Linear Regression:

The linear relationship between a response (or dependent) variable (Y) and one or more explanatory (independent) variables (X).

We can express it in the form of the following equation:

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

In the case of a single explanatory variable, it is called simple linear regression, and if there is more than one explanatory variable, it is multiple linear regression.

Usually, the OLS (Ordinary Least Squares) method is used to estimate the regression coefficients. OLS finds the best coefficients by minimizing the sum of the squares of the errors.

Assumptions of Linear Regression:

Assumptions about the explanatory variables (features):

- **Linearity**: the linearity is only with respect to the parameters. Oddly enough, there's no such restriction on the degree or form of the explanatory variables themselves.

  So both the following equations represent linear regression:

  $$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

  Here, the model is linear in parameters as well as linear in the explanatory variable(s).

  $$Y_i = \beta_0 + \beta_1 X_i^2 + \epsilon_i$$

  The following equation is NOT linear regression:

  $$Y_i = \beta_0 + \beta_1^2 X_i + \epsilon_i$$

- **No multicollinearity:** If there is a linear relationship between one or more explanatory variables, it adds to the complexity of the model without being able to delineate the impact of each explanatory variable on the response variable.

  Now, the reason multicollinearity is problematic is that, a linear regression problem transfers $Y = XB$

  where X is the design matrix, B is the coefficient vector of each of the variables. Now, in general, the solution of this system is

  $$B = (X^T X)^{-1} XY$$

  Now, based on this matrix $X^T X$,we find point estimates and the standard errors of the betas i.e. the variables and therefore, the reliability of the linear regression depends on the invertibility of the above matrix.

For example If we were to model the salaries of a group of professionals based on their ages and years of experience.

$$salary_i = \beta_0 + \beta_1(years\ of\ experience)_i + \beta_2(age\ in\ years)_i + \epsilon_i\,(salary)_i$$

Linear regression studies the effect of each of the independent variables (X) on the dependent variable (Y). But when the independent variables are correlated, as in this case, it is difficult to isolate the impact of a single factor on the dependent variable. If you increase the $years\ of\ experience$, the age also will increase.

So did the salary increase due to the experience or the age? This will affect the accuracy of the coefficients and also the standard errors.

How to detect multicollinearity?

- Check the correlation among the independent variables.
- Variance Inflation Factor

How to fix multicollinearity?

- One way to deal with multicollinearity among the independent variables is to do dimensionality reduction using techniques like PCA to create uncorrelated features with the maximum variance.
- In a practical approach, you will consider the correlated variables and look at their individual prediction power. Then, you will remove the variables which are weaker in predicting and are involved in creating the multicollinearity.

- **Everything about Multicollinearity:**

https://towardsdatascience.com/everything-you-need-to-know-about-multicollinearity-2f21f082d6dc

- **Homoskedasticity**: Unequal variance in the error terms is called heteroskedasticity and may be caused due to many reasons like the presence of outliers, or an incorrectly specified model. If we run the regression with heteroskedasticity present, the standard errors would be large and the model would have unreliable predictions.

  Detect homoskedasticity

  - The residual plot can give a good idea of the variance of the residuals. Homoskedastic residuals show no pattern, whereas heteroskedastic residuals show a systematic increasing or decreasing variation

  How to fix

  - As discussed earlier, one way to deal with heteroskedasticity is to transform the equation. Taking a log of the dependent variable usually gets rid of the heteroskedasticity.
  - Using White's errors (also known as Robust errors) that correct the standard errors in the OLS for heteroskedasticity.

- Use the weighted least squares model, which is an efficient variation of OLS to deal with heteroskedasticity.

- **No autocorrelation:** Autocorrelation or serial correlation is a problem specific to regression involving time-series data. When working with time-series data, when the value at a particular time t is dependent on its value at a time ($t-1$), there is a chance of correlation between the residuals after the model fitting step. When this assumption is violated, although the model is still unbiased, its efficiency will be impacted. The standard errors would increase, and the goodness of fit may be exaggerated. The significant regression coefficients may appear to be statistically significant when they are not.
  detect autocorrelation
  - A snake-like wavy pattern showing sort-of clustering of the residuals in the residual plot indicates autocorrelation of the error terms.
  - Durbin-Watson test
  - Ljung Box Test
  - Moran's I statistic

  How to fix it
  - Check for any other parameters influencing the dependent variable and include them in the linear regression model. After including this new XX term, we can check if the residual plot evens out.
  - Cochrane-Orchutt procedure
  - AR(1) method
- **Zero conditional mean**:
- **Assumptions about the error terms (residuals): Gaussian distribution:** If this assumption is violated, it is not a big problem, especially if we have a large number of observations. This is because the central limit theorem will apply if we have a large number of observations, implying that the sampling distribution will resemble normality irrespective of the parent distribution for large sample sizes.
  However, if the number of observations is small and the normality assumption is violated, the standard errors in your model's output will be unreliable.
  How to detect normality
  In the residual plot, most points should be centered around zero and should get sparse as we move further away from the best-fit line. The residual means should be zero.
    - Q-Q plot
    - Kolmogorov-Smirnov Test
    - Anderson-Darling Test
    - Shapiro-Wilk Test
    - Ryan-Joiner Test

  How to fix normality

The first thing to check is whether we have any outliers. As mentioned earlier, the linear regression model uses the 'least squares' to come up with the best fit. So any outliers will severely impact the residuals.

- Add more observations, if possible.
- We can try modifying the functional form of the regression equation by applying nonlinear transformations, as mentioned above.

Limitations of linear regression:

- **Simplicity in some cases:** For a lot of real-world applications, especially when dealing with time-series data, it does not fit the bill. For non-linear relationships (when you see a curve in your residual plot), using logistic regression would be a better option. An underlying assumption of the linear regression model for time-series data is that the underlying series is stationary. However, this does not hold true for most economic series in their original form are non-stationary.
- **Sensitivity to outliers:** outliers can have an outsize impact on the output of the model.
- **Prone to underfitting:** it cannot capture the complexity of most real-life applications. The accuracy of this model is low in such cases.
- **Overfitting of complex model:** it does not perform well on the test data. In linear regression, this usually happens when the model is too complex with many parameters, and the data is too less.

R-Squared:

SSR (Sum of Squares of Residuals) is the sum of the squares of the difference between the actual observed value (y) and the predicted value ($\boldsymbol{y_{pred}}$) i.e. $\boldsymbol{SSR = \Sigma(y - y_{pred})^2}$

SST (Total Sum of Squares) is the sum of the squares of the difference between the actual observed value (y) and the average of the observed y value ($\boldsymbol{y_{avg}}$) i.e. $\boldsymbol{SST = \Sigma(y - y_{avg})^2}$

$$\boldsymbol{R^2 = 1 - \frac{SSR}{SST}}$$

Adjusted R-Squared:

$\boldsymbol{R^2}$ is R-Squared value

p is the number of predictor

N is the total sample size

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

What Is the Difference Between R-Squared and Adjusted R-Squared?

The most vital difference between adjusted R-squared and R-squared is simply that adjusted R-squared considers and tests different independent variables against the model and R-squared does not.

Key Differences

The most obvious difference between adjusted R-squared and R-squared is simply that adjusted R-squared considers and tests different independent variables against the stock index and R-squared does not. Because of this, many investment professionals prefer using adjusted R-squared because it has the potential to be more accurate. Furthermore, investors can gain additional information about what is affecting a stock by testing various independent variables using the adjusted R-squared model.

R-squared, on the other hand, does have its limitations. One of the most essential limits to using this model is that R-squared cannot be used to determine whether or not the coefficient estimates and predictions are biased. Furthermore, in multiple linear regression, the R-squared can not tell us which regression variable is more important than the other.

Which Is Better, R-Squared or Adjusted R-Squared?

Many investors prefer adjusted R-squared because adjusted R-squared can provide a more precise view of the correlation by also taking into account how many independent variables are added to a particular model against which the stock index is measured.

Should I Use Adjusted R-Squared or R-Squared?

Using adjusted R-squared over R-squared may be favored because of its ability to make a more accurate view of the correlation between one variable and another. Adjusted R-squared does this by taking into account how many independent variables are added to a particular model against which the stock index is measured.

What Is an Acceptable R-Squared Value?

Many people believe there is a magic number when it comes to determining an R-squared value that marks the sign of a valid study however this is not so. Because some data sets are inherently set up to have more unexpected variations than others, obtaining a high R-squared value is not always realistic. However, in certain cases an R-squared value between 70-90% is ideal.

How to Calculate Correlation Between Categorical Variables

Often we use the Pearson Correlation Coefficient to calculate the correlation between continuous numerical variables. There are three metrics that are commonly used to calculate the correlation between categorical variables:

1. Tetrachoric Correlation: Tetrachoric correlation is used to calculate the correlation between binary categorical variables. Recall that binary variables are variables that can only take on one of two possible values.

The value for tetrachoric correlation ranges from -1 to 1 where -1 indicates a strong negative correlation, 0 indicates no correlation, and 1 indicates a strong positive correlation.

2. Polychoric Correlation: Polychoric correlation is used to calculate the correlation between ordinal categorical variables. Recall that ordinal variables are variables whose possible values have a natural order.

The value for polychoric correlation ranges from -1 to 1 where -1 indicates a strong negative correlation, 0 indicates no correlation, and 1 indicates a strong positive correlation.

For example, suppose want to know whether or not two different movie ratings agencies have a high correlation between their movie ratings.

We ask each agency to rate 20 different movies on a scale of 1 to 3 with 1 indicating “bad”, 2 indicating “mediocre”, and 3 indicating “good.”

3. Cramer’s V: Cramer’s V is used to calculate the correlation between nominal categorical variables. Recall that nominal variables are ones that take on category labels but have no natural ordering.

The value for Cramer’s V ranges from 0 to 1, with 0 indicating no association between the variables and 1 indicating a strong association between the variables.

Logistic Regression:

Logistic regression is a supervised machine learning algorithm that accomplishes binary classification tasks by predicting the probability of an outcome, event, or observation. Examples: Determine the probability of heart attacks, Possibility of enrolling into a university Identifying spam emails

Assumptions of Logistic Regression

- **The Response is Binary**
- **The Observations are Independent**
- **No Multicollinearity Among Explanatory Variables:** If the degree of correlation is high enough between variables, it can cause problems when fitting and interpreting the model.

  How to check this assumption: The most common way to detect multicollinearity is by using the variance inflation factor (VIF), which measures the correlation and strength of correlation between the predictor variables in a regression model.
- **No Extreme Outliers**
- **Linear Relationship Between Explanatory Variables and the Logit of the Response Variable**
- **Sample Size is Sufficiently Large**

Logistic Regression vs. Linear Regression (Assumptions):

In contrast to linear regression, logistic regression does not require:

- A linear relationship between the explanatory variable(s) and the response variable.
- The residuals of the model to be normally distributed.
- The residuals to have constant variance, also known as homoscedasticity.

Advantages:

- easier to implement, interpret, and very efficient to train.
- makes no assumptions about distributions of classes in feature space.
- easily extend to multiple classes (multinomial regression) and a natural probabilistic view of class predictions.
- not only provides a measure of how appropriate a predictor(coefficient size)is, but also its direction of association (positive or negative)
- very fast at classifying unknown records.
- Good accuracy for many simple data sets and it performs well when the dataset is linearly separable.
- can interpret model coefficients as indicators of feature importance
- less inclined to over-fitting but it can overfit in high dimensional datasets. One may consider Regularization (L1 and L2) techniques to avoid over-fitting in these scenarios.

Disadvantages:

- If the number of observations is lesser than the number of features, Logistic Regression should not be used, otherwise, it may lead to overfitting.
- It constructs linear boundaries.

- The major limitation of Logistic Regression is the assumption of linearity between the dependent variable and the independent variables.
- can only be used to predict discrete functions. Hence, the dependent variable of Logistic Regression is bound to the discrete number set
- Non-linear problems can't be solved with logistic regression because it has a linear decision surface. Linearly separable data is rarely found in real-world scenarios

Support Vector Machine or SVM:

Support vector machines are a supervised learning method used to perform binary classification on data. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme points/vectors are called as support vectors.

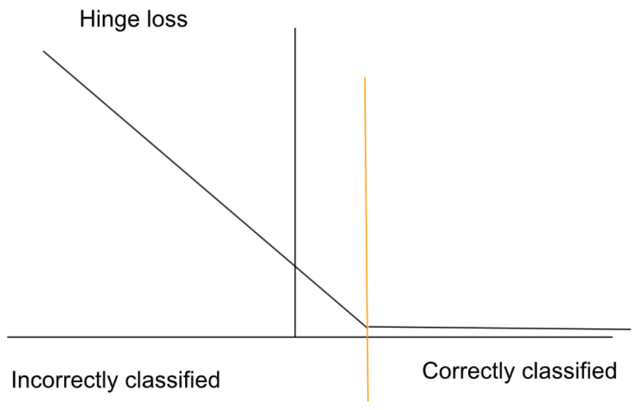
Hinge Loss:

The hinge loss is a loss function used for training classifiers, most notably the SVM. The x-axis represents the distance from the boundary of any single instance, and the y-axis represents the loss size, or penalty, that the function will incur depending on its distance.

$$l = max(0, 1 - t.y) \quad t \in \{1, -1\} \text{ is the label}$$

y is the output of the classifier. For SVM $y = w^T x + b$

$$\begin{aligned} L &= 0 \qquad && if\ y.(w.x) \geq 1, \\ &= 1 - y.(w.x) && otherwise \end{aligned}$$



Types of SVM:

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

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

Kernel — a mathematical function used to transform input data into a different form. Common kernel functions include linear, nonlinear, polynomial, etc.

Assumptions of SVM:

- The margin should be as large as possible.
- The support vectors are the most useful data points because they are the ones most likely to be incorrectly classified.

Advantages of SVM:

- SVM works relatively well when there is a clear margin of separation between classes.
- SVM is more effective in high dimensional spaces.
- SVM is effective in cases where the number of dimensions is greater than the number of samples.
- SVM is relatively memory efficient

Disadvantages of SVM:

- SVM algorithm is not suitable for large data sets.
- SVM does not perform very well when the data set has more noise i.e. target classes are overlapping.
- In cases where the number of features for each data point exceeds the number of training data samples, the SVM will underperform.
- As the support vector classifier works by putting data points, above and below the classifying hyperplane there is no probabilistic explanation for the classification.

K means clustering:

K means algorithm is an iterative algorithm that tries to partition the dataset into *K* pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to **only one group**. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster. The approach kmeans follows to solve the problem is called **Expectation-Maximization**. The E-step is assigning the data points to the closest cluster. The M-step is computing the centroid of each cluster.

Applications

kmeans algorithm is very popular and used in a variety of applications such as market segmentation, document clustering, image segmentation and image compression, etc. The goal usually when we undergo a cluster analysis is either:

- Get a meaningful intuition of the structure of the data we're dealing with.
- Cluster-then-predict where different models will be built for different subgroups if we believe there is a wide variation in the behaviors of different subgroups. An example of that is clustering patients into different subgroups and build a model for each subgroup to predict the probability of the risk of having heart attack.

Elbow Method

Elbow method gives us an idea on what a good *k* number of clusters would be based on the sum of squared distance (SSE) between data points and their assigned clusters' centroids. Inertia is the sum of squared distance of samples to their closest cluster center. We would like this number to be as small as possible. But, if we choose K that is equal to the number of samples we will get inertia=0. This is the smallest inertia value we can achieve, however we miss our goal of clustering the data into the optimal number of clusters.
The value of inertia decreases as the number of clusters increases - so we will need to manually pick K while considering the trade-off between the inertia value and the number of clusters. For that, we usually use the Elbow Method- and we choose the elbow point in the inertia graph. After that point the improvement in the inertia value is not significant.

Silhouette Analysis

Silhouette analysis can be used to determine the degree of separation between clusters. For each sample:

- Compute the average distance from all data points in the same cluster (ai).

- Compute the average distance from all data points in the closest cluster (bi).
- Compute the coefficient:

$$\frac{b^i - a^i}{max(a^i, b^i)}$$

The coefficient can take values in the interval [-1, 1].

- If it is 0 –> the sample is very close to the neighbouring clusters.
- If it is 1 –> the sample is far away from the neighbouring clusters.
- If it is -1 –> the sample is assigned to the wrong clusters.

Therefore, we want the coefficients to be as big as possible and close to 1 to have a good cluster.

Assumptions of K means clustering:

- **Limited to spherical shaped clusters**: As K-means calculates distance from centroid, it forms a spherical shape. Thus, it cannot cluster complicated geometrical shape.
  Solution — KERNEL method
  transform to higher dimensional representation which makes data linearly separable.
- **Size of clusters:** This algorithm considers only distance, thus does not account for clusters with different sizes or densities. It assumes that features within a cluster have equal variance.

Advantages of K means clustering:

- Relatively simple to implement.
- Scales to large data sets.
- Guarantees convergence.
- Can warm-start the positions of centroids.
- Easily adapts to new examples.
- Generalizes to clusters of different shapes and sizes, such as elliptical clusters.

Disadvantages of K means clustering:

- **Choosing** k **manually.**
- **Being dependent on initial values:** For a low k, you can mitigate this dependence by running k-means several times with different initial values

and picking the best result. As k increases, you need advanced versions of k-means to pick better values of the initial centroids (called **k-means seeding**).

- **Clustering data of varying sizes and density:** k-means has trouble clustering data where clusters are of varying sizes and density. To cluster such data, you need to generalize k-means
- **Clustering outliers:** Centroids can be dragged by outliers, or outliers might get their own cluster instead of being ignored. Consider removing or clipping outliers before clustering.
- **Scaling with number of dimensions:** As the number of dimensions increases, a distance-based similarity measure converges to a constant value between any given examples. Reduce dimensionality either by using **PCA** on the feature data, or by using “spectral clustering” to modify the clustering algorithm.

KNN:

The K-Nearest Neighbors (KNN) algorithm is one of the simplest supervised machine learning algorithms that is used to solve both classification and regression problems.
KNN is also known as an instance-based model or a lazy learner because it doesn’t construct an internal model.
For classification problems, it will find the k nearest neighbors and predict the class by the majority vote of the nearest neighbors.
For regression problems, it will find the k nearest neighbors and predict the value by calculating the mean value of the nearest neighbors.

___________Alternative____________

The KNN algorithm classifies unclassified data points based on their proximity and similarity to other available data points. The underlying assumption this algorithm makes is that similar data points can be found near one another. It’s commonly used to solve problems in various industries because of its ease of use, application to classification and regression problems, and the ease of interpretability of the results it generates.

How to find the optimum k value?
The choice of K is crucial for the model, if chosen incorrectly it can cause the model to be over / underfit. A K value too small will cause noise in the data to have a high influence on the prediction, however, a K value too large will make it computationally expensive.
The industry standard for choosing the optimal value of K is by taking the square root of N, where N is the total number of samples. Of course, take this with a grain of salt as it varies from problem to problem.
Generally, k gets decided based on the square root of the number of data points.

Always use k as an odd number. Since KNN predicts the class based on the majority voting mechanism, the chances of getting into a tie situation will be minimized.
We can also use an error plot or accuracy plot to find the most favorable K value. Plot possible k values against error and find the k with minimum error and that k value is chosen as the favorable k value.

Use Euclidean Distance.

Algorithm:
KNN algorithm calculates the distance of all data points from the query points using techniques like euclidean distance.
Then, it will select the k nearest neighbors.
Then based on the majority voting mechanism, KNN algorithm will predict the class of the query point.

Why KNN is known as an instance-based method or Lazy learner:
KNN algorithm is known as an instance-based method or lazy learner because it doesn't explicitly learn a model. It doesn't learn a discriminative function from the training data. It just memorizes the training instances which are used as "knowledge" for the prediction phase.

Assumptions of KNN:

- The data is in feature space, which means data in feature space can be measured by distance metrics such as Manhattan, Euclidean etc.
- Each of the training data points consists of a set of vectors and class label associated with each vector.

Advantages of KNN:

- Simple & intuitive — The algorithm is very easy to understand and implement
- Memory based approach — Allows it to immediately adapt to new training data
- Variety of distance metrics — There is flexibility from the users side to use a distance metric which is best suited for their application (Euclidean, Minkowski, Manhattan distance etc.)

Disadvantages of KNN:

- Computational complexity — As your training data increases, the speed at which calculations are made rapidly decrease
- Poor performance on imbalanced data — When majority of the data the model is being trained on represents 1 label then that label will have a high likelihood of being predicted
- Optimal value of K — If chosen incorrectly, the model will be under or over fitted to the data

- Feature Scaling- Data in all the dimensions should be scaled (normalized and standardized) properly .

Naive Bayes:

Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

Assumptions of Naive Bayes:

The fundamental Naive Bayes assumption is that each feature makes an

- independent
- equal

Advantages of Naive Bayes:

- Simple to Implement. The conditional probabilities are easy to evaluate.
- Very fast – no iterations since the probabilities can be directly computed. So, this technique is useful where speed of training is important.
- If the conditional Independence assumption holds, it could give great results.

Disadvantages of Naive Bayes:

- Conditional Independence Assumption does not always hold. In most situations, the feature shows some form of dependency.
- **Zero probability problem:** When we encounter words in the test data for a particular class that are not present in the training data, we might end up with zero class probabilities as $P(y|x_1, x_2, \dots, x_n) = \frac{P(x_1|y)P(x_2|y)\dots P(x_n|y)}{P(x_1)P(x_2)\dots P(x_n)}$
-

Decision Trees:

Decision Tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

Gini Index:

Gini Index is a score that evaluates how accurate a split is among the classified groups. Gini index evaluates a score in the range between 0 and 1, where 0 is when all observations belong to one class, and 1 is a random distribution of the elements within classes. In this case, we want to have a Gini index score as low as possible. Gini Index is the evaluation metrics we shall use to evaluate our Decision Tree Model.
Gini Impurity tells us what is the probability of misclassifying an observation.

$$gini = 1 - \sum_j p_j^2 \cdot$$

Where p_j is class j probability.

Entropy:
Entropy is a measure of information that indicates the disorder of the features with the target. Similar to the Gini Index, the optimum split is chosen by the feature with less entropy. It gets its maximum value when the probability of the two classes is the same and a node is pure when the entropy has its minimum value, which is 0:

$$entropy = - \sum_j plogp$$

Which is Better?
Computationally, entropy is more complex since it makes use of logarithms and consequently, the calculation of the Gini Index will be faster

How to use a decision tree for regression?
The most common method for constructing a regression tree is the CART (Classification and Regression Tree) methodology, which is also known as recursive partitioning.
The method starts by searching for every distinct value of all its predictors, and splitting the value of a predictor that minimizes the following statistic (other regression tree models have different optimization criteria):

$$RSS = \sum_{i \in S_1} (yi - y1)^2 + \sum_{i \in S_2} (yi - y2)^2$$

y_1 and y_2 are averages of dependent variables of class S1 and S2.

Overfitting:
Overfitting refers to a model that learns the training data (the data it uses to learn) so well that it has problems generalizing to new (unseen) data.
In other words, the model learns the detail and noise (irrelevant information or randomness in a dataset) in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model.
Under this condition, your model works perfectly well with the data you provide upfront, but when you expose that same model to new data, it breaks down. It's unable to repeat its highly detailed performance.

How to handle overfitting in DT?
You need to exclude branches that fit data too specifically. You want a DT that can generalize and work well on new data, even though this may imply losing precision on the training data.

Pruning:

Pruning is a technique used to deal with overfitting that reduces the size of DTs by removing sections of the Tree that provide little predictive or classification power.
The goal of this procedure is to reduce complexity and gain better accuracy by reducing the effects of overfitting and removing sections of the DT that may be based on noisy or erroneous data.

Decision Tree Algorithm:

CART:

CART is a DT algorithm that produces binary Classification or Regression Trees, depending on whether the dependent (or target) variable is categorical or numeric, respectively. It handles data in its raw form (no preprocessing needed), and can use the same variables more than once in different parts of the same DT, which may uncover complex interdependencies between sets of variables.
In the case of **Classification Trees**, CART algorithm uses a metric called Gini Impurity to create decision points for classification tasks. Gini Impurity gives an idea of how fine a split is (a measure of a node's "purity"), by how mixed the classes are in the two groups created by the split.
In the case of **Regression Trees**, CART algorithm looks for splits that minimize the Least Square Deviation (LSD), choosing the partitions that minimize the result over all possible options. The LSD (sometimes referred as "variance reduction") metric minimizes the sum of the squared distances (or deviations) between the observed values and the predicted values. The difference between the predicted and observed values is called "residual", which means that LSD chooses the parameter estimates so that the sum of the squared residuals is minimized
The idea behind CART algorithm is to produce a sequence of DTs, each of which is a candidate to be the "optimal Tree". This optimal Tree is identified by evaluating the performance of every Tree through testing (using new data, which the DT has never seen before) or performing cross-validation.

ID:

The Iterative Dichotomiser 3 (ID3) is a DT algorithm that is mainly used to produce Classification Trees.
ID3 splits data attributes (dichotomizes) to find the most dominant features, performing this process iteratively to select the DT nodes in a top-down approach.
For the splitting process, ID3 uses the Information Gain metric to select the most useful attributes for classification. Information Gain is a concept extracted from Information Theory, that refers to the decrease in the level of randomness in a set of data: basically, it measures how much "information" a feature gives us about a class. ID3 will always try to maximize this metric, which means that the attribute with the highest Information Gain will split first.
But ID3 has some disadvantages: it can't handle numeric attributes nor missing values, which can represent serious limitations.

Assumptions of Decision Trees:

The below are the some of the assumptions we make while using Decision tree:

- At the beginning, the whole training set is considered as the **root.**
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- Records are **distributed recursively** on the basis of attribute values.
- Order to placing attributes as root or internal node of the tree is done by using some statistical approach

Advantages of Decision Trees:

- Decision trees are able to generate understandable rules.
- Decision trees perform classification without requiring much computation.
- Decision trees are able to handle both continuous and categorical variables.
- Decision trees provide a clear indication of which fields are most important for prediction or classification.

Disadvantages of Decision Trees:

- Surely DTs have lots of advantages. Because of their simplicity and the fact that they are easy to understand and implement, they are widely used for different solutions in a large number of industries.
- They also suffer from **high variance**, which means that a small change in the data can result in a very different set of splits, making interpretation somewhat complex.
- They suffer from an **inherent instability** since due to their hierarchical nature, the effect of an error in the top splits propagates down to all of the splits below.
- DTs can also create biased Trees if some classes dominate over others. This is a problem in unbalanced datasets (where different classes in the dataset have different number of observations), in which case it is recommended to balance de dataset prior to building the DT.
- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
- Decision trees are prone to errors in classification problems with many classes and a relatively small number of training examples.
- Decision tree can be computationally expensive to train. The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found. In some algorithms, combinations of fields are used and a search must be made for optimal combining weights. Pruning algorithms can also be expensive since many candidate sub-trees must be formed and compared.

Random Forest:

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model’s prediction.

The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. **The reason for this wonderful effect is that the trees protect each other from their individual errors** (as long as they don’t constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

Assumptions of Random Forest:

- There needs to be some actual signal in our features so that models built using those features do better than random guessing.
- The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

Advantages of Random Forest:

- Robust to outliers.
- Works well for non-linear data.
- Low risk of overfitting.
- Runs efficiently on large datasets.

Disadvantages of Random Forest :

- Slow training.
- Biased when dealing with categorical variables.

Bagging:

Advantages of Bagging:

- increases classifier stability and accuracy;
- reduces classifier variance, in terms of the bias-variance decomposition

The use of the bagging technique improves the classification results whenever the base classifiers are unstable, this being the main reasons why the bagging approach works well for classification.

Boosting:

Gradient Boosting:

What is: Gradient boosting is one of the variants of ensemble methods where you create multiple weak models and combine them to get better performance as a whole.

XGBoost:

XgBoost stands for Extreme Gradient Boosting. XGBoost is an implementation of Gradient Boosted decision trees. XGBoost models majorly dominate in many Kaggle Competitions.
In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.

Assumptions of XGBoost:

- It may have an assumption that encoded integer value for each variable has ordinal relation.

Advantages of XGBoost:

- Can work parallelly.
- Can handle missing values.
- Auto prouning
- No need for scaling or normalizing data.
- Fast to interpret.
- Great execution speed.
- Cache optimization

Disadvantages of XGBoost:

- Can easily overfit if parameters are not tuned properly.

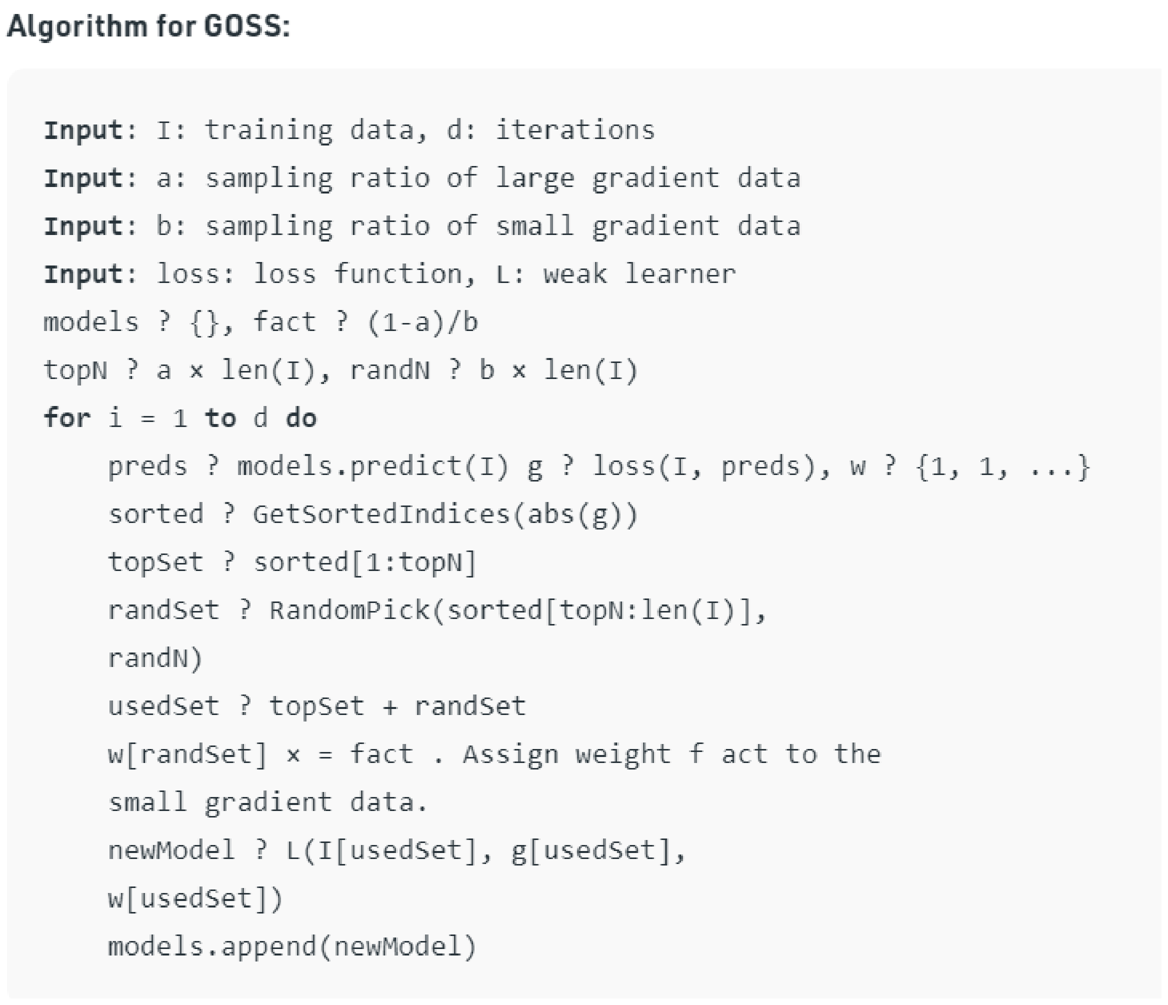
- Hard to tune.

LightGBM (Light Gradient Boosting Machine):

LightGBM is a gradient boosting framework based on decision trees to increases the efficiency of the model and reduces memory usage. It uses two novel techniques: **Gradient-based One Side Sampling** and **Exclusive Feature Bundling (EFB)** which fulfills the limitations of histogram-based algorithm that is primarily used in all GBDT (Gradient Boosting Decision Tree) frameworks. The two techniques of GOSS and EFB described below form the characteristics of LightGBM Algorithm. They comprise together to make the model work efficiently and provide it a cutting edge over other GBDT frameworks

Gradient-based One Side Sampling Technique for LightGBM:

Different data instances have varied roles in the computation of information gain. The instances with larger gradients(i.e., under-trained instances) will contribute more to the information gain. GOSS keeps those instances with large gradients (e.g., larger than a predefined threshold, or among the top percentiles), and only randomly drop those instances with small gradients to retain the accuracy of information gain estimation. This treatment can lead to a more accurate gain estimation than uniformly random sampling, with the same target sampling rate, especially when the value of information gain has a large range.



Algorithm for GOSS:

```
Input: I: training data, d: iterations
Input: a: sampling ratio of large gradient data
Input: b: sampling ratio of small gradient data
Input: loss: loss function, L: weak learner
models ? {}, fact ? (1-a)/b
topN ? a × len(I), randN ? b × len(I)
for i = 1 to d do
    preds ? models.predict(I) g ? loss(I, preds), w ? {1, 1, ...}
    sorted ? GetSortedIndices(abs(g))
    topSet ? sorted[1:topN]
    randSet ? RandomPick(sorted[topN:len(I)],
    randN)
    usedSet ? topSet + randSet
    w[randSet] × = fact . Assign weight f act to the
    small gradient data.
    newModel ? L(I[usedSet], g[usedSet],
    w[usedSet])
    models.append(newModel)
```

Mathematical Analysis for GOSS Technique (Calculation of Variance Gain at splitting feature j)
For a training set with n instances $\{x_1, \cdot\cdot\cdot, x_n\}$, where each x_i is a vector with dimension *s* in space X^s. In each iteration of gradient boosting, the negative gradients of the loss function with respect to the output of the model are denoted as $\{g_1, \cdot\cdot\cdot, g_n\}$. In this GOSS method, the training instances are ranked according to their absolute values of their gradients in the descending order. Then, the top-a × 100% instances with the larger gradients are kept and we get an instance subset A. Then, for the remaining set A^c consisting (1- a) × 100% instances with smaller gradients., we further randomly sample a subset B with size b × $|A^c|$. Finally, we split the instances according to the estimated variance gain at vector V_j(d) over the subset A ? B

$$\tilde{V}_j(d) = \frac{1}{n}\left(\frac{\left(\sum_{x_i \in A_l} g_i + \frac{1-a}{b}\sum_{x_i \in B_l} g_i\right)^2}{n_l^j(d)} + \frac{\left(\sum_{x_i \in A_r} g_i + \frac{1-a}{b}\sum_{x_i \in B_r} g_i\right)^2}{n_r^j(d)}\right) \qquad (1)$$

where $A_l = \{x_i ? A : x_{ij} ? d\}$, $A_r = \{x_i ? A : x_{ij} > d\}$, $B_l = \{x_i ? B : x_{ij} ? d\}$, $B_r = \{x_i ? B : x_{ij} > d\}$, and the coefficient *(1-a)/b* is used to normalize the sum of the gradients over B back to the size of A^c.

Exclusive Feature Bundling Technique for LightGBM: High-dimensional data are usually very sparse which provides us a possibility of designing a nearly lossless approach to reduce the number of features. Specifically, in a sparse feature space, many features are mutually exclusive, i.e., they never take nonzero values simultaneously. The exclusive features can be safely bundled into a single feature (called an Exclusive Feature Bundle). Hence, the complexity of histogram building changes from *O(#data × #feature)* to *O(#data × #bundle)*, while *#bundle<<#feature* . Hence, the speed for training framework is improved without hurting accuracy.

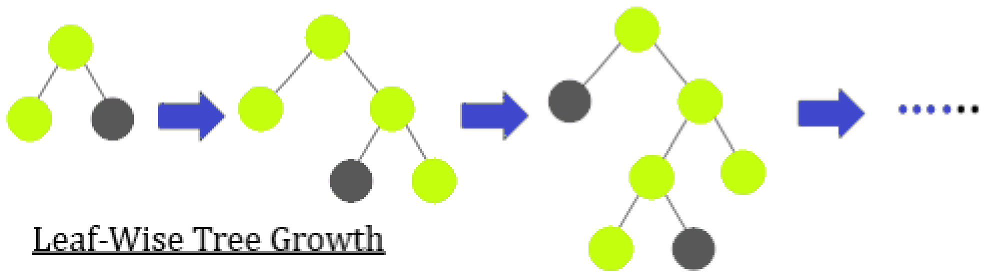
Algorithm for Exclusive Feature Bundling Technique:

```
Input: numData: number of data
Input: F: One bundle of exclusive features
binRanges ? {0}, totalBin ? 0
for f in F do
    totalBin += f.numBin
    binRanges.append(totalBin)
newBin ? new Bin(numData)
for i = 1 to numData do
    newBin[i] ? 0
    for j = 1 to len(F) do

        if F[j].bin[i] != 0 then
            newBin[i] ? F[j].bin[i] + binRanges[j]
Output: newBin, binRanges
```

Architecture:

LightGBM splits the tree leaf-wise as opposed to other boosting algorithms that grow tree level-wise. It chooses the leaf with maximum delta loss to grow. Since the leaf is fixed, the leaf-wise algorithm has lower loss compared to the level-wise algorithm. Leaf-wise tree growth might increase the complexity of the model and may lead to overfitting in small datasets. Below is a diagrammatic representation of Leaf-Wise Tree Growth:



Parameter Tuning

Few important parameters and their usage is listed below :

1. **max_depth :** It sets a limit on the depth of tree. The default value is 20. It is effective in controlling over fitting.
2. **categorical_feature :** It specifies the categorical feature used for training model.
3. **bagging_fraction :** It specifies the fraction of data to be considered for each iteration.
4. **num_iterations :** It specifies the number of iterations to be performed. The default value is 100.
5. **num_leaves :** It specifies the number of leaves in a tree. It should be smaller than the square of *max_depth*.
6. **max_bin :** It specifies the maximum number of bins to bucket the feature values.
7. **min_data_in_bin :** It specifies minimum amount of data in one bin.
8. **task :** It specifies the task we wish to perform which is either train or prediction. The default entry is *train*. Another possible value for this parameter is *prediction.*
9. **feature_fraction** : It specifies the fraction of features to be considered in each iteration. The default value is one.