| China | 5.878 | 7.54 | 0.687 | 73 | | 46.9 | 10.22 | ... |
| India | 1.632 | 3.41 | 0.547 | 64.7 | | 36.8 | 0.735 | ... |
| Russia | 1.48 | 19.84 | 0.755 | 65.5 | | 39.9 | 0.72 | ... |

| Country | z_1 | z_2 |
|---------|-------|-------|
| Canada | 1.6 | 1.2 |
| China | 1.7 | 0.3 |
| India | 1.6 | 0.2 |
| Russia | 1.4 | 0.5 |

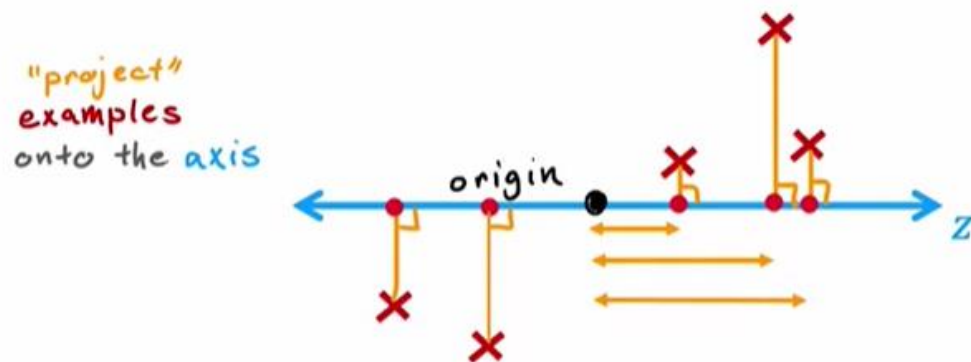
Data visualization



Consider the data ,

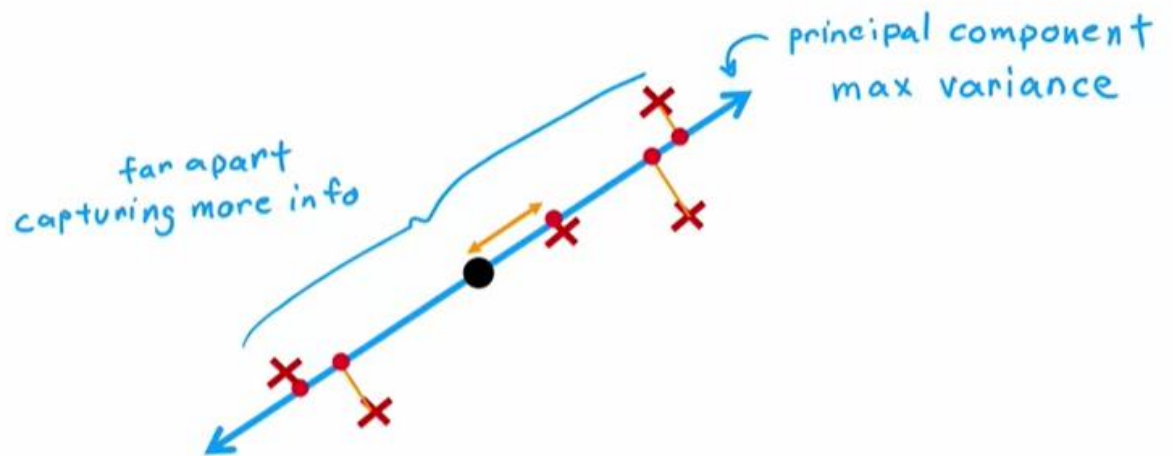


Choose an axis



Best choice,

Choose an axis

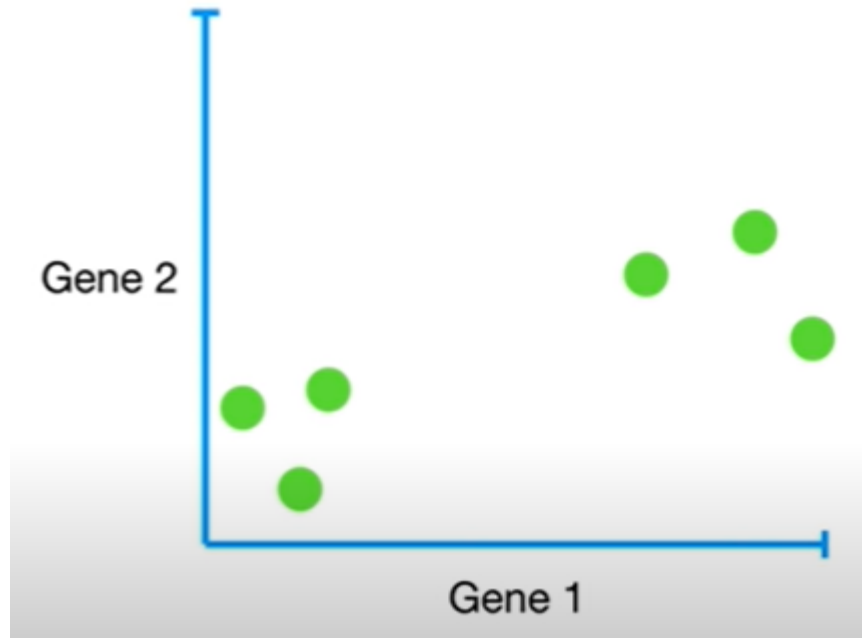


Now let's understand the PCA.

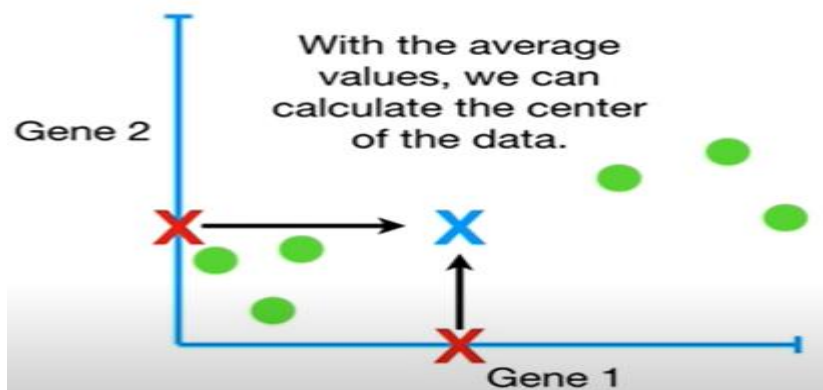
The data set is below,

| | Mouse 1 | Mouse 2 | Mouse 3 | Mouse 4 | Mouse 5 | Mouse 6 |
|--------|---------|---------|---------|---------|---------|---------|
| Gene 1 | 10 | 11 | 8 | 3 | 2 | 1 |
| Gene 2 | 6 | 4 | 5 | 3 | 2.8 | 1 |

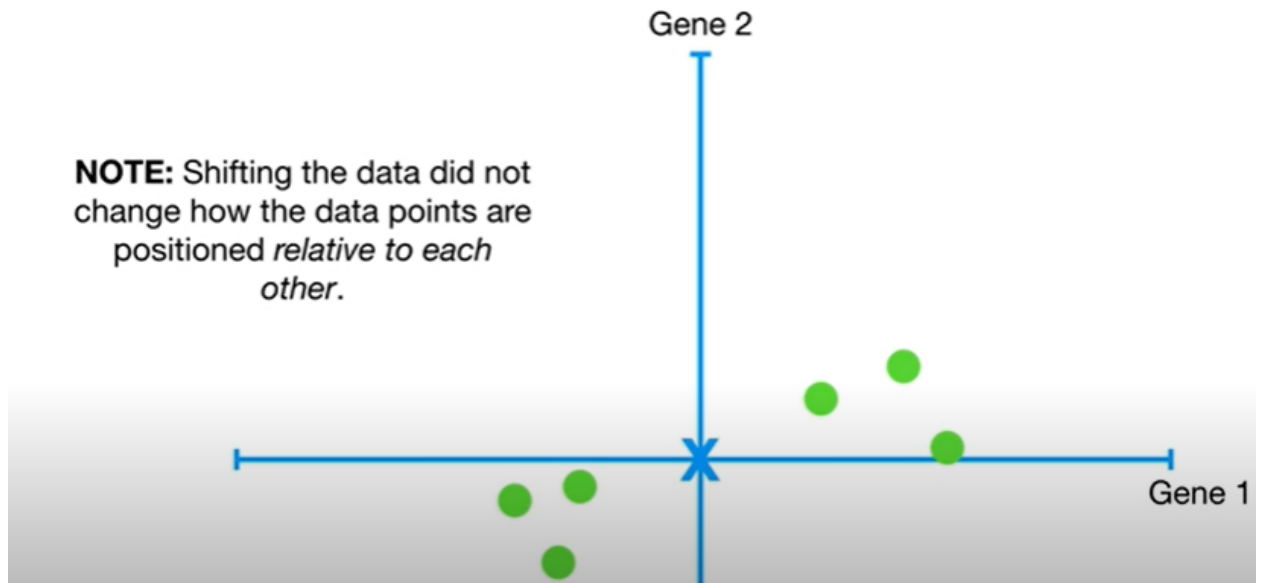
Visualization of data.



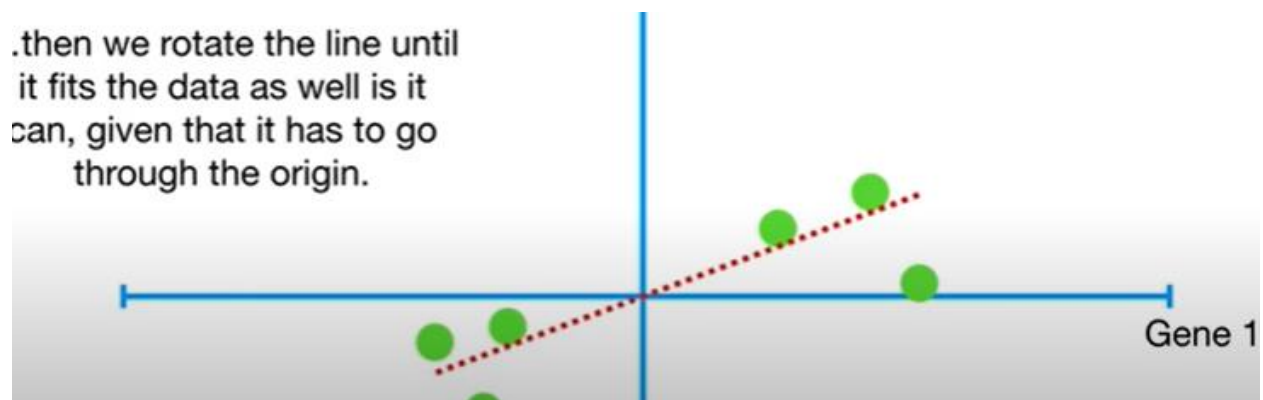
First we plot the data then we calculate the average measurement of Gen1 then for Gen 2.



Now we will shift the data so that the center is on the top of origin.



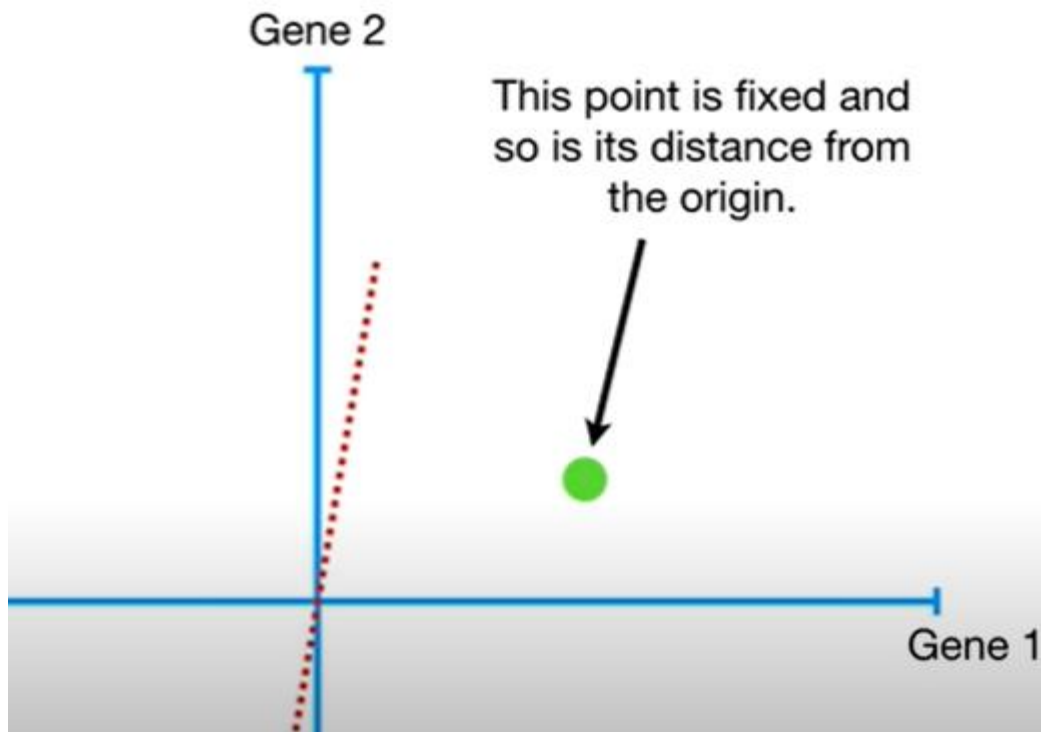
Now we will try to fit the line to it. We will drawing a random line that goes through the origin then we rotate the line until it fits the data well.



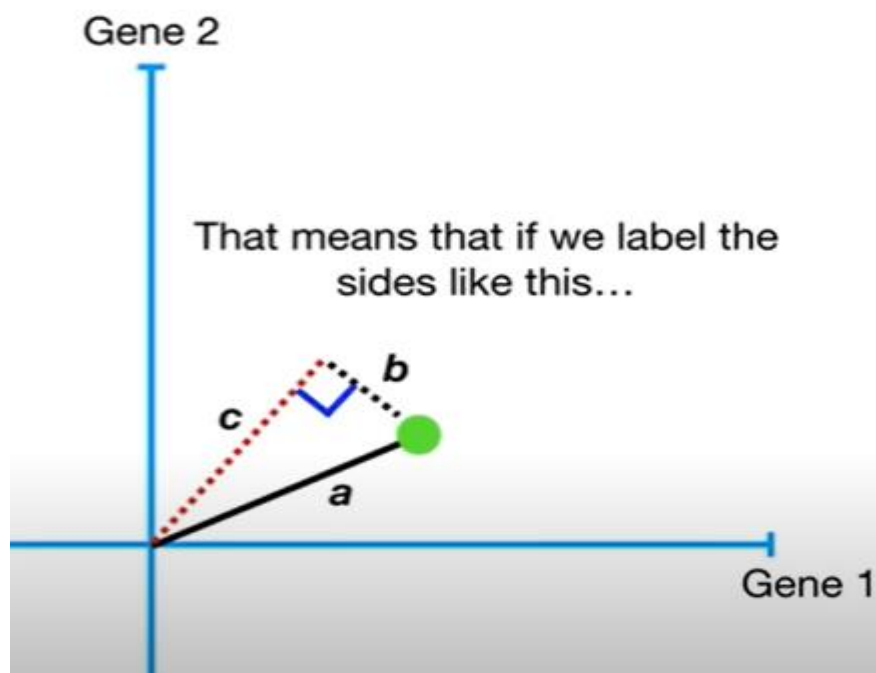
Now the question is how PCA knows which line is fit or not let's look below.

So, first, we project the data onto the line then for the best-fit line, either we minimize the distance between the data points and the line or we try to find the line that maximizes the distances from projected points to the origin.

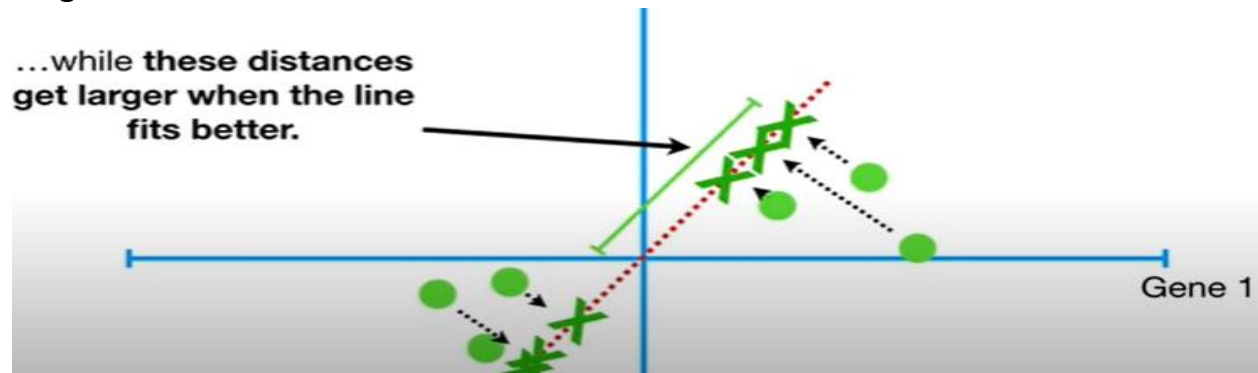
Now lets look into what PCA does to a single point.



So you can see that the distance between the point to the origin remains the same whether the line moves or not.



Now if the line fits best in the data then the b length will decrease and the c length will increase.



So PCA fits the best line by maximizing the sum of squared distances from projected points to the origin. This line is called the **principal component**.

This principal component is a linear combination of all input features.

Now consider if you have 100 features and you cannot visualize it as it's 100D. So you will reduce it to 2D so that you can visualize the data easily. Then you will draw a line called PC1 as explained above. The goal is to find a line that captures the most variation in the data means the majority of data points lie near the PC1. Then you will plot PC2 exactly perpendicular to PC1 without further optimization. Now, you have two lines (PC1 and PC2) that represent the most important directions of variation in your data. Together, they form a new coordinate system. Instead of working with the original 100 parameters, you can now use these two components to represent your data points in a lower-dimensional space.

For example, if the equations for PC1 and PC2 are:

$$\text{PC1: } 0.7 x_1 + 0.3 x_2$$

$$\text{PC2: } -0.3 x_1 + 0.7 x_2$$

Then, for a given data point with original feature values $x_1=3$ and $x_2=4$ you can calculate the corresponding coordinates along PC1 and PC2.

$$X^* = 2.9$$

$$Y^* = 2.5$$

So, x^* and y^* will be the new coordinates of the data point. You can then plot these transformed points to visualize the data in terms of the principal

components. PCA assumes linear relationships and may not always capture complex, non-linear patterns in the data.

More things coming soon !

Prepared by Muhammad Rayyan

Linkedin: [Linkedin](#)

Email: m18rayyan@gmail.com

Medium: [Medium](#)

