Supervised Learning Alog's Questions :

1. Explain Linear Regression in detail.

Linear regression is a type of statistical analysis used to predict the relationship between two variables. It assumes a linear relationship between the independent variable and the dependent variable, and aims to find the best-fitting line that describes the relationship. The line is determined by minimizing the sum of the squared differences between the predicted values and the actual values.

The equation for the Linear model is **Y = mX+c**, where **m is the slope and c is the intercept**. If we have more than 2 independent variables, then it is 'Multiple Linear Regression'.

To get the best fit line, we should chosse optimal values for 'm' & 'c' and we will get this using Loss function and that is the goal of linear regression. It is helpful to determine which model performs better and which parameters are better.

In Regression, the difference between the observed value of the dependent variable(yi) and the predicted value(predicted) is called the **residuals**.

In Linear Regression, generally **Mean Squared Error (MSE)** cost function is used, which is the average of squared error that occurred between the ypredicted and yi.

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

Assumptions of Linear Regression:

- There should be linear relationship between input and output. (Check using Scatter plot)
- No Multi-collinearity. (Check using VIF or Heatmap)
- Homoscedasticity. Variance of residuals should be same. (Check using Scatter plot)
- Normal distribution of residuals. (Check using QQ plot)
- No correlation between residuals. (Check using residual plot)

Advantages:

- Easy to implement.
- Performs well for linearly separable data.
- Can handle overfitting.

Disadvantages:

- Not suitable for data having non-linear relationship.
- Lot of feature engineering needs to be done.
- Prone to noise.
- Sensitive to outliers and missing values.

Linear Regression using OLS:

Ordinary Least Squares(OLS) is one of the methods to estimate the parameters of linear regression model. It aims to find the parameters of linear regression model and minimizes the Sum of squared residuals.

OLS assumes that errors are normally distributed with zero mean and constant variance, and there is no multi-collinearity among independent variables.

To use OLS method, we apply the below formula to find the equation.

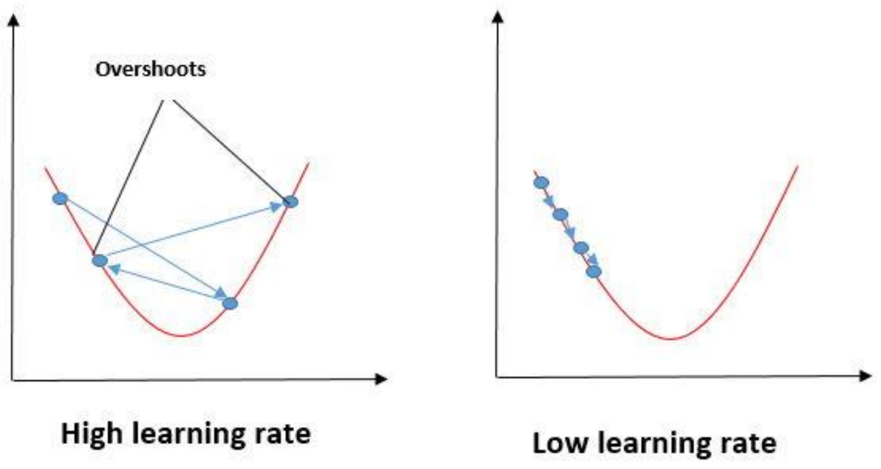
$$m = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

$$b = \bar{y} - m * \bar{x}$$

Linear Regression using Gradient descent:

Gradient Descent is one of the iterative first order optimization algorithms that optimize the cost function(objective function) to reach the optimal minimal solution. To find the optimum solution we need to reduce the cost function(MSE) for all data points. A regression model optimizes the gradient descent algorithm to update the coefficients of the line by reducing the cost function by randomly selecting coefficient values and then iteratively updating the values to reach the minimum cost function.

Let's take an example to understand this. Imagine a U-shaped pit. And you are standing at the uppermost point in the pit, and your motive is to reach the bottom of the pit. Suppose there is a treasure at the bottom of the pit, and you can only take a discrete number of steps to reach the bottom. If you opted to take one step at a time, you would get to the bottom of the pit in the end but, this would take a longer time. If you decide to take larger steps each time, you may achieve the bottom sooner but, there's a probability that you could overshoot the bottom of the pit and not even near the bottom. In the gradient descent algorithm, the number of steps you're taking can be considered as the learning rate, and this decides how fast the algorithm converges to the minima.



Learning Rate is a tuning parameter in optimization algorithm that determines the step size at each iteration while moving towards the minimum of loss function.

Below are the formulas for partial derivatives and updated coefficients.

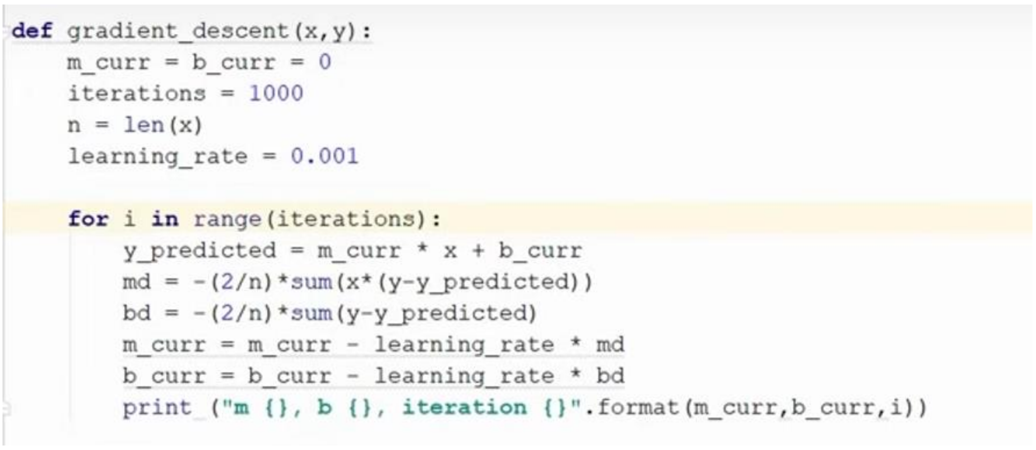
$$ms\varepsilon = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - (mx_i + b)\right)^2$$

$$\partial/_{\partial m} = \frac{2}{\mathrm{n}}\sum_{i=1}^{n} -x_i\left(y_i - (mx_i + b)\right)$$

$$\partial/_{\partial b} = \frac{2}{\mathrm{n}}\sum_{i=1}^{n} -\left(y_i - (mx_i + b)\right)$$

$$\mathrm{m} = \mathrm{m} - \text{learning rate} * \partial/_{\partial m}$$

$$\mathrm{b} = \mathrm{b} - \text{learning rate} * \partial/_{\partial b}$$



```python
def gradient_descent(x,y):
    m_curr = b_curr = 0
    iterations = 1000
    n = len(x)
    learning_rate = 0.001

    for i in range(iterations):
        y_predicted = m_curr * x + b_curr
        md = -(2/n)*sum(x*(y-y_predicted))
        bd = -(2/n)*sum(y-y_predicted)
        m_curr = m_curr - learning_rate * md
        b_curr = b_curr - learning_rate * bd
        print ("m {}, b {}, iteration {}".format(m_curr,b_curr,i))
```

Here are some differences between OLS and gradient descent:

- OLS is non-iterative, while gradient descent is iterative.
- OLS uses partial differentiation to find the minima of the equation, while gradient descent uses a learning rate along with partial differentiation.
- OLS only works with one specific objective function, while gradient descent can work with any arbitrary objective function.
- OLS tries to solve a closed form solution, while gradient descent does not.

<u>Evaluation Metrics for Linear Regression:</u>

a) Mean Squared Error (MSE):

Mean squared error states that finding the squared difference between actual and predicted value.We perform squared to avoid the cancellation of negative terms and it is the benefit of MSE.

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

Advantages:

- Most common and easy to interpret.
- The graph of MSE is differentiable, so you can easily use it as a loss function

Disadvantages:

- Sensitive to outliers.
- The value you get after calculating MSE is a squared unit of output. for example, the output variable is in meter(m) then after calculating MSE the output we get is in meter squared.

b) Root Mean Squared Error (RMSE):

As RMSE is clear by the name itself, that it is a simple square root of mean squared error. We can use this if there are any large errors i.e., model overstimates the prediction or underestimates the prediction.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

Advantages:

- More interpretable than MSE and not sensitive to outliers.
- The output value you get is in the same unit as the required output variable which makes interpretation of loss easy.

Disadvantages:

- Not as common as MSE.

c) Mean Absolute Error (MAE):

MAE is a simple metric which calculates the absolute difference between actual and predicted values.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} | y_i - \hat{y}_i |$$

Advantages:

- The MAE you get is in the same unit as the output variable.
- It is most Robust to outliers than MSE and RMSE.

Disadvantages:

- The graph of MAE is not differentiable so we have to apply various optimizers like Gradient descent which can be differentiable.

d) R-squared (R2):

R2 score is a metric that tells the performance of your model, not the loss in an absolute sense that how many wells did your model perform. In contrast, MAE and MSE depend on the context as we have seen whereas the R2 score is independent of context.

Hence, R2 squared is also known as Coefficient of Determination or sometimes also known as Goodness of fit.

$$R^2 = 1 - \frac{RSS}{TSS}$$

R2 is scaled metric. It means, value of R2 will change if data gets scaled. It measures the proportion of variance explained by the model. It is bit difficult to interpret.

e) Adjusted R-squared:

The disadvantage of the R2 score is while adding new features in data the R2 score starts increasing or remains constant but it never decreases because It assumes that while adding more data variance of data increases.

But the problem is when we add an irrelevant feature in the dataset then at that time R2 sometimes starts increasing which is incorrect.

Hence, To control this situation Adjusted R Squared came into existence.

$$Adjusted\ R^2 = 1 - \frac{(1 - R^2)(N-1)}{N-p-1}$$

Where

R^2 Sample R-Squared

N Total Sample Size

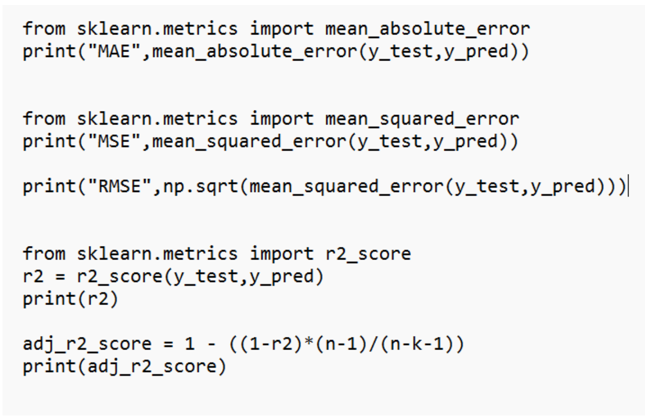
p Number of independent variable

Now as 'p' increases by adding some features so the denominator will decrease, n-1 will remain constant. R2 score will remain constant or will increase slightly so the complete answer will increase and when we subtract this from one then the resultant score will decrease. so this is the case when we add an irrelevant feature in the dataset.

And if we add a relevant feature then the R2 score will increase and 1-R2 will decrease heavily and the denominator will also decrease so the complete term decreases, and on subtracting from one the score increases.

<u>Advantages:</u>

- More interpretable than R2 and no.of predictors will consider.



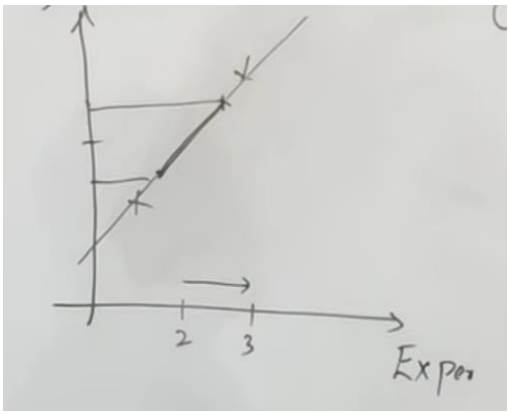
```
from sklearn.metrics import mean_absolute_error
print("MAE",mean_absolute_error(y_test,y_pred))


from sklearn.metrics import mean_squared_error
print("MSE",mean_squared_error(y_test,y_pred))

print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))


from sklearn.metrics import r2_score
r2 = r2_score(y_test,y_pred)
print(r2)

adj_r2_score = 1 - ((1-r2)*(n-1)/(n-k-1))
print(adj_r2_score)
```

[Linear Regression | Everything you need to Know about Linear Regression]

2. Explain Lasso & Ridge Regularization.

Problem with Linear Regression:

The main aim of any machine learning model is to create generalized model. Let's say I have plotted my data points as shown below and in this case my training error was zero. I can also see that there is huge steep in my slope w.r.t movement in unit change of X. If I don't add any penality to my cost function, my error was zero and I will consider it as the best fit line and it overfits. If I add some penality, then there will some error so I will not stop my training and I will see for other best line and steepness will go down. Here we are penalizing the features which are having higher slopes.



One more problem with linear regression is that estimated coefficients of the model may become large and making model sensitive to inputs and unstable. This problem occurs mainly for the data with fewer samples. These issues can be addressed using Regularization.

What is Regularization?

Regularization is one of the ways to improve our model to work on unseen data by ignoring the less important features. It avoids overfitting by adding a penalty to the model with high variance, thereby shrinking the beta coefficients to zero.

a) Lasso Regularization (L1):

Stands for **Least Absolute Shrinkage and Selection Operator** and adds L1 penality to the loss function.

Lasso shrinks the less important feature's coefficient to zero, thus removing some features altogether. So, this works well for feature selection in case we have a huge number of features. Along with shrinking coefficients, the lasso performs feature selection, as well. Because some of the coefficients become exactly zero, which is equivalent to the particular feature being excluded from the model.

$$\text{Cost function} = \text{Loss} + \lambda + \Sigma\ ||w||$$

b) Ridge Regularization (L2):

Regularization adds the penalty as model complexity increases. The regularization parameter (lambda) penalizes all the parameters except intercept so that the model generalizes the data and won't overfit. Ridge regression adds "squared magnitude of the coefficient" as penalty term to the loss function.

$$\text{Cost function} = \text{Loss} + \lambda + \Sigma\ ||w||^2$$

```
Loss = sum of squared residual
λ = penalty
w = slope of the curve
```

If lambda is zero, then it is equivalent to OLS. But if lambda is very large, then it will add too much weight, and it will lead to under-fitting.

Ridge regularization forces the weights to be small but does not make them zero and does not give the sparse solution.

Ridge is not robust to outliers as square terms blow up the error differences of the outliers, and the regularization term tries to fix it by penalizing the weights.

L2 regularization can learn complex data patterns.

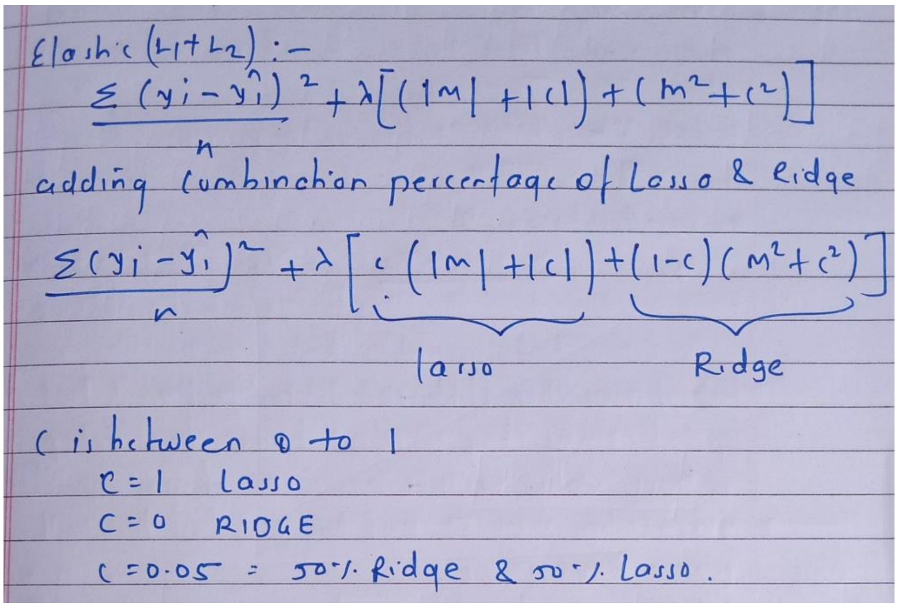
Ridge is mainly going to focus on Overfitting and Lasso is mainly going to focus on feature selection.

Lasso(L1)	**Ridge(L2)**
1. Penality is absolute value of coefficients.	1. Penality is the square of coefficients.
2. Estimates the median of the data.	2. Estimates the mean of the data.
3. Shrinks coefficients to Zero.	3. Shrinks coefficients equally.
4. Used for feature selection and dimensionality reduction.	4. Useful if we have collinear features.

If you are working on high dimensional data and if some features are not important then you can prefer Lasso over Ridge.

c) Elastic Net:

It is the combination of both Lasso and Ridge.



Elastic (L1+L2) :-

$$\frac{\sum (y_i - \hat{y}_i)^2}{n} + \lambda\left[(|m| + |c|) + (m^2 + c^2)\right]$$

adding combination percentage of Lasso & Ridge

$$\frac{\sum (y_i - \hat{y}_i)^2}{n} + \lambda\left[\underbrace{c \cdot (|m| + |c|)}_{\text{lasso}} + \underbrace{(1-c)(m^2 + c^2)}_{\text{Ridge}}\right]$$

c is between 0 to 1

c = 1 Lasso

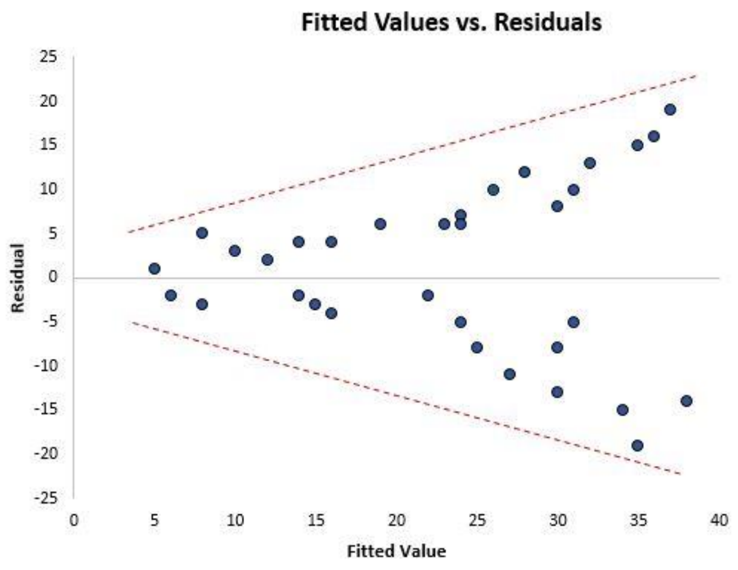
C = 0 RIDGE

c = 0.05 = 50% Ridge & 50% Lasso.

https://www.youtube.com/watch?v=9lRv01HDU0s

3. How do you treat heteroscedasticity in Regression ?

Heteroscedasticity means unequal scattered distribution. In regression analysis, we generally talk about the heteroscedasticity in the context of the error term. Heteroscedasticity is the systematic change in the spread of the residuals or errors over the range of measured values. Heteroscedasticity is the problem because Ordinary least squares (OLS) regression assumes that all residuals are drawn from a random population that has a constant variance.



<u>What causes Heteroscedasticity?</u>

Heteroscedasticity occurs more often in datasets, where we have a large range between the largest and the smallest observed values. There are many reasons why heteroscedasticity can exist, and a generic explanation is that the error variance changes proportionally with a factor.

We can categorize Heteroscedasticity into two general types:-

Pure heteroscedasticity : It refers to cases where we specify the correct model and let us observe the non-constant variance in residual plots.

Impure heteroscedasticity: It refers to cases where you incorrectly specify the model, and that causes the non-constant variance. When you leave an important variable out of a model, the omitted effect is absorbed into the error term. If the effect of the omitted variable varies throughout the observed range of data, it can produce the telltale signs of heteroscedasticity in the residual plots.

<u>How to Fix Heteroscedasticity?</u>

- Transform the dependent variable
- Redefine the dependent variable
- Use weighted regression

https://www.statology.org/heteroscedasticity-regression/

4. Explain Logistic Regression in detail.

Logistic Regression is a classification algorithm, used to classify elements of a set into two groups (binary classification) by calculating the probability of each element of the set. Logistic Regression is the appropriate regression analysis to conduct when the dependent variable has a binary solution, we predict the values of categorical variables.

Limitations of Linear Regression for classification:

In Linear Regression, target variable is continuous, whereas in logistic regression, target is binary. If we have any outlier in our linear regression model, then our best fit line shifts to that point.

Another problem with linear regression is that the predicted values may be out of range. We know that probability can be between 0 and 1, but if we use linear regression this probability may exceed 1 or go below 0.

To overcome these problems we use Logistic Regression, which converts this straight best fit line in linear regression to an S-curve using the sigmoid function, which will always give values between 0 and 1.

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

What are odds?

Odds are nothing but the ratio of probability of success to probability of failures (P/(1-P))

Multiclass vs Multilabel classification:

In multi-class classification, each input have only one output class. Eg: Dog or cant but can’t be both,

In multi-label classification, eacn input can have multiple output classes. Eg: Genre of movie, it could be action,thriller,comedy.

Advantages:

- Easy to implement and understand.
- Less inclined to overfitting and requires less training.
- Easy to interpret and gives good accuracy.

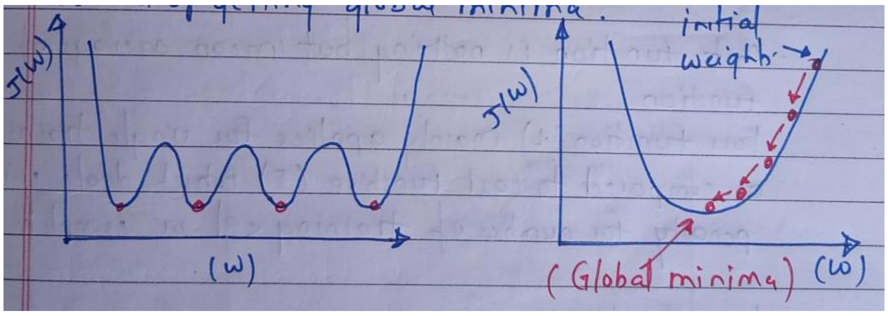
Disadvantages:

- Lots of feature engineering needs to be done.
- High dimensional data causes overfitting.
- Sensitives to outliers and missing values.
- Requires large datasets.

Loss function in Logistic Regression:

Why don’t we use MSE as loss function in logistic regression ?

In linear regression, points which we plot are linear. Whereas in logistic regression, we deal with non-linearity and because of this our error function becomes non linear which leads to many local minima values instead of getting global minima.



In logistic regression we use Logloss or Binary Cross Entropy as loss function. Logloss is the classification metric based on probabilities.

$$\text{Log loss} = \frac{1}{N} \sum_{i=1}^{N} -(y_i * \log(\hat{Y}_i) + (1-y_i) * \log(1-\hat{Y}_i))$$

https://www.analyticsvidhya.com/blog/2020/12/beginners-take-how-logistic-regression-is-related-to-linear-regression/#h-comparison-of-linear-regression-logistic-regression

5. Differences between Linear Regression and Logistic Regression.

Similarities:

- Linear Regression and Logistic Regression both are supervised Machine Learning algorithms.
- Linear Regression and Logistic Regression, both the models are parametric regression i.e. both the models use linear equations for predictions.

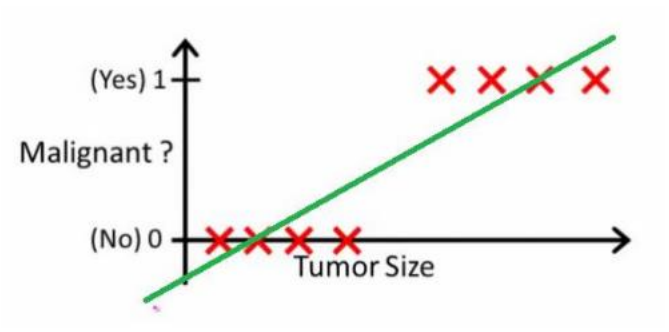
Differences:

- Linear Regression is used to handle regression problems whereas Logistic regression is used to handle the classification problems.
- Linear regression models data using continuous numeric value. As against, logistic regression models the data in the binary values.
- The purpose of Linear Regression is to find the best-fitted line which is a straight line while Logistic regression is one step ahead and fitting the line values to the sigmoid curve.
- The method for calculating loss function in linear regression is the mean squared error whereas for logistic regression it is Logloss.
- Linear regression requires to establish the linear relationship among dependent and independent variables, whereas it is not necessary for logistic regression.
- In linear regression, the independent variable can be correlated with each other. On the contrary, in the logistic regression, the variable must not be correlated with each other.

https://www.simplilearn.com/tutorials/machine-learning-tutorial/linear-regression-vs-logistic-regression#linear_vs_logistic_regression_differences

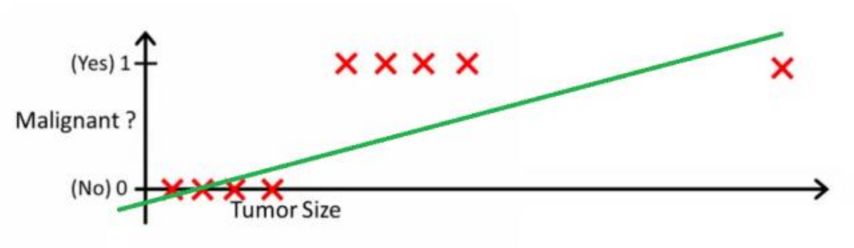
6. Why we can't do a classification problem using Regression?

With linear regression you fit a polynomial through the data - say, like on the example below, we fit a straight line through {tumor size, tumor type} sample set:



Above, malignant tumors get 1, and non-malignant ones get 0, and the green line is our hypothesis h(x). To make predictions, we may say that for any given tumor size x, if h(x) gets bigger than 0.5, we predict malignant tumors. Otherwise, we predict benignly. It looks like this way, we could correctly predict every single training set sample, but now let's change the task a bit.

Intuitively it's clear that all tumors larger certain threshold are malignant. So let's add another sample with huge tumor size, and run linear regression again:



Now our h(x)>0.5→malignant doesn't work anymore. To keep making correct predictions, we need to change it to h(x)>0.2 or something - but that's not how the algorithm should work.

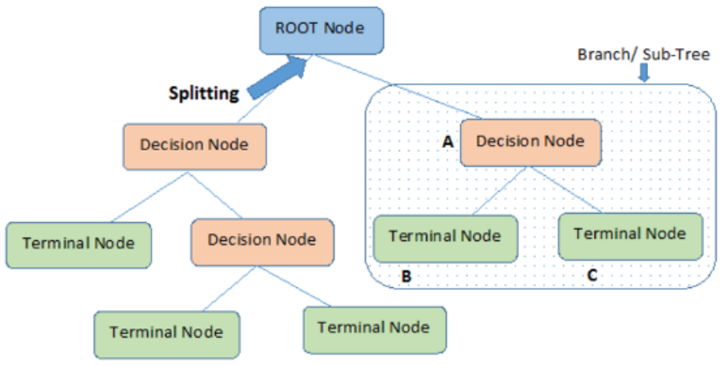
We cannot change the hypothesis each time a new sample arrives. Instead, we should learn it off the training set data, and then (using the hypothesis we've learned) make correct predictions for the data we haven't seen before.

7. Explain about Decision Tree in detail.

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

The input to a decision tree can be both continuous as well as categorical. The decision tree works on an if-then statement. Decision tree tries to solve a problem by using tree representation. The logic behind selecting the decision tree is it can be easily understood because it shows a tree-like structure.

- *Step-1:* Begin the tree with the root node, says S, which contains the complete dataset.
- *Step-2:* Find the best attribute in the dataset using Attribute Selection Measure (ASM).
- *Step-3:* Divide the S into subsets that contains possible values for the best attributes.
- *Step-4:* Generate the decision tree node, which contains the best attribute.
- *Step-5:* Recursively make new decision trees using the subsets of the dataset created in step3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.



In order to build a tree, we use either CART or ID3 algorithms.

CART and ID3 are both decision tree algorithms. CART stands for **Classification and Regression Trees**, while ID3 stands for **Iterative Dichotomizer 3**. CART only produces binary trees, where non-leaf nodes always have two children. ID3 can produce decision trees with nodes having more than two children.

Here are some other differences between CART and ID3:

- CART uses Gini Impurity, while ID3 uses Entropy and Information Gain.
- CART is used for continuous features and continuous labels, while ID3 is used for categorical features and categorical labels.
- CART performs better than ID3 and C4.5 in terms of classifier accuracy. However, CART trees are larger and harder to interpret.

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as Attribute selection measure or ASM. By this measurement, we can easily select the best attribute for the nodes of the tree. There are four popular techniques for ASM, which are:

- Information Gain
- Entropy
- Gini Index
- Reduction in Variance

1. Information Gain:

Information Gain calculates how much information a feature provides us about a class. According to the value of information gain, we split the node and build the decision tree. A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. The split with the highest information gain will be taken as the first split and the process will continue until all children nodes are pure, or until the information gain is 0.

Information gain computes the difference between entropy before split and average entropy after split of the dataset.

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

2. Entropy:

Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. It is used for calculating the purity of a node. **Lower the value of entropy, higher is the purity of the node**.

Suppose you have a group of friends who decides which movie they can watch together on Sunday. There are 2 choices for movies, one is “Lucy” and the second is “Titanic” and now everyone has to tell their choice. After everyone gives their answer we see that “Lucy” gets 4 votes and “Titanic” gets 5 votes. Which movie do we watch now? Isn’t it hard to choose 1 movie now because the votes for both the movies are somewhat equal.

This is exactly what we call disorderness, there is an equal number of votes for both the movies, and we can’t really decide which movie we should watch. It would have been much easier if the votes for “Lucy” were 8 and for “Titanic” it was 2. Here we could easily say that the majority of votes are for “Lucy” hence everyone will be watching this movie.

$$E(S) = -p_{(+)} \log p_{(+)} - p_{(-)} \log p_{(-)}$$

Here, p+ is the probability of positive class
p– is the probability of negative class
S is the subset of the training example

Goal of DT is to decrease the uncertainty or impurity in the dataset, here by using the entropy we are getting the impurity of a particular node.

3. Gini Index:

Gini Impurity is bit similar to Entropy, measures the purity of the split when the target variable is categorical. *Gini is the probability of correctly labeling a randomly chosen element if it was randomly labeled according to the distribution of labels in the node.*

It measures the likelihood that randomly selected data point would be incorrectly classified by specific node.

$$GI = 1 - \sum_{i=1}^{n} (p)^2$$

$$GI = 1 - \left[(P_{(+)})^2 + (P_{(-)})^2 \right]$$

- Gini's max impurity is 0.5 wheras for entropy, max impurity is 1.
- In terms of computation, entropy takes more time as it includes logarithms.

4. Reduction in Variance:

Reduction in Variance is a method for splitting the node used when the target variable is continuous, i.e., regression problems. It is so-called because it uses variance as a measure for deciding the feature on which node is split into child nodes.

$$Variance = \frac{\sum (X - \mu)^2}{N}$$

Variance is used for calculating the homogeneity of a node. If a node is entirely homogeneous, then the variance is zero.

When to stop Splitting ?

Usually, real-world datasets have a large number of features, which will result in a large number of splits, which in turn gives a huge tree. Such trees take time to build and can lead to overfitting. That means the tree will give very good accuracy on the training dataset but will give bad accuracy in test data.

There are many ways to tackle this problem through hyperparameter tuning. We can set the maximum depth of our decision tree using the **max_depth** parameter. The more the value of max_depth, the more complex your tree will be. The training error will off-course decrease if we increase the max_depth value but when our test data comes into the picture, we will get a very bad accuracy. Hence you need a value that will not overfit as well as underfit our data and for this, you can use GridSearchCV.

Another way is to set the minimum number of samples for each spilt. It is denoted by **min_samples_split**. Here we specify the minimum number of samples required to do a spilt. For example, we can use a minimum of 10 samples to reach a decision. That means if a node has less than

10 samples then using this parameter, we can stop the further splitting of this node and make it a leaf node.

There are more hyperparameters such as :

min_samples_leaf : It represents the minimum number of samples required to be in the leaf node. The more you increase the number, the more is the possibility of overfitting.
max_features : It helps us decide what number of features to consider when looking for the best split.

<u>Pruning :</u>

Pruning is another method that can help us avoid overfitting. Pruning is the process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.It helps in improving the performance of the tree by cutting the nodes or sub-nodes which are not significant. Additionally, it removes the branches which have very low importance. A too large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning.

There are mainly 2 ways for pruning:

- Pre-pruning : It is also known as the early stopping criteria. We can stop growing the tree earlier, which means we can prune/remove/cut a node if it has low importance while growing the tree.
- Post-pruning : In Post-pruning, the idea is to allow the decision tree to grow fully and observe the CP value. Next, we prune/cut the tree with the optimal CP(Complexity Parameter) value as the parameter. Once our tree is built to its depth, we can start pruning the nodes based on their significance.

<u>Advantages:</u>

- It is simple to understand as it follows the same process which humans follow while making any decision in real-life.
- Output can be easily interpreted and training period is less than Random Forest.
- Can handle both continuous and categorical variables.
- Decision trees implicitly perform variable screening or feature selection, so explicitly feature scaling is not required.
- Robust to outliers and handles missing values.
- There is less requirement of data cleaning compared to other algorithms.

<u>Disadvantages:</u>

- Leads to overfitting and high Variance. (It can be controlled by Pruning)
- The decision tree contains lots of layers, which makes it complex.
- Small changes in the data alter the tree structure and leads to instability.

https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/#Information_Gain

https://iq.opengenus.org/id3-algorithm/

DT%20Problem.xlsx

8. How to control leaf height and Pruning?

To control the leaf size, we can set the parameters:

1. Maximum depth :
Maximum tree depth is a limit to stop the further splitting of nodes when the specified tree depth has been reached during the building of the initial decision tree.

2. Minimum split size:
Minimum split size is a limit to stop the further splitting of nodes when the number of observations in the node is lower than the minimum split size. This is a good way to limit the growth of the tree. When a leaf contains too few observations, further splitting will result in overfitting (modeling of noise in the data).

3. Minimum leaf size:
Minimum leaf size is a limit to split a node when the number of observations in one of the child nodes is lower than the minimum leaf size.

9. How to handle a decision tree for numerical and categorical data?

Decision trees can handle both categorical and numerical variables at the same time as features. There is not any problem in doing that.

Every split in a decision tree is based on a feature.
- If the feature is categorical, the split is done with the elements belonging to a particular class.
- If the feature is continuous, the split is done with the elements higher than a threshold.

At every split, the decision tree will take the best variable at that moment. This will be done according to an impurity measure with the split branches. And the fact that the variable used to do split is categorical or continuous is irrelevant (in fact, decision trees categorize continuous variables by creating binary regions with the threshold). At last, the good approach is to always convert your categoricals to continuous using LabelEncoder or OneHotEncoding.

10. Explain about Ensemble Learning.

Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model. Thus a collection of models is used to make predictions rather than an individual model.The underlying concept behind ensemble learning is to combine the outputs of diverse models to create a more precise prediction. By considering multiple perspectives and utilizing the strengths of different models, ensemble learning improves the overall performance of the learning system. This approach not only enhances accuracy but also provides resilience against uncertainties in the data. By effectively merging predictions from multiple models, ensemble learning has proven to be a powerful tool in various domains, offering more robust and reliable forecasts.

There are 3 most common ensemble learning methods in machine learning. These are as follows:

- Bagging
- Boosting
- Stacking

Bagging:

The idea behind bagging is combining the results of multiple models (for instance, all decision trees) to get a generalized result. Here's a question: If you create all the models on the same set of data and combine it, will it be useful? There is a high chance that these models will give the same result since they are getting the same input. So how can we solve this problem? One of the techniques is bootstrapping.

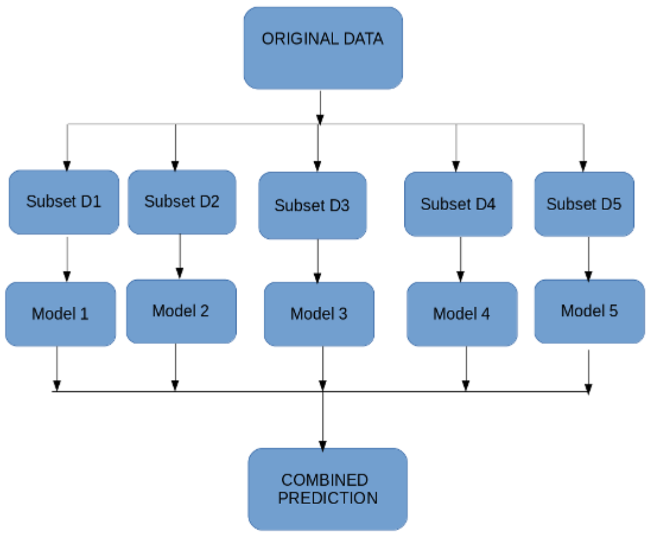
Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset. Hence each model is generated from the samples provided by the Original Data with replacement known as **row sampling**.This step of row sampling with replacement is called **bootstrap**. The size of the subsets is the same as the size of the original set. Now each model is trained independently, which generates results. The final output is based on majority voting after combining the results of all models. This step which involves combining all the results and generating output based on majority voting, is known as **aggregation**.

Bootstrapping: It is a random sampling method that is used to derive samples from the data using the replacement procedure. In this method, first, random data samples are fed to the primary model, and then a base learning algorithm is run on the samples to complete the learning process.

Aggregation: This is a step that involves the process of combining the output of all base models and, based on their output, predicting an aggregate result with greater accuracy and reduced variance.

Bagging (or Bootstrap Aggregation) technique uses these subsets (bags) to get a fair idea of the distribution (complete set). The size of subsets created for bagging may be less than the original set.

1. Multiple subsets are created from the original dataset, selecting observations with replacement.
2. A base model (weak model) is created on each of these subsets.
3. The models run in parallel and are independent of each other.
4. The final predictions are determined by combining the predictions from all the models.



<u>Boosting:</u>

Before we go further, here's another question for you: If a data point is incorrectly predicted by the first model, and then the next (probably all models), will combining the predictions provide better results? Such situations are taken care of by boosting.

Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model. The succeeding models are dependent on the previous model. Let's understand the way boosting works in the below steps.

1. A subset is created from the original dataset.
2. Initially, all data points are given equal weights.
3. A base model is created on this subset.
4. This model is used to make predictions on the whole dataset.
5. Errors are calculated using the actual values and predicted values.
6. The observations which are incorrectly predicted, are given higher weights.
7. Another model is created and predictions are made on the dataset.
8. Similarly, multiple models are created, each correcting the errors of the previous model.
9. The final model (strong learner) is the weighted mean of all the models (weak learners).

Thus, the boosting algorithm combines a number of weak learners to form a strong learner. The individual models would not perform well on the entire dataset, but they work well for some part of the dataset. Thus, each model actually boosts the performance of the ensemble.

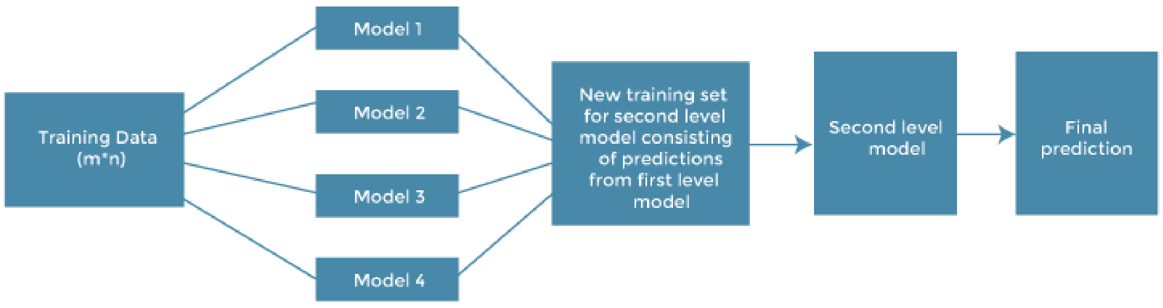
<u>Stacking:</u>

Stacking is one of the popular ensemble modeling techniques in machine learning. Various weak learners are ensembled in a parallel manner in such a way that by combining them with Meta learners, we can predict better predictions for the future.

This ensemble technique works by applying input of combined multiple weak learners' predictions and Meta learners so that a better output prediction model can be achieved.

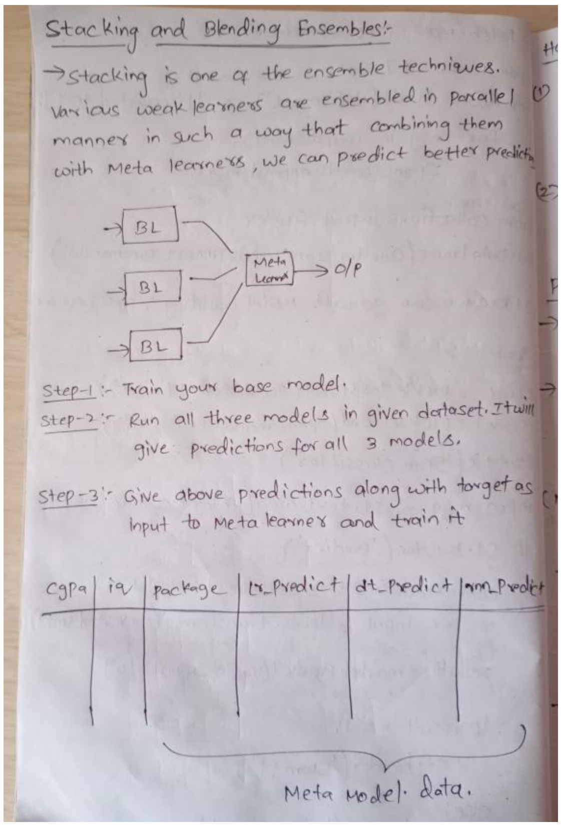
Stacking Architecture:

The architecture of the stacking model is designed in such as way that it consists of two or more base/learner's models and a meta-model that combines the predictions of the base models. These base models are called level 0 models, and the meta-model is known as the level 1 model. So, the Stacking ensemble method includes original (training) data, primary level models, primary level prediction, secondary level model, and final prediction. The basic architecture of stacking can be represented as shown below the image.


Training Data (m*n)
Model 1
Model 2
Model 3
Model 4
New training set for second level model consisting of predictions from first level model
Second level model
Final prediction

https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/

https://www.javatpoint.com/stacking-in-machine-learning



Stacking and Blending Ensembles:-

→ Stacking is one of the ensemble techniques. Various weak learners are ensembled in parallel manner in such a way that combining them with Meta learners, we can predict better predictions

$$\begin{tikzcd}
\arrow[r] & \boxed{BL} \arrow[dr] & & \\
\arrow[r] & \boxed{BL} \arrow[r] & \boxed{\text{Meta Learner}} \arrow[r] & o/p \\
\arrow[r] & \boxed{BL} \arrow[ur] & &
\end{tikzcd}$$

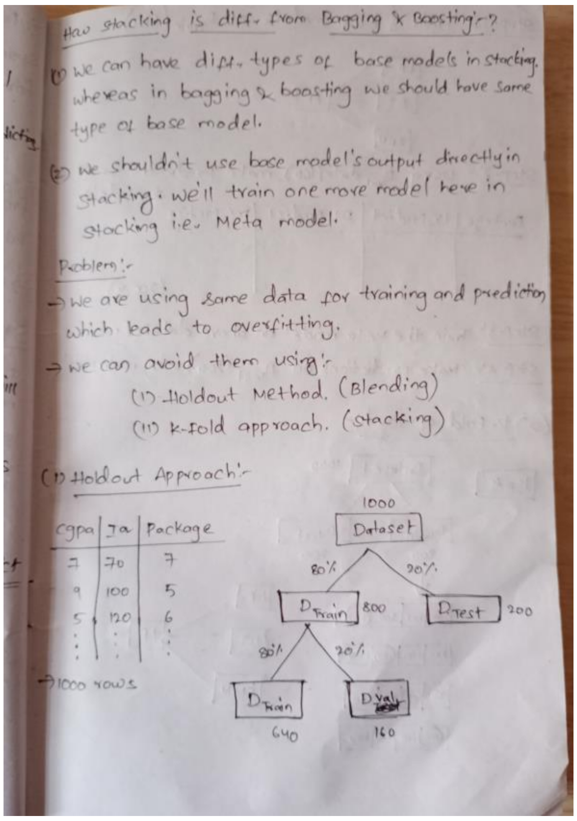
Step-1 :- Train your base model.

Step-2 :- Run all three models in given dataset. It will give predictions for all 3 models.

Step-3 :- Give above predictions along with target as input to Meta learner and train it

cgpa	iq	package	lr_Predict	dt_Predict	knn_Predict

$\underbrace{\hspace{6em}}_{\text{Meta Model data.}}$



How stacking is diff. from Bagging & Boosting:-?

(1) We can have diff. types of base models in stacking. whereas in bagging & boosting we should have same type of base model.

(2) We shouldn't use base model's output directly in stacking. we'll train one more model here in stacking i.e. Meta model.

Problem:-

→ We are using same data for training and prediction which leads to overfitting.

→ We can avoid them using:-

(I) Holdout Method. (Blending)

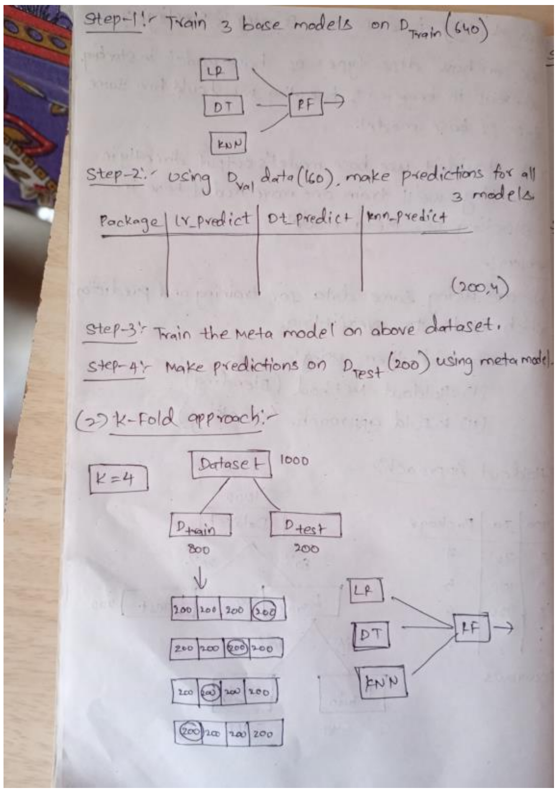
(II) K-fold approach. (stacking)

(1) Holdout Approach:-

cgpa	Iq	Package
7	70	7
9	100	5
5	120	6
⋮	⋮	⋮

→ 1000 rows

$$\begin{tikzcd}
 & \overset{1000}{\boxed{\text{Dataset}}} \arrow[dl, "80\%"'] \arrow[dr, "20\%"] & \\
\boxed{D_{Train}}\ 800 \arrow[d, "80\%"'] \arrow[dr, "20\%"] & & \boxed{D_{Test}}\ 200 \\
\underset{640}{\boxed{D_{Train}}} & \underset{160}{\boxed{D_{val}}} &
\end{tikzcd}$$



Step-1:- Train 3 base models on D_{Train}(640)

$$\begin{tikzcd}
\boxed{LR} \arrow[dr] & & \\
\boxed{DT} \arrow[r] & \boxed{RF} \arrow[r] & {} \\
\boxed{KNN} \arrow[ur] & &
\end{tikzcd}$$

Step-2:- Using D_{val} data(160), make predictions for all 3 models.

Package	lr_predict	Dt_predict	knn_predict

(200,4)

Step-3:- Train the Meta model on above dataset.

Step-4:- Make predictions on D_{Test}(200) using meta model.

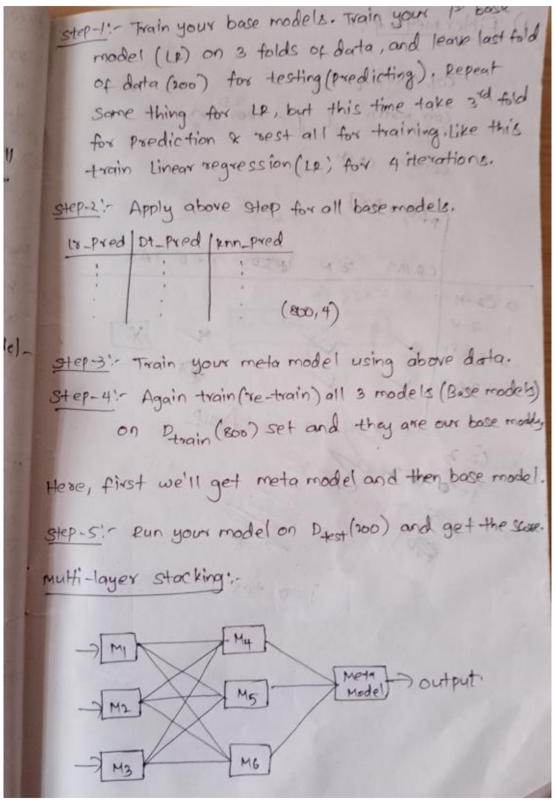
(2) K-Fold approach:-

$\boxed{K=4}$

$$\begin{tikzcd}
 & \boxed{\text{Dataset}}\ 1000 \arrow[dl] \arrow[dr] & \\
\underset{800}{\boxed{D_{train}}} \arrow[d] & & \underset{200}{\boxed{D_{test}}} \\
{}
\end{tikzcd}$$

200	200	200	(200)
200	200	(200)	200
200	(200)	200	200
(200)	200	200	200

$$\begin{tikzcd}
\boxed{LR} \arrow[dr] & & \\
\boxed{DT} \arrow[r] & \boxed{RF} \arrow[r] & {} \\
\boxed{KNN} \arrow[ur] & &
\end{tikzcd}$$



Step-1:- Train your base models. Train your 1st base model (LR) on 3 folds of data, and leave last fold of data (200) for testing (predicting). Repeat same thing for LR, but this time take 3rd fold for prediction & rest all for training. Like this train Linear regression (LR) for 4 iterations.

Step-2:- Apply above step for all base models.

lr_Pred	Dt_Pred	knn_pred
⋮	⋮	⋮

(800,4)

Step-3:- Train your meta model using above data.

Step-4:- Again train (re-train) all 3 models (Base models) on D_{train}(800) set and they are our base models.

Here, first we'll get meta model and then base model.

Step-5:- Run your model on D_{test}(200) and get the score.

Multi-layer stacking:-

$$\begin{tikzcd}
\arrow[r] & \boxed{M_1} \arrow[r] \arrow[dr] \arrow[ddr] & \boxed{M_4} \arrow[dr] & & \\
\arrow[r] & \boxed{M_2} \arrow[ur] \arrow[r] \arrow[dr] & \boxed{M_5} \arrow[r] & \boxed{\text{Meta Model}} \arrow[r] & \text{output} \\
\arrow[r] & \boxed{M_3} \arrow[uur] \arrow[ur] \arrow[r] & \boxed{M_6} \arrow[ur] & &
\end{tikzcd}$$

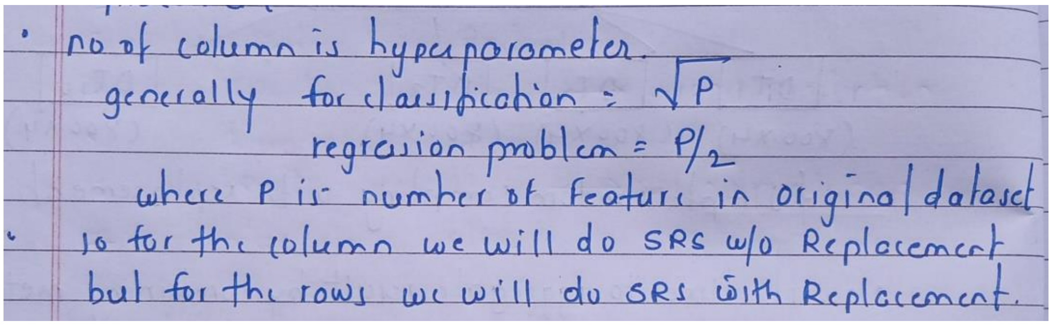
11. Explain about Random Forest.

Random Forest is another ensemble machine learning algorithm that follows the bagging technique. It is an extension of the bagging estimator algorithm. The base estimators in random forest are decision trees. Unlike bagging meta estimator, random forest randomly selects a set of features which are used to decide the best split at each node of the decision tree.

Looking at it step-by-step, this is what a random forest model does:

1. In the Random forest model, a subset of data points and a subset of features is selected for constructing each decision tree. Simply put, n random records and m features are taken from the data set having k number of records.
2. Individual decision trees are constructed for each sample.
3. Each decision tree will generate an output.
4. Final output is considered based on Majority Voting or Averaging .

If number of features increases, then complexity of creating descision tree also increases. To avoid this, we can use Random Forest. Instead of selecting random number of rows, we will select random number of columns. Here, no.of columns, no.of trees, no.of data points are hyperparameters.



• no of column is hyperparameter.
generally for classification = $\sqrt{P}$
regression problem = $P/2$
where P is number of feature in original dataset
• so for the column we will do SRS w/o Replacement
but for the rows we will do SRS with Replacement.

<u>Why it is called Random sampling?</u>
Because, everytime we select the features randomly.

To sum up, the Random forest randomly selects data points and features and builds multiple trees. Random Forest is also used for feature importance selection. The attribute (.feature_importances_) is used to find feature importance.

<u>Advantages:</u>

- Doesn't overfit and best for kaggle competitions.
- No feature scaling and robust to outliers.
- It performs well even if the data contains null/missing values.
- It can be used in classification and regression problems.
- It solves the problem of overfitting as output is based on majority voting or averaging.
- Each decision tree created is independent of the other; thus, it shows the property of parallelization.
- It is immune to the curse of dimensionality. Since each tree does not consider all the attributes, feature space is reduced.

Disadvantages:

- Biased in multi-class problems towards most frequent classes.
- Training time is more than other models due to its complexity. Whenever it has to make a prediction, each decision tree has to generate output for the given input data.

Decision tree vs Random Forest:

Decision Tree	Random Forest
1.Decision trees normally suffer from the problem of overfitting if it's allowed to grow without any control.	1.Random forests are created from subsets of data, and the final output is based on average or majority ranking; hence the problem of overfitting is taken care of.
2.A single decision tree is faster in computation.	2.It is comparatively slower.
3.When a data set with features is taken as input by a decision tree, it will formulate some rules to make predictions.	3.Random forest randomly selects observations, builds a decision tree, and takes the average result. It doesn't use any set of formulas.

Thus random forests are much more successful than decision trees only if the trees are diverse and acceptable.

https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

12. Explain about SVM in detail.

SVM is a supervised machine learning problem where we try to find a hyperplane that best separates the two classes. It works for both classification and regression problems.

"Don't get confused between SVM and logistic regression. Both the algorithms try to find the best hyperplane, but the main difference is logistic regression is a probabilistic approach whereas support vector machine is based on statistical approaches."

Types of SVM:

- Linear SVM:
  When the data is perfectly linearly separable, then only we can use Linear SVM. Perfectly linearly separable means that the data points can be classified into 2 classes by using a single straight line(if 2D).

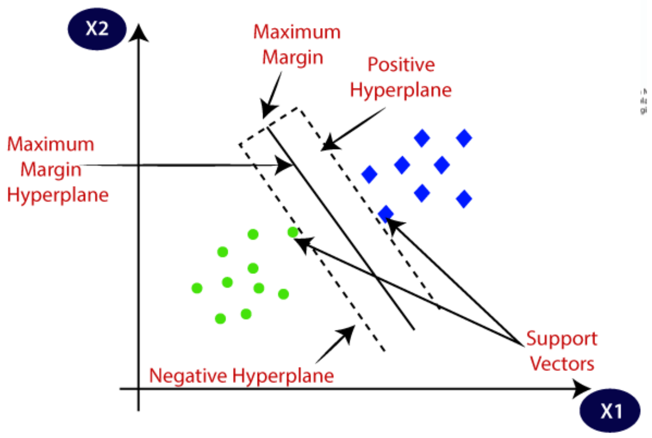
- Non-linear SVM:
  When the data is not linearly separable then we can use Non-Linear SVM, which means when the data points cannot be separated into 2 classes by using a straight line (if 2D) then we use some advanced techniques like kernel tricks to classify them. In most real-world applications we do not find linearly separable datapoints hence we use kernel trick to solve them.

There can be an infinite number of hyperplanes passing through a point and classifying the two classes perfectly. So, which one is the best? SVM does this by finding the maximum margin between the hyperplanes that means maximum distances between the two classes, it is the best hyperplane.

Hyperplane: Hyperplane is a line (in 2D) or a plane that separates the data points into two classes.

Support Vectors: These are the points that are closest to the hyperplane. A separating line will be defined with the help of these data points.

Margin: it is the distance between the hyperplane and the observations closest to the hyperplane (support vectors). In SVM large margin is considered a good margin. There are two types of margins hard margin and soft margin.



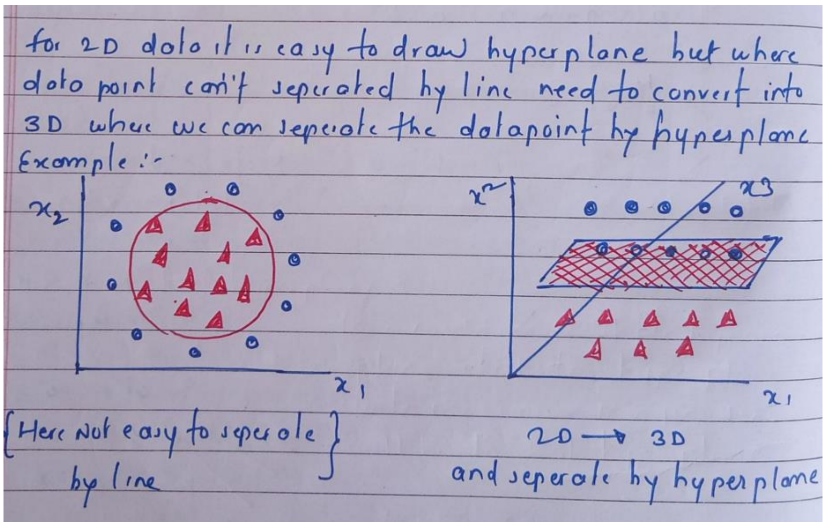
Advantages:

- Works well with smaller data.
- Works well with higher dimensional data, effective even number of features are quite large.
- Non-Linear data can also be classified using customized hyperplanes built by using kernel trick.
- Kernel trick is strength and we can solve any complex problem.
- Overfitting is less in SVM.

Disadvantages:

- Not suitable for larger datasets as training time would be very high.
- Sensitive to outliers and missing values.
- Difficult to choose kernel,c & gamma.

SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.



for 2D data it is easy to draw hyperplane but where data point can't seperated by line need to convert into 3D where we can seperate the datapoint by hyperplane

Example :-

{Here Not easy to seperate by line}

$2D \rightarrow 3D$ and seperate by hyperplane

Hard Margin: In a hard margin SVM, the goal is to find the hyperplane that can perfectly separate the data into two classes without any misclassifications. However, this is not always possible when the data is not linearly separable or contains outliers. In such cases, the hard margin SVM will fail to find a hyperplane that can perfectly separate the data.

Hard margin SVM doesn't allow any misclassifications to happen.

Soft Margin: In a soft margin SVM, we allow some misclassification by allowing some data points to be on the wrong side of the margin.

Geometrically, the soft margin SVM introduces a penalty for the data points that lie on the wrong side of the margin or even on the wrong side of the hyperplane. The slack variables ξi allow these data points to be within the margin or on the wrong side of the hyperplane, but they incur a penalty in the objective function. The optimization problem in a soft margin SVM finds the hyperplane that maximizes the margin while minimizing the penalty for the misclassified data points.

C:

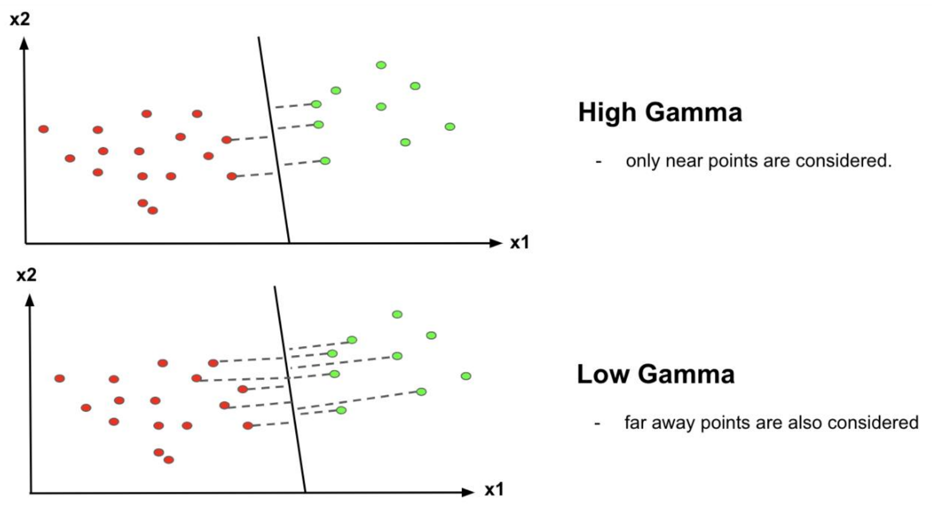
- ✓ Hard Margin has larger value of 'C' which results in boundary with small margin but no misclassifications.
- ✓ Soft Margin has smaller value of 'C' which results in boundary with large margin but with some misclassifications bound to happen.

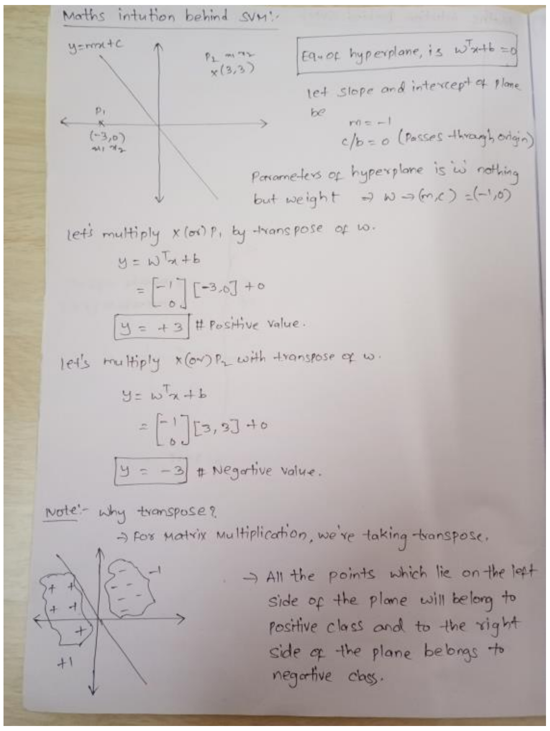
*** Large value of 'C' may lead to overfitting.*

Gamma:

Gamma is used when we use the Gaussian RBF kernel. if you use linear or polynomial kernel then you do not need gamma only you need C hypermeter. Gamma is a hyperparameter which we have to set before training model. Gamma decides that how much curvature we want in a decision boundary.

- ✓ High Gamma means more curvature.
- ✓ Low Gamma means less curvature.


x2
x1
High Gamma
- only near points are considered.
x2
x1
Low Gamma
- far away points are also considered



Maths intution behind SVM:-

$y=mx+c$
P_2 x_1 x_2
$x(3,3)$
P_1
$(-3,0)$
x_1 x_2

$\boxed{\text{Eq. of hyperplane, is } w^Tx+b=0}$

let slope and intercept of plane be

$m=-1$

$c/b=0$ (Passes through origin)

Parameters of hyperplane is 'w' nothing but weight $\Rightarrow w \to (m,c) = (-1,0)$

let's multiply x (or) P_1 by transpose of w.

$y = w^Tx+b$

$= \begin{bmatrix}-1\\0\end{bmatrix}[-3,0]+0$

$\boxed{y=+3}$ # Positive value.

let's multiply x (or) P_2 with transpose of w.

$y = w^Tx+b$

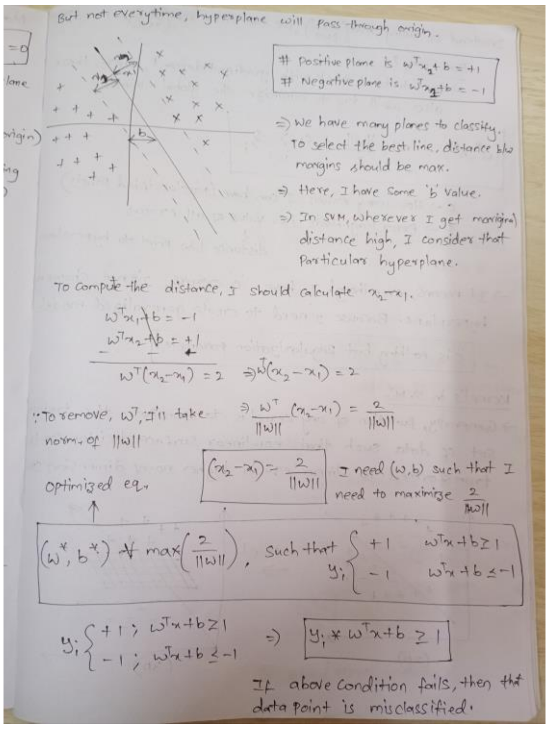
$= \begin{bmatrix}-1\\0\end{bmatrix}[3,3]+0$

$\boxed{y=-3}$ # Negative value.

Note:- why transpose?

→ For Matrix Multiplication, we're taking transpose.

→ All the points which lie on the left side of the plane will belong to positive class and to the right side of the plane belongs to negative class.



But not everytime, hyperplane will pass through origin.

$\boxed{\begin{array}{l}\text{\# Positive plane is } w^Tx_2+b=+1\\ \text{\# Negative plane is } w^Tx_1+b=-1\end{array}}$

⇒ We have many planes to classify. To select the best line, distance b/w margins should be max.

⇒ Here, I have some 'b' value.

⇒ In SVM, wherever I get marigin distance high, I consider that particular hyperplane.

To compute the distance, I should calculate x_2-x_1.

$w^Tx_1+b=-1$

$w^Tx_2+b=+1$

$w^T(x_2-x_1)=2 \Rightarrow w^T(x_2-x_1)=2$

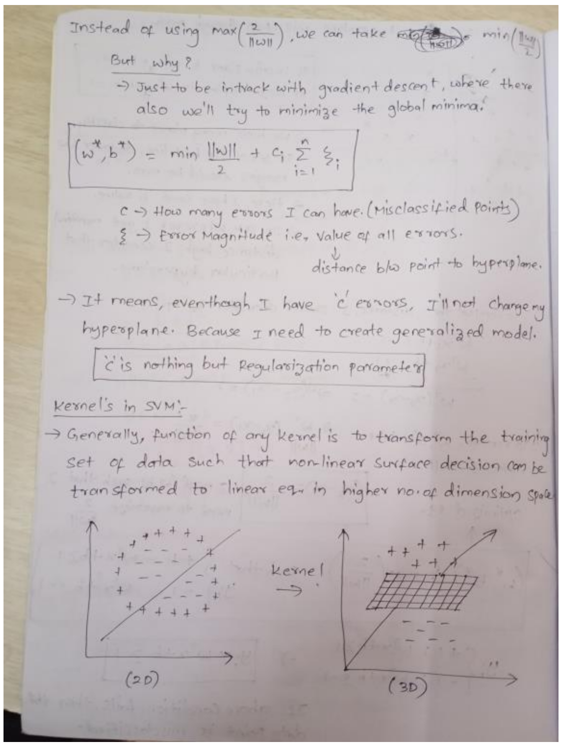
∵ To remove, w^T, I'll take norm. of $||w||$ $\Rightarrow \frac{w^T}{||w||}(x_2-x_1)=\frac{2}{||w||}$

optimized eq. $\boxed{(x_2-x_1)=\frac{2}{||w||}}$ I need (w,b) such that I need to maximize $\frac{2}{||w||}$

$$\boxed{(w^*,b^*)\ \forall\ \max\left(\frac{2}{||w||}\right),\quad \text{such that } y_i\begin{cases}+1 & w^Tx+b\geq 1\\ -1 & w^Tx+b\leq -1\end{cases}}$$

$y_i\begin{cases}+1; & w^Tx+b\geq 1\\ -1; & w^Tx+b\leq -1\end{cases} \Rightarrow \boxed{y_i * w^Tx+b \geq 1}$

If above condition fails, then that data point is misclassified.



Instead of using $\max\left(\frac{2}{||w||}\right)$, we can take ~~$\min\left(\frac{2}{||w||}\right)$~~ $\min\left(\frac{||w||}{2}\right)$

But why?

→ Just to be intrack with gradient descent, where there also we'll try to minimize the global minima.

$$\boxed{(w^*,b^*) = \min \frac{||w||}{2} + c_i \sum_{i=1}^{n} \xi_i}$$

c → How many errors I can have. (Misclassified Points)

ξ → Error Magnitude i.e. value of all errors.

↓

distance b/w point to hyperplane.

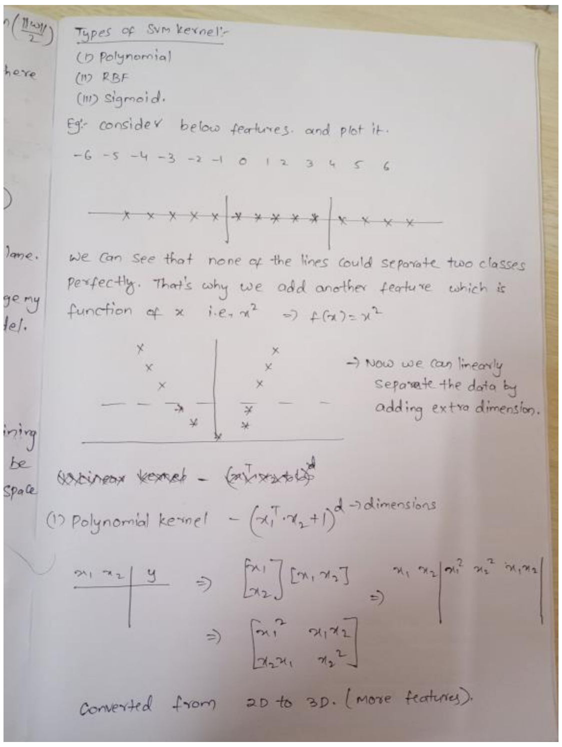
→ It means, eventhough I have 'c' errors, I'll not change my hyperplane. Because I need to create generalized model.

$\boxed{\text{'c' is nothing but Regularization parameter}}$

Kernel's in SVM:-

→ Generally, function of any kernel is to transform the training set of data such that non-linear surface decision can be transformed to linear eq. in higher no. of dimension space

kernel →
(2D)
(3D)



Types of SVM kernel:-

(I) Polynomial

(II) RBF

(III) Sigmoid.

Eg:- consider below features. and plot it.

-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6

We can see that none of the lines could separate two classes perfectly. That's why we add another feature which is function of x i.e. x^2 $\Rightarrow f(x)=x^2$

→ Now we can linearly separate the data by adding extra dimension.

~~Linear kernel~~ - [illegible]

(1) Polynomial kernel $- (x_1^T \cdot x_2+1)^d$ d → dimensions

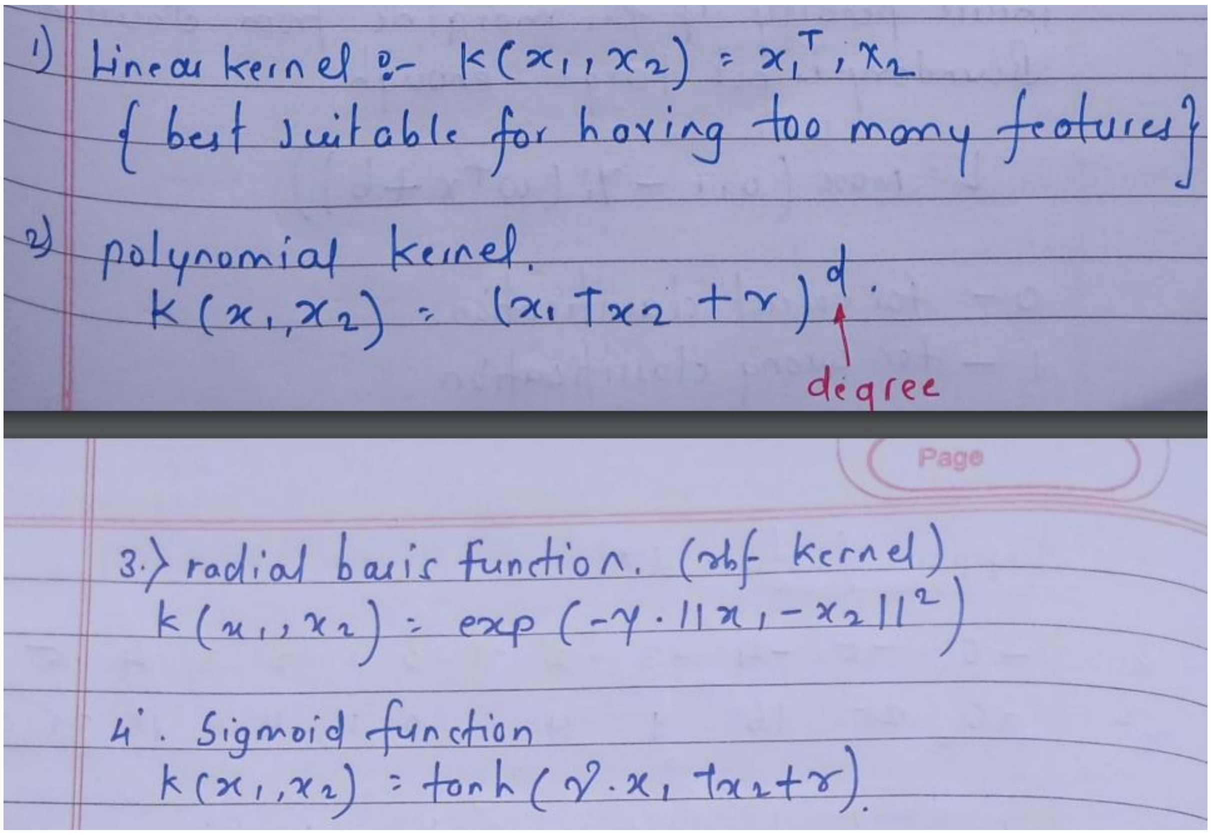
x_1	x_2	y

$\Rightarrow \begin{bmatrix}x_1\\x_2\end{bmatrix}[x_1, x_2] \Rightarrow \begin{bmatrix}x_1^2 & x_1x_2\\ x_2x_1 & x_2^2\end{bmatrix}$

$\Rightarrow$

x_1	x_2	x_1^2	x_2^2	x_1x_2

Converted from 2D to 3D. (More features).


1) Linear kernel :- $K(x_1, x_2) = x_1^T, x_2$

{best suitable for having too many features}

2) polynomial kernel.

$K(x_1, x_2) = (x_1^T x_2 + r)^{d}$.

degree

Page

3.) radial basis function. (rbf kernel)

$k(x_1, x_2) = \exp(-\gamma \cdot ||x_1 - x_2||^2)$

4. Sigmoid function

$k(x_1, x_2) = \tanh(\gamma \cdot x_1^T x_2 + r)$.

https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/

https://www.analyticsvidhya.com/blog/2021/06/support-vector-machine-better-understanding/

13. Explain about KNN in detail.

K Nearest Neighbor algorithm falls under Supervised Learning category and is used for both classification (most commonly) and regression problems. As the name (K Nearest Neighbor) suggests it considers K Nearest Neighbors (Data points) to predict the class or continuous value for the new Datapoint.

KNN is :

Instance-based / Memory-based learning: Here we do not learn weights from training data to predict output (as in model-based algorithms) but use entire training instances to predict output for unseen data.

Lazy Learning: Model is not learned using training data, which means it does not need any training data points for model generation. All training data used in the testing phase. Hence, it makes training faster and the testing phase slower and costlier.

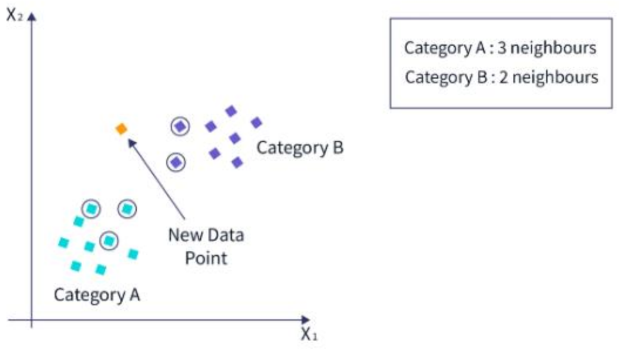
But Why? Because it does not create a generalized model during the time of training, so the testing phase is very important where it does the actual job. Hence testing is very costly - in terms of time & money.

Non -Parametric: KNN algo is non-parametric which means there is no assumption for underlying data distribution, there is no predefined form of the mapping function.

Though it is elementary to understand, it is a powerful technique for identifying the class of an unknown sample point.

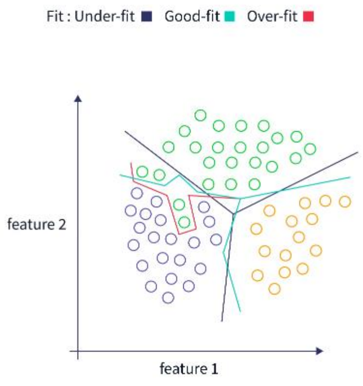
How does K-Nearest Neighbours Work?

- *Step 1* : Select the K number of neighbors.
- *Step 2* : Calculate the Euclidean distance of each point from the target point.
- *Step 3* : Take the K nearest neighbors as per the calculated Euclidean distance.
- *Step 4* : Among these k neighbors, count the number of the data points in each category.
- *Step 5* : Assign the new data points to that category for which the number of neighbors is max.



How to choose K Value ?

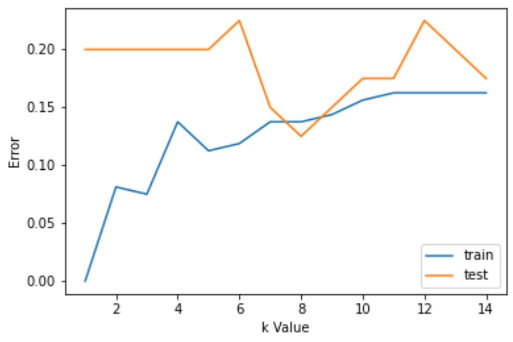
Choosing an appropriate value of k in KNN is crucial to get the best out of the model. If the value of K is small, the model's error rate will be large, especially for new data points, since the number of votes is small. Hence, the model is overfitted and highly sensitive to noise in the input. Moreover, the boundaries for classification become too rigid, as seen in the image below.



On the other hand, if the value of K is large, the model's boundaries become loose(smooth decision boundary), and the number of misclassifications increases. In this case, the model is said to be underfitted.

There is no fixed method to set K value, but consider following points.

- Set K as an odd number, so we have an extra point to settle tie-breakers in extreme cases.
- A decent value of k can be the square root of the number of data points involved in the training dataset. Of course, this method may not work in all cases.
- Plot the error rate vs K graph.

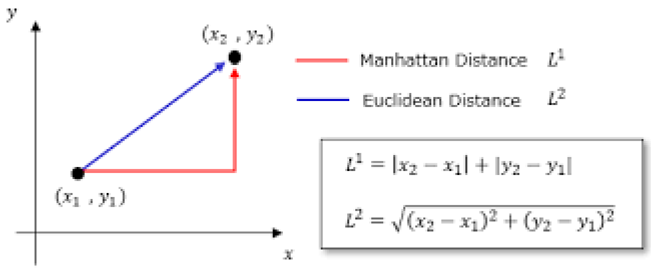


Start with k = 1 and predict with KNN for each data point in the testing data. Compare these predictions with the actual labels, and find the model's accuracy. Find the error rate. Repeat this process for different values of k and plot the error rate vs. k graph, and choose the value of k which has the least error rate. (Most common values are to be 3 & 5.)

Also, domain knowledge is very useful in choosing the K value.

Types of Distance metric:

- Euclidean distance
- Manhattan distance



y
x
(x_2, y_2)
(x_1, y_1)
Manhattan Distance L^1
Euclidean Distance L^2
$L^1 = |x_2 - x_1| + |y_2 - y_1|$
$L^2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

For categorical variables, we can use Hamming distance.

Advantages:

- KNN is extremely useful in scenarios that require high accuracy and do not contain many unknown points to classify. The concept of learning does not apply to KNN. Hence, if we need a quick prediction for a limited number of data points, then KNN is the go-to algorithm since other models, like neural networks and logistic regression, require time for training.
- Easy to implement.
- Simple hyperparameter tuning. (unlike in neural networks)

Disadvantages:

- KNN is sensitive to noise,outliers and missing values, so accurate data preprocessing must be done.
- Not applicable to huge datasets since distance would conumse lot of time.
- KNN employs instance-based learning and is a lazy algorithm that refers to an algorithm that does not learn anything from the dataset. Each time a new data point is supplied, it calculates the distance between that point and all other points in the dataset, which is computationally expensive. Essentially, the idea of training the KNN model does not exist because it makes a new decision by calculating the distances from a new point.

https://www.scaler.com/topics/machine-learning/knn-algorithm-in-machine-learning/

14. Explain about Naive Baye's in detail.

- Naive Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used mainly for solving classification problems.
- It is mainly used in text classification that includes a high-dimensional training dataset.
- Naive Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- Some popular examples of Naive Bayes Algorithm are spam filtration, sentimental analysis, and classifying articles.

For example, you cannot identify a bird based on its features and color as there are many birds with similar attributes. But, you make a probabilistic prediction about the same, and that is where the Naive Bayes Algorithm comes in.

Why is it called Naive Bayes?

Naive: It is called Naive because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the basis of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.

Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

Bayes' theorem finds the probability of an event occurring given the probability of another event that has already occurred. It depends on the conditional probability. The formula for Bayes' theorem is given as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

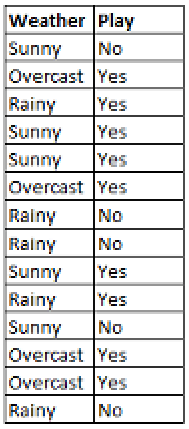
Where, **P(A)** is Prior Probability : Probability of hypothesis before observing the evidence.
P(B) is Marginal Probability : Probability of Evidence.
P(A|B) is Posterior probability : Probability of hypothesis A on the observed event B.
P(B|A) is Likelihood probability : Probability of the evidence given that the probability of a h ypothesis is true.

How Do Naive Bayes Algorithms Work?

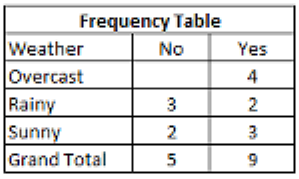
1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

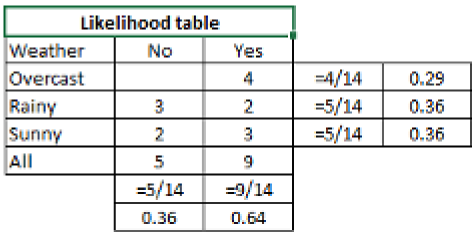
Solution: We can solve it using the above-discussed method of posterior probability.
Consider the below dataset:



Weather	**Play**
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No



Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9



Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Applying Baye's theorem, **P(Yes | Sunny) = P(Sunny | Yes) * P(Yes) / P (Sunny)**

Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P(Yes)= 9/14 = 0.64

Now, P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.

Types of Naive Bayes Model:

Gaussian: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.

Multinomial: The Multinomial Naive Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc. The classifier uses the frequency of words for the predictors.

Bernoulli: The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

Assumptions:

- We assume that no pair of features are dependent. For example the outlook being 'Rainy' does not affect other feature 'Windy'. Hence, the features are assumed to be independent.

- Secondly, each feature is given the same weight (or importance). For example, knowing the only temperature and humidity alone can't predict the outcome accurately. None of the attributes is irrelevant and assumed to be contributing equally to the outcome.

Advantages:

- Naive Bayes is one of the fast and easy ML algorithms which is straightforward to implement.
- It doesn't require larger amounts of training data.
- It can be used for Binary as well as Multi-class Classifications.
- It **performs well in Multi-class predictions** as compared to the other algorithms.
- It is the most popular choice for text classification problems.
- When assumption of independence holds, the classifier performs better compared to other machine learning models like logistic regression or decision tree, and requires less training data.

Disadvantages:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.
- If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency". To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.
- The Naive Bayes Algorithm has trouble with the 'zero-frequency problem'. It happens when you assign zero probability for categorical variables in the training dataset that is not available. When you use a smooth method for overcoming this problem, you can make it work the best.
- It will estimate things wrong sometimes, so you shouldn't take its probability outputs seriously.

Applications:

- Text classification /Sentiment analysis /Spam filtering
- Recommendation system
- Multi-class Prediction
- Medical diagnosis
- Weather prediction
- Face recognition

https://www.javatpoint.com/machine-learning-naive-bayes-classifier

https://www.turing.com/kb/an-introduction-to-naive-bayes-algorithm-for-beginners#probability,-bayes-theory,-and-conditional-probability

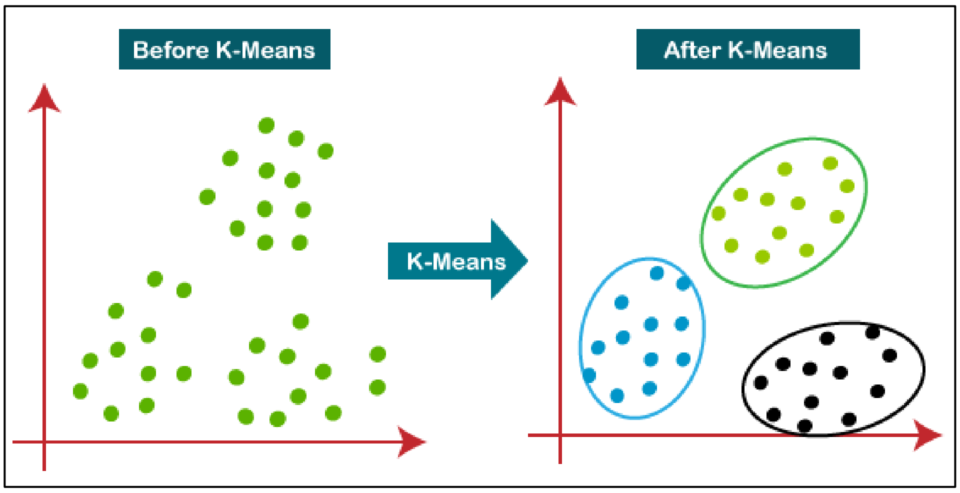
https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/#Sample_Project_to_Apply_Naive_Bayes

Unsuperviesed Learning Algo's Questions :

1. Explain K-Means Clustering in detail.

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabelled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties. It is a centroid-based algorithm, where each cluster is associated with a centroid.

Clustering is the process of division of the dataset into groups such that data points in the same group are more similar to the other points and dissimilar to the points which are in different group. It is basically the collection of objects on the basis of similarity and dissimilarity between them.



It is suggested to normalize the data while dealing with clustering algorithms such as K-Means since such algorithms employ distance-based measurement to identify the similarity between data points.

Because of the iterative nature of K-Means and the random initialization of centroids, K-Means may become stuck in a local optimum and fail to converge to the global optimum. As a result, it is advised to employ distinct centroids' initializations.

<u>How does the K-Means Algorithm Work?</u>

- *Step 1* : Select the value of K to decide the number of clusters.
- *Step 2* : Randomly initialize K centroids or points.
- *Step 3* : Compute the distance of every point to the centroid and a assign each data point to their closest centroid and cluster them.
- *Step 4* : Adjust the centroids in such a way they become center in the cluster.
- *Step 5* : Again recluster every point and adjust the centroids.
- *Step 6* : Repeat the above steps till there is no change in the clusters.

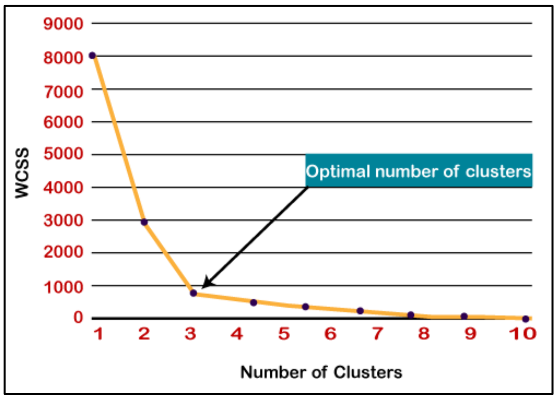
How to choose the best value of 'K' ?

We can select the K value using **Elbow method**. It is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. WCSS stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster.

$$\mathrm{WCSS} = \sum_{\mathrm{P_i\ in\ Cluster\ 1}} \mathrm{distance}(\mathrm{P_i}, \mathrm{C_1})^2 + \sum_{\mathrm{P_i\ in\ Cluster\ 2}} \mathrm{distance}(\mathrm{P_i}, \mathrm{C_2})^2 + \sum_{\mathrm{P_i\ in\ Cluster\ 3}} \mathrm{distance}(\mathrm{P_i}, \mathrm{C_3})^2$$

To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance. To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values. (from 1-10)
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp bend of the plot looks like an arm, that point is considered as the best value of K.



Advantages:

- Simple to use and K-Means is bit faster than Hierarchical clustering.

Disadvantages:

- It is bit difficult to estimate the value of 'K' and quite sensitive to rescaling.

Applications :

- Market segmentation & Customer segmentation.
- Document Clustering & Image segmentation.

https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning/#Introduction

2. Explain Hierarchical clustering in detail.

There are certain challenges with K-means. It always tries to make clusters of the same size. Also, we have to decide the number of clusters at the beginning of the algorithm. Ideally, we would not know how many clusters should we have, in the beginning of the algorithm and hence it a challenge with K-means. Herarchical Clustering takes away the problem of having to pre-define the number of clusters.

Hierarchical clustering is an unsupervised learning technique used to group similar objects into clusters. It creates a hierarchy of clusters by merging or splitting them based on similarity measures. Hierarchical clustering groups similar objects into a dendrogram. It merges similar clusters iteratively, starting with each data point as a separate cluster. This creates a tree-like structure that shows the relationships between clusters and their hierarchy.

Dendogram provides a visual representation of the relationships between clusters, helping to identify patterns and outliers, making it a useful tool for exploratory data analysis.

We are essentially building a hierarchy of clusters. That's why this algorithm is called hierarchical clustering.

There are two types of hierarchical clustering.

1. Agglomerative hierarchical clustering
2. Divisive Hierarchical clustering

1. Agglomerative hierarchical clustering:

- ✓ We assign each point to an individual cluster in this technique. Suppose there are 4 data points, we will assign each of these points to a cluster and hence will have 4 clusters in the beginning.
- ✓ Then, at each iteration, we merge the closest pair of clusters and repeat this step until only a single cluster is left.

2. Divisive hierarchical clustering:

- ✓ Divisive hierarchical clustering works in the opposite way. Instead of starting with n clusters , we start with a single cluster and assign all the points to that cluster.
- ✓ Now, at each iteration, we split the farthest point in the cluster and repeat this process until each cluster only contains a single point.

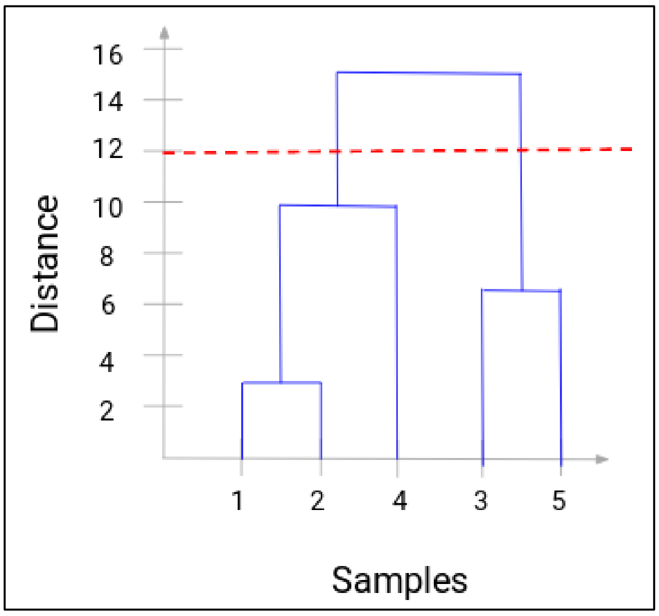
Now the question is - how do we decide which points are similar and which are not? We can use distance based algorithms for similar data points.

How to Choose the Number of Clusters?

To get the number of clusters for clustering, we make use of **Dendrogram**.

A dendrogram is a tree-like diagram that records the sequences of merges or splits.

We can visualize the steps of hierarchical clustering. More the distance of the vertical lines in the dendrogram, more the distance between those clusters.



Now, we can set a threshold distance and draw a horizontal line (*Generally, we try to set the threshold so that it cuts the tallest vertical line*). Let's set this threshold as 12 and draw a horizontal line.

The number of clusters will be the number of vertical lines intersected by the line drawn using the threshold. In the above example, since the red line intersects 2 vertical lines, we will have 2 clusters. One cluster will have a sample (1,2,4) and the other will have a sample (3,5).

This is how we can decide the number of clusters using a dendrogram in Hierarchical Clustering.

https://www.javatpoint.com/hierarchical-clustering-in-machine-learning

https://www.analyticsvidhya.com/blog/2019/05/beginners-guide-hierarchical-clustering/#h-supervised-vs-unsupervised-learning

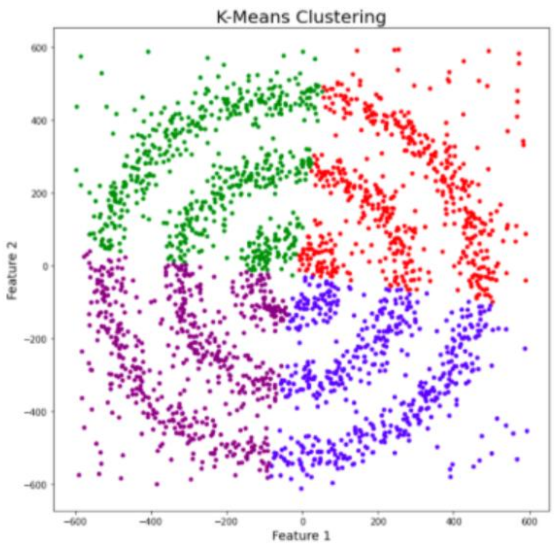
https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec

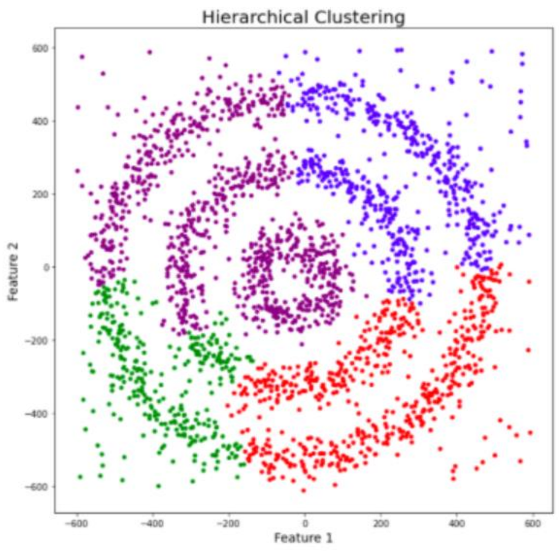
3. How does DBSCAN clustering works?

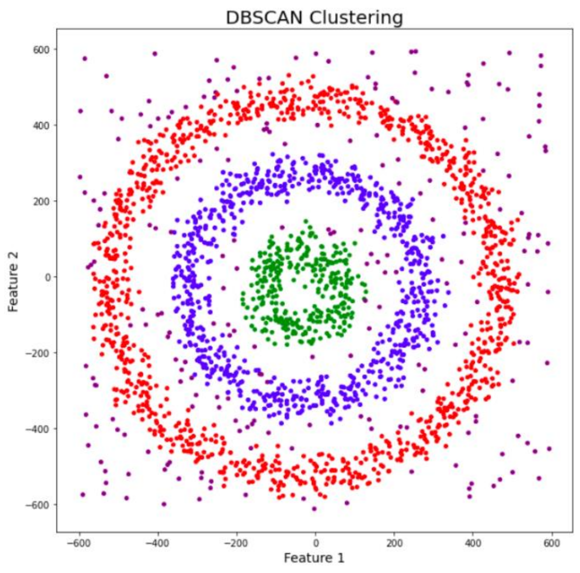
K-Means and Hierarchical Clustering both fail in creating clusters of arbitrary shapes. They are not able to form clusters based on varying densities. That’s why we need DBSCAN clustering.

DBSCAN stands for **D**ensity **B**ased **S**patial **C**lustering of **A**pplications with **N**oise.

The most exciting feature of DBSCAN clustering is that it is robust to outliers.







There are two different parameters to calculate the density-based clustering.

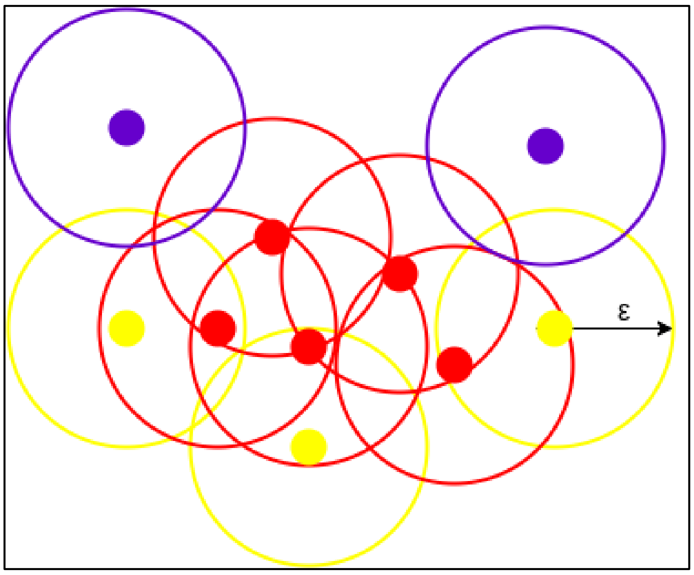
1. Eps(ε) : Epsilon is the radius of the circle to be created around each data point to check the density. It is the distance that specifies the neighborhood. Two points are considered to be neighbors iff distance between them is less than or equal to eps.

2. Min pts : Minimum number of data points to defina a cluster.

Core Point : A point is a core point if there is are atleast **minpts** number of points (including the point itself) in its surrounding area with radius epsilon (ε).

Border Point : A point is a border point if it is reachable from core point and if the number of points are less than minPoints within its surrounding area, then it is classified as Border Point.

Noise/ Outlier : If there are no other data points around any data point within *epsilon* radius, then it treated as Noise.



The above figure shows us a cluster created by DBCAN with *minPoints = 3*.

All the data points with at least 3 points in the circle including itself are considered as **Core** points represented by red color. All the data points with less than 3 but greater than 1 point in the circle including itself are considered as **Border** points. They are represented by yellow color. Finally, data points with no point other than itself present inside the circle are considered as **Noise** represented by the purple color.

For locating data points in space, DBSCAN uses Euclidean distance, although other methods can also be used (like great circle distance for geographical data).

The value of *minPoints* should be at least one greater than the number of dimensions of the dataset.

$$\boxed{minPoints >= Dimensions+1}$$

Advantages:

- ✓ Doesn't require to specify the number of clusters before hand.
- ✓ Performs well with arbitrary shaped data.
- ✓ Robust to outliers and able to detect outliers.

Disadvantages:

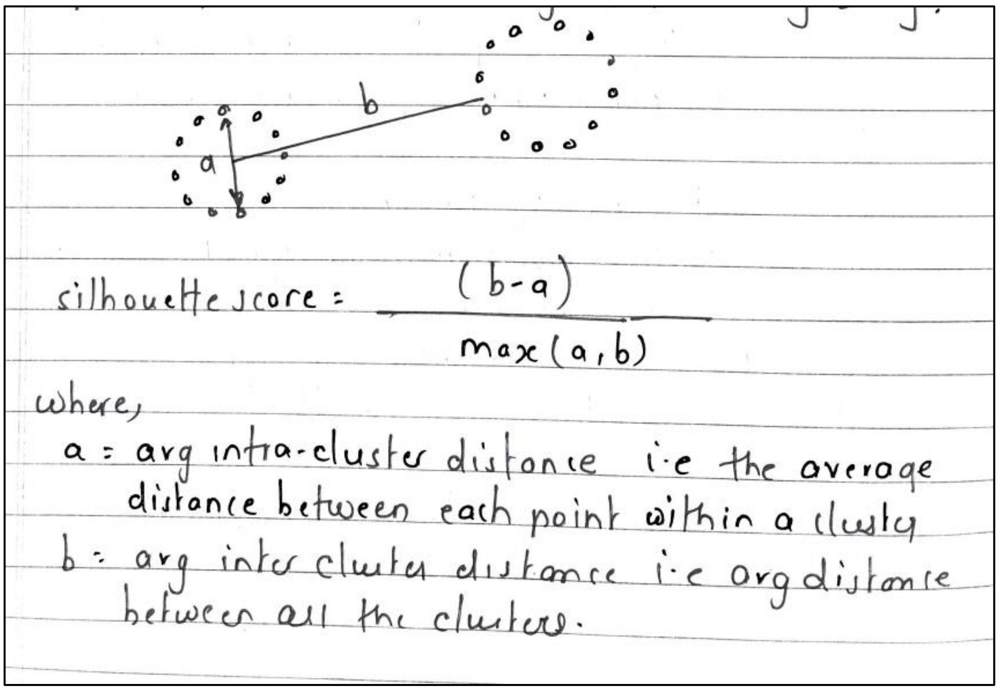
- ✓ Determining eps is not easy and it requires domain knowledge.

Cluster validation technique:

Elbow method can be used only for K-Means. Silhouette score can be used everywhere. Silhouette score is a metric used to calculate the goodness of clustering technique. It's value ranges from -1 to +1.

Silhouette score can also be used to check optimal number of clusters.

- ✓ -1 : Clusters are well apart from each other and clearly distinguished.
- ✓ 0 : Distance between clusters is not significant.
- ✓ +1 : Clusters are assigned in wrong way.



silhouette score = $\dfrac{(b-a)}{\max(a,b)}$

where,
a = avg intra-cluster distance i.e the average distance between each point within a cluster
b = avg inter cluster distance i.e avg distance between all the clusters.

https://www.analyticsvidhya.com/blog/2020/09/how-dbscan-clustering-works/#:~:text=DBSCAN%20is%20a%20density%2Dbased,points%20into%20a%20single%20cluster

https://towardsdatascience.com/dbscan-clustering-explained-97556a2ad556

Boosting Algo’s Questions :

1. Explain Adaboost in detail.

Boosting grants power to machine learning models to improve their accuracy of prediction. Boosting algorithms are one of the most widely used algorithm in data science competitions.

What is Boosting?

The term ‘Boosting’ refers to a family of algorithms which converts weak learner to strong learners.

Let’s understand this with small eg by solving a problem of spam email identification.

How would you classify an email as **SPAM or not**? Like everyone else, our initial approach would be to identify ‘spam’ and ‘not spam’ emails using following criteria.

1. Email has only one image file (promotional image), it’s a SPAM.
2. Email has only link(s), it’s a SPAM.
3. Email body consist of sentence like “You won a prize money of $ xxxxxx, it’s a SPAM.
4. Email from our official domain “Analyticsvidhya.com” , Not a SPAM.
5. Email from known source, Not a SPAM.

Above, we’ve defined multiple rules to classify an email into **spam** or **not spam**. But, do you think these rules individually are strong enough to successfully classify an email? No. Individually, these rules are not powerful enough to classify an email into **spam** or **not spam**. Therefore, these rules are called as **weak learner**.

To convert weak learner to strong learner, we’ll combine the prediction of each weak learner using methods like:

1. Using average/ weighted average
2. Considering prediction has higher vote

For example, Above, we have defined 5 weak learners. Out of these 5, 3 are voted as **SPAM** and 2 are voted as **Not a SPAM**. In this case, by default, we’ll consider an email as SPAM because we have higher(3) votes for ‘SPAM’.

Bagging	Boosting
1.Each model is built independently.	1.New models will get effected by the implementation of previous models
2.Several training datasets are randomly drawn.	2.Each new subset includes the components that were misclassified by previous models.
3.Can solve overfitting.	3.Can boost overfitting.
4.Decreases the variance.	4.Decreases the bias.

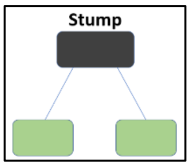
How Boosting Algorithm works?

We know that boosting combines weak learners to form strong learner. How boosting identifies weak learners ?

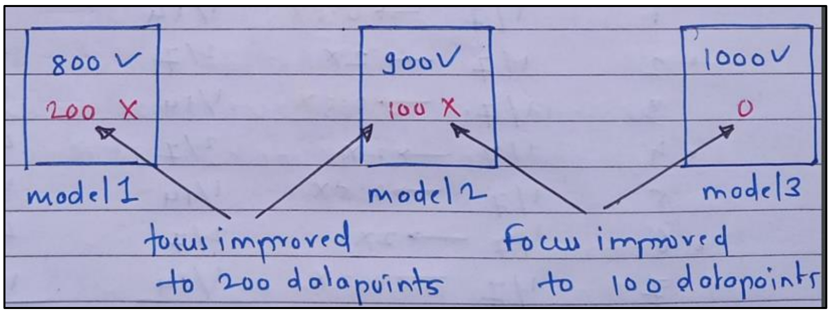
To find weak learners, we apply base learning (ML) algorithms with a different distribution. Each time base learning algorithm is applied, it generates a new weak prediction rule. This is an iterative process. After many iterations, the boosting algorithm combines these weak rules into a single strong prediction rule.

Let's see how boosting works with example and here we will discuss about Adaboosting.

Adaboost, also known as 'Adaptive Boosting' is a technique in machine learning used as an ensemble method. The most common estimator used in Adaboost is Decision trees with one level which means Decision tree with only one split. These trees are also called as 'STUMPS'.



Boosting is nothing but wherever my model is performing bad, I need to improve focus on there and I shoud give more priority over to those points.


800 ✓
200 X
model 1
900 ✓
100 X
model 2
1000 ✓
0
model 3
focus improved to 200 datapoints
Focus improved to 100 datapoints

Consider below table which is the actual representation of dataset and let's preidct the output.

Feature1	**Feature2**	**Feature3**	**Output**	**Weight**
1	40	50000	Yes	1/7
2	10	20000	No	1/7
3	20	45000	No	1/7
4	50	30000	Yes	1/7
5	30	15000	No	1/7
6	60	32000	Yes	1/7
7	45	28000	Yes	1/7

Step 01 : Initially assign sample weights to all the data points. (1/N)
Step 02 : Create a decision stump for the features. The tree with lowest entropy is our first stump.
Step 03 : Above stump is a weak learner and it is one level decision tree.

Step 04 : Now, we need to calculate the total error for misclassified points. The total error is nothing but the summation of all the sample weights of misclassified data points. (4th record is misclassified)

Total error will always be between 0 and 1. 0 indicates 'perfect stump' and 1 indicates 'horrible stump'.

Step 05 : Now calculate the performance of the stump using below formula.

$$\boxed{Performance\ of\ the\ stump\ =\ \frac{1}{2}\log_e\left(\frac{1-Total\ Error}{Total\ Error}\right)}$$

Why we are calculating the total error & performance ? To update the weights and change the focus. The wrong predictions will be given more weight, whereas the correct predictions will be given less weights. Now when we build our next model after updating the weights, more preference will be given to the points with higher weights.

Step 06 : Using below formula, update the weights and normalize the weights. (Because, after updating summation of the weights may greater than 1).

$$\boxed{New\ sample\ weight\ =\ old\ weight\ *\ e^{\pm Amount\ of\ say\ (\alpha)}}$$

If the points are correctly classified, alpha is negative.
If the points are incorrectly classified, alpha is positive.

Now our dataset looks like this.

Feature1	**Feature2**	**Feature3**	**Output**	**Old weights**	**Normalized weights**	**Buckets**
1	40	50000	Yes	1/7	0.07	0-0.07
2	10	20000	No	1/7	0.07	0.07-0.14
3	20	45000	No	1/7	0.07	0.14-0.21
4	50	30000	Yes	1/7	0.537	0.21-0.747
5	30	15000	No	1/7	0.07	0.747-0.817
6	60	32000	Yes	1/7	0.07	0.817-0.887
7	45	28000	Yes	1/7	0.07	0.887-0.957

Step 07 : We are almost done. Our algorithm selects random number between 0 and 1. Since incorrectly classified records have higher sample weights, the probability of selecting those records is very high.

Suppose random numbers are 0.09, 0.24,0.39,0,53,0.71, 0.94.

Feature1	**Feature2**	**Feature3**	**Output**	**Old weights**	**Normalized weights**	**Buckets**
2	10	20000	No	1/7	0.07	0.07-0.14
4	50	30000	Yes	1/7	0.537	0.21-0.747
7	45	28000	Yes	1/7	0.07	0.887-0.957

This comes out to be our new dataset, and we see the data point, which was wrongly classified, has been selected 3 times because it has a higher weight. We need to repeat all the above steps.

- Assign equal weights to all the data points.
- Find the stump that does the best job classifying the new collection of samples by finding their Gini Index / Entropy and selecting the one with the lowest Gini index.
- Calculate the "Total error" & "Performance" to update the previous sample weights.
- Normalize the new sample weights.

Iterate through these steps until and unless a low training error is achieved.

Suppose, with respect to our dataset, we have constructed 3 decision trees (DT1, DT2, DT3) in a sequential manner. If we send our test data now, it will pass through all the decision trees, and finally, we will see which class has the majority, and based on that, we will do predictions for our test dataset.

Advantages :

- Less preprocessing is required.
- Need not to scale the features.
- Less prone to overfitting.

https://www.analyticsvidhya.com/blog/2021/09/adaboost-algorithm-a-complete-guide-for-beginners/

https://www.youtube.com/watch?v=NLRO1-jp5F8

2. Explain Gradient boosting in detail.

The main idea behind this algorithm is to build models sequentially and these subsequent models try to reduce the errors of the previous model. But how do we do that? How do we reduce the error? This is done by building a new model on the errors or residuals of the previous model.

When the target column is continuous, we use **Gradient Boosting Regressor** whereas when it is a classification problem, we use **Gradient Boosting Classifier**. The only difference between the two is the “Loss function”. The objective here is to minimize this loss function by adding weak learners using gradient descent. Since it is based on loss function hence for regression problems, we’ll have different loss functions like Mean squared error (MSE) and for classification, we will have different for e.g log-likelihood.

Let’s understand Gradient Boosting with the below example.

Consider the below dataset where we have to predict the salary of candidate. Target column is salary.

Experience	**Degree**	**Salary**
2	B.Tech	50,000
3	Masters	70,000
5	Masters	80,000
6	Phd	1,00,000

Step 01 : First step in gradient boosting is to create one base model to predict the observations. In base model, predicted value is always the average of all target variables.

Experience	**Degree**	**Salary**	**Pred1(ŷ)**
2	B.Tech	50,000	75,000
3	Masters	70,000	75,000
5	Masters	80,000	75,000
6	Phd	1,00,000	75,000

Why we should take the average of the target column? Well, there is math involved behind this. Mathematically the first step can be written as:

$$\boxed{F_0(x) = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L(y_i, \gamma).}$$

‘L’ is loss function and gamma is predicted value. argmin means we have to find the predicted value (gamma) such that loss function is minimum. Since it is Regression problem, we are using MSE.

$$\boxed{L = \frac{1}{n} \sum_{i=0}^{n} \left(y_i - \gamma_i\right)^2}$$

$$\boxed{\frac{\mathrm{dL}}{\mathrm{d}\gamma} = \frac{2}{2}\left(\sum_{i=0}^{n} \left(y_i - \gamma_i\right)\right) = -\sum_{i=0}^{n} \left(y_i - \gamma_i\right)}$$

Step 02 : Next step is to calculate the pseudo residuals by taking the difference. (actual-predicted)

Experience	Degree	Salary	Pred1(ŷ)	Residual (R1)
2	B.Tech	50,000	75,000	-25,000
3	Masters	70,000	75,000	-5,000
5	Masters	80,000	75,000	5,000
6	Phd	1,00,000	75,000	25,000

Again the question comes, why we take only difference ? It is also mathematically proved as,

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

Here, F(xi) is the previous model and m is number of DT's made. If you take the derivative of the above equation, it gives you the difference between actual value and predicted value.

In the next step, we will build a model on these pseudo residuals and make predictions. Why do we do this? Because we want to minimize these residuals and minimizing the residuals will eventually improve our model accuracy and prediction power. So, using the Residual as target and the original features we will generate new predictions. Note that the predictions, in this case, will be the error values, not the predicted salary values since our target column is an error now.

Step 03 : Let's say $h_m(x)$ is our DT made on these residuals and find the output values on our DT.

Experience	Degree	Salary	Pred1(ŷ)	Residuals (R1)	Pred2 (R2)
2	B.Tech	50,000	75,000	-25,000	-23,000
3	Masters	70,000	75,000	-5,000	-3,000
5	Masters	80,000	75,000	5,000	3,000
6	Phd	1,00,000	75,000	25,000	20,000

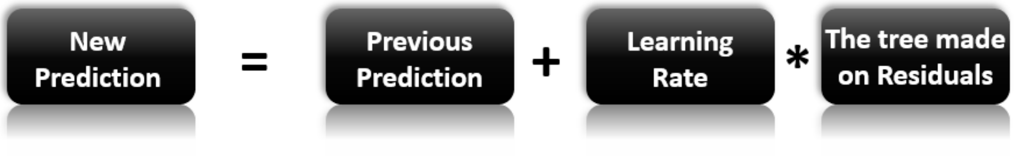
$$\gamma_m = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

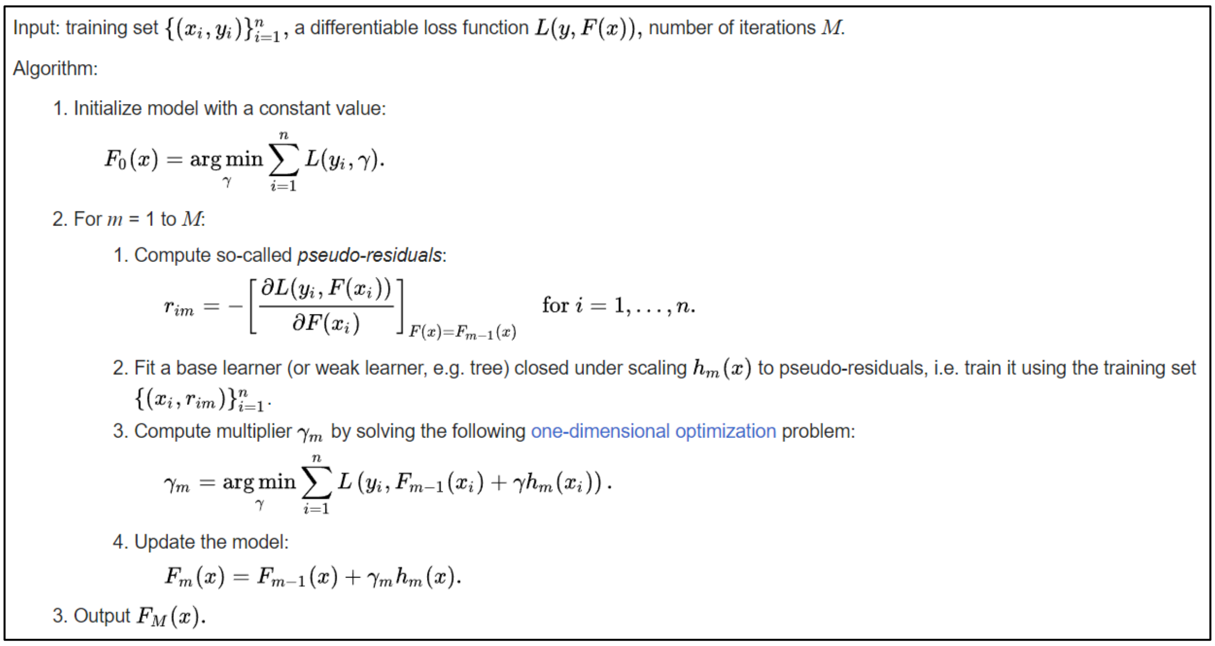
Step 04 : This is the last step where we have to update the predictions of the previous model.

Update the model:

$$F_m(x) = F_{m-1}(x) + \nu_m h_m(x)$$

$F_{m-1}(x)$ is the prediction of the base model and $h_m(x)$ is the recent DT made on the residuals.





```
Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations $M$.
Algorithm:
    1. Initialize model with a constant value:
           $F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$
    2. For $m$ = 1 to $M$:
        1. Compute so-called pseudo-residuals:
               $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$   for $i = 1, \ldots, n.$
        2. Fit a base learner (or weak learner, e.g. tree) closed under scaling $h_m(x)$ to pseudo-residuals, i.e. train it using the training set
           $\{(x_i, r_{im})\}_{i=1}^{n}$.
        3. Compute multiplier $\gamma_m$ by solving the following one-dimensional optimization problem:
               $\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$
        4. Update the model:
               $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$
    3. Output $F_M(x)$.
```

What is Gradient Boosting Classifier?

A gradient boosting classifier is used when the target column is binary. All the steps explained in the Gradient boosting regressor are used here, the only difference is we change the loss function. Earlier we used Mean squared error when the target column was continuous but this time, we will use log-loss as our loss function.

$$\boxed{L = -\sum_{i=1}^{n} y_i \log(p) + (1-p)\log(1-p)}$$

https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/

https://www.youtube.com/watch?v=Nol1hVtLOSg&pp=ygUUZ3JhZGJvb3N0IGtyaXNoIG5haWs%3D

https://www.youtube.com/watch?v=Oo9q6YtGzvc&pp=ygUUZ3JhZGJvb3N0IGtyaXNoIG5haWs%3D

3. Explain XGBoost in detail.

Researchers thought that ensembling trees randomly was time-consuming and computationally inefficient. Why not build trees sequentially and improve over those parts where previous trees failed. That's where boosting came into the picture. Later these boosting algorithms started utilizing the gradient descent algorithm to form trees sequentially and minimize the error in predictions. Hence these algorithms are called Gradient Boosting. Later, researchers proposed model, algorithmic, and hardware optimizations to further improve the Gradient boosting algorithms' performance.

How XGBoost works?

Salary	**Credit Score**	**Approval**
<= 50K	B	0
<= 50K	G	1
<= 50K	G	1
> 50K	B	0
> 50K	G	1
> 50K	N	1
<= 50K	N	0

Step 01 : Construct the tree with the root. Initially, let's take probability as 0.5, which is the average value of the target.

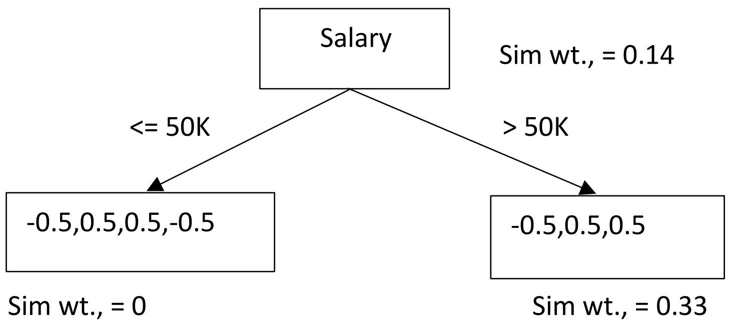
Step 02 : Using the below formula, calculate similarity weights for every branch. Assume λ = 0.

$$\boxed{\frac{(\Sigma(\text{Residual}))^2}{\Sigma(\Pr(1-\Pr)) + \lambda}}$$

Step 03 : Calculate the Info gain for all the features and select the one which is having more gain.

Let's say in our case, salary has the highest gain and take it as root.

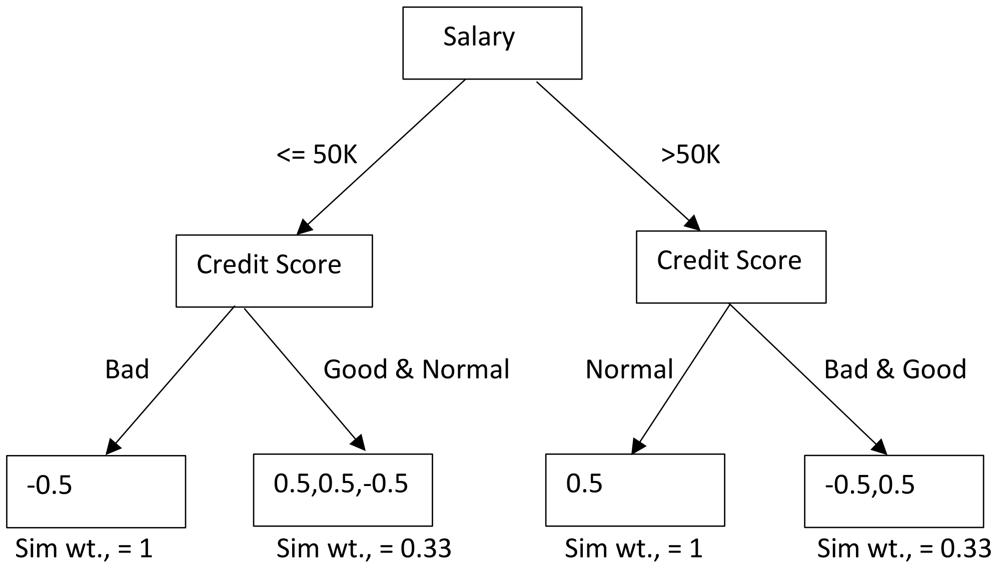
[-0.5,0.5,0.5,-0.5,0.5,0.5,-0.5]



Gain = Sim wt., of left tree + Sim wt., of right tree - Sim wt., of root

Gain (Salary) = 0+0.33-0.14 ⇒ 0.21 and Gain (Credit) = 0.03

Below is our first level decision tree which is built on salary as root.



Once our tree got built, we can do post pruning just to avoid overfitting. Post pruning can be calculate by 'Cover Value' . *If my Gain is less than cover value, then I will cut that branch.*

$$\boxed{\text{Cover Value} = Pr(1-Pr)}$$

For new data, we will use sigmoid function and we will get the new probabilities for next tree.

$Pr(<=50K \mid B) = \sigma\,(0 + 0.1 * 1) \Rightarrow 0.6$
$Pr(<=50K \mid G) \Rightarrow 0.4$

$$\boxed{\text{New Probabilities} = \sigma[BM + \alpha\ DT1 + \alpha\ DT2 + \ldots + \alpha Tn]}$$

Where, '**α**' is learning rate which is the hyperparameter.

Salary	**Credit Score**	**Approval**	**Residuals**	**New Probability**	**New Residuals**
<= 50K	B	0	-0.5	+0.6	-0.6
<= 50K	G	1	+0.5	+0.4	+0.6
<= 50K	G	1	+0.5	+0.3	+0.7
> 50K	B	0	-0.5	+0.7	-0.7
> 50K	G	1	+0.5	+0.2	+0.8
> 50K	N	1	+0.5	+0.1	+0.9
<= 50K	N	0	-0.5	+0.3	-0.3

$$\begin{tikzcd}\boxed{\text{Base Model}} \arrow[r] & \boxed{\text{DT1}} \arrow[r] & \boxed{\text{DT2}} \arrow[r] & {}\end{tikzcd}$$

https://www.youtube.com/watch?v=w-_vmVfpssg
https://www.youtube.com/watch?v=gPciUPwWJQQ