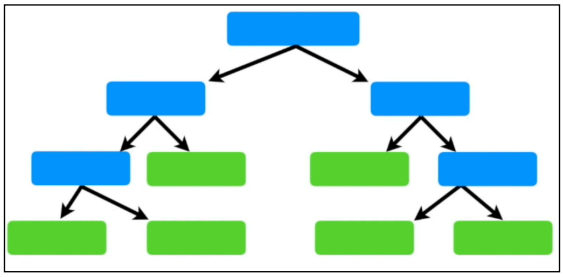
TREE STRUCTURED MACHINE LEARNING

DECISION TREES

1. A Decision Tree is a supervised non-parametric form of machine learning algorithm used for both classification and regression problems.
2. A tree like structure where nodes split the data points based on a feature and a split value.
    1. Internal Node or Node - a node is where a condition is present and a decision is made to split the data points.
    2. Root Node or Root - node present at the peak of decision tree.
    3. Leaf Node or Terminal Node - nodes present at the end of decision tree. These nodes do not have a condition or a decision to make because they have already met the stopping criteria.
    4. Subtree - a small part of a decision tree is called subtree.



3. Working
    1. The tree starts from a root node that has all the data points and split those data points at each subsequent node based on a criteria.
    2. At each node, a feature-value combination splits the data points with maximum homogeneity until a stopping criteria is met.
    3. Homogeneity is measured by calculating the impurity. Impurity is the inability of the model to split the data perfectly. Various metrics are used to measure impurity.
    4. Low impurity means high homogeneity and high homogeneity means good split.
4. Regression Tree
    1. For split, a feature-value combination is selected based on mse or mae which are the metrics use to calculate impurity for regression problems.
    2. Values in all the nodes are the averages of data points present in that node.
    3. Error is calculated with respect to the means of data points present in a both nodes after split.
5. Classification Tree
    1. For split, a feature-value combination is selected based on Gini impurity or entropy etc which are the metrics use to calculate impurity for classification problems.
    2. Values in all the nodes are based on which category has a higher probability based on data points present in that node.
6. Advantages
    1. Decision trees are highly interpretable. We can plot a tree and track each data point enabling us to understand how the model gives output.
    2. Decision trees are immune to outliers and scaling is also not required which save time during data processing.
    3. Decision trees work well for non-linear relationship between predictor variables and output.
7. Disadvantages
    1. Decision trees are prone to very high variance and hence overfit. They usually give good training accuracy but bad testing accuracy. Or we can say that same model trained on same data will build very distinct tree structures every time.
    2. A full grown decision tree will definitely overfit i.e. track each data point instead of understanding the underlying trend.

TREE PRUNING

1. Decision trees are prone to overfit which leads to high training accuracy but low testing accuracy.

2. Pre-Pruning — is to control the tree growth while building using model hyperparameters.
3. Post-Pruning — is a method used to remove the insignificant subtrees from a full grown decision tree.
4. Most commonly used post-pruning technique is called Cost-Complexity Pruning or Weakest Link Pruning
5. The main idea behind CCP is to add some amount of bias to the model which will reduce variance and maintain a balance between bias and variance.
6. Bias is added by adding a penalty to the error. This penalty value factors in the number of leaf nodes in the tree. So more number of leaves, larger the penalty.

Cost Complexity Pruning

1. Replace the splits with leaf nodes until the root node.
2. Now we have a number of multiple tree structures.
3. Calculate Tree Score for each structure with set value of alpha.

$$\text{Tree Score} = \text{Error} + \alpha . T$$

where T = number of terminal nodes [leaf nodes in tree]

Alpha = tuning parameter

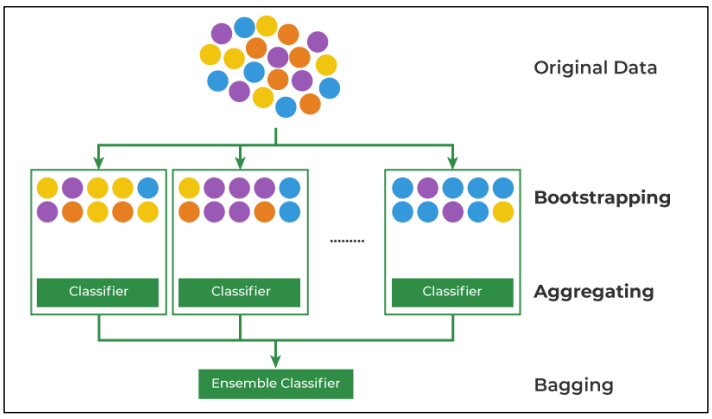
4. The one with the lowest score will be the best option.
5. The value of alpha affects the choice of tree. Choosing the value of alpha is very important in cost-complexity pruning.

ENSEMBLE LEARNING

1. Ensemble learning is a supervised machine learning technique that combines the strengths of multiple machine learning models to make predictions.
2. Various ensemble learning methods are —
    1. Bagging [parallel]
    2. Boosting [sequential]
    3. Stacking

BAGGING

- Bagging is an ensemble learning technique where multiple models are trained parallel to each other to make predictions by aggregating or majority voting.



Working

1. Bootstrapping — is a resampling technique used to create multiple training sets [bootstraps] randomly from a single training set with replacement. Here with replacement means that a data point may be chosen more than once or even not be chosen to create a bootstrap.
2. Training — each bootstrap is trained on multiple models parallel to each other.
3. Predictions — the predictions of all models are combined. The predictions are either averaged or voted based on the type of problem.

Advantages

1. Reduced variance —
    1. Predictions of multiple models combined together through averaging or majority voting makes the output more stable. This technique is good for models having high variance.

2. For example, a full grown decision tree has very high variance and low bias. Combining the results of n number of full grown decision trees will reduce the variance thus creating a stable model with low bias. An unpruned decision tree is used for bagging because of high variance.

Disadvantages

1. Bagging focuses on reducing variance and and has not influence on bias.
2. When compared to Decision trees, Bagging models have low interpretability and high computational cost.
3. High correlation between models —
    1. Bagging uses complete set of features to train models where some features might dominate over other features. So when a decision tree is used as a base estimator, this creates a situation where trained models are almost identical to each other because they prefer dominating features for split. Hence, there is high correlation between models which prevents bagging to reduce variance by a significant level.
    2. High correlation between models is a problem while using decision trees as base estimators but may or may not be a problem for other machine learning models because of their distinct learning methods.

RANDOM FOREST

- Random Forest is an ensemble learning technique where multiple decision trees are trained parallel to each other to make predictions by averaging or majority voting.
- But instead of using a complete set of features like Bagging, random forest uses a random subset of features.
- Bagging is also an ensemble learning technique where models are trained parallel to each other but bagging uses complete set of features to train the model which leads to high model correlation. On the other hand, random forest uses a random subset of features to train decision trees to prevent the dominance of some features above others.

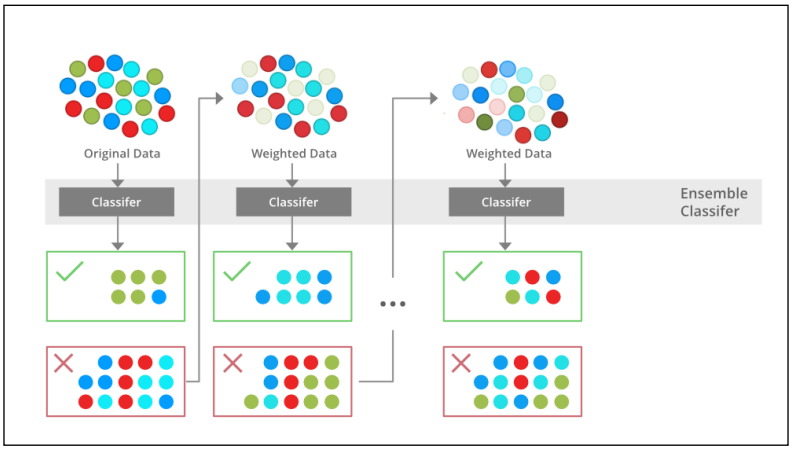
Working

1. Bootstrapping
    1. A resampling technique used to create multiple training sets [bootstraps] randomly from a single training set with replacement.
    2. A data point may be chosen more than once or even not be chosen to create a bootstrap.
2. Random feature selection
    1. Instead of using complete set of features for training trees, random forest uses a random subset of features to prevent the dominance of some features over others.
    2. Regression, p/4 features generally
    3. Classification, square root of p features generally
    4. If the features are highly correlated, use even smaller subset of features
3. Model training — Each bootstrap is trained on decision trees parallel to each other.
4. Prediction — The predictions of all trees are combined. The predictions are averaged or voted based on the type of problem.

- Adding more and more trees to Random Forest and Bagging models is not always beneficial. More trees eventually just repeat what others already do. The reduction of variance eventually reaches plateau. So adding more trees only makes the model computationally expensive without bringing any significant improvement to the model.

BOOSTING

- Boosting is an ensemble learning technique where multiple models are trained iteratively and each consecutive model learns from the mistakes of previous model.
- The idea behind boosting is that we first build a model on the training dataset and then build a second model to rectify the errors present in the first model.
- Individual models in boosting are called weak learners.
- Boosting techniques are specifically used to handle low bias [enhance accuracy]



Advantages

1. Enhanced accuracy — boosting enhances accuracy by working on the mistakes made by previous models.
2. Captures underlying trends — some uncomplicated machine learning models are often unable to capture the underlying trends in a complicated dataset. Boosting algorithm iteratively captures the underlying trends that the previous models were unable to capture.

Disadvantages

1. Sensitive to outliers — outliers lie far away from other data points so they give high error. When an outlier is encountered by a Boosting algorithm, it puts a lot of weight on them for the next learner.
2. Boosting methods are very prone to overfit if not tuned properly.

1. **AdaBoost [Adaptive Boosting]**

- AdaBoost is an ensemble learning technique where weak learners are trained iteratively while updating the weights of data points and allocating weights to weak learners for final prediction.
- Generally, decision stumps are used as weak learners. Decision stumps are decision trees with a single split [max depth = 1]
- AdaBoost improves accuracy by iteratively training data on decision stumps.
- It calculates amount-of-say [alpha] for each selected stump that defines the influence of that stump on final prediction and also updates the weights of data points to feed to the next stump.
- A decision stump that gives low error gets high value of alpha which means more influence on the final prediction and vice-versa. Also updating of the weights of data points ensure which data points have to be dealt with on the next stump.
- For prediction, we select the class with highest sum of alphas [classification] or calculate weighted average of predictions of all stumps [regression]

Working

1. Allocate equal weights to each data point [weight = 1/n, n being the number of rows]
2. Create stumps with each predictor variable, calculate impurity and choose the stump with least impurity.
3. Calculate alpha to allocate weight to that stump for final prediction.

   alpha = 1/2 Log(1-error/error)
4. Calculate updated weights of data points using alpha.

   updated weight for incorrectly classified = weight X e^ alpha

   update weight for correctly classified = weight X e^(-alpha)
5. Sum of updated weights should be equal to 1. So, normalise the new updated weights [divide each updated weight with sum of all updated weights]
6. Repeat the steps above [weighted gini impurity is calculated to select stumps]

Application

1. Simple boosting method majorly used for classification and known for its ease of implementation.
2. AdaBoost is flexible and can work with a variety of weak learner like decision tree, linear models etc
3. The performance of AdaBoost on complicated datasets may not be as good as other complex boosting algorithms but it can still perform well in many cases.

4. AdaBoost is very sensitive to outliers because new weights are based on alpha, alpha is calculated from error and outliers have a significant influence on error.
5. AdaBoost is prone to overfit with very high number of weak learners.

2. Gradient Boosting

- Gradient boosting is a powerful ensemble learning technique known for high speed and accuracy.
- The aim of gradient boosting is to minimise the loss function. It iteratively trains models to reduce error of the previous model.
- AdaBoost updates the weights of data points while GBM calculates the residuals of data points. Residuals are actual values minus predicted values.
- Consecutive residual values keep on reducing and the speed of reduction depends on a factor called learning rate.
- The loss function used are mse[regression] and log-likelihood[classification]
- Final prediction is the sum of outputs of all the weak learners with a multiplication factor [learning rate]
- Final prediction is equal to the sum of product of learning rate and residual output.

$$\text{prediction} = \text{1st prediction} + \Sigma\ (\text{LR X residuals})$$

- Learning Rate
    1. Learning rate handles how fast the trees make adjustments at each iteration.
    2. It is an important parameter in gradient boosting.
    3. High learning rate
        1. The trees at each iteration will make large adjustments and learn faster.
        2. Good accuracy will be achieved with less trees.
        3. Prone to overfit to training data [low accuracy on unseen data]
    4. Low learning rate
        1. The trees at each iteration will make small adjustments and learn slower.
        2. Good accuracy will be achieved with higher number of trees prevent overfit.

Working [Regression]

1. Calculate average of continuous output variable [1st prediction]
2. Calculate pseudo-residuals for each data point with respect to that average.
3. Create a decision tree [small decision tree, low depth] using predictor variables and pseudo-residuals as output variable.
4. Predict new output using learning rate.

$$\text{prediction} = \text{1st prediction} + (\text{LR X residual})$$

where residuals are values in leaf nodes of decision tree

5. Calculate new pseudo-residuals for each data point (observed values — predicted values)
6. Create another tree using predictor variables and new pseudo-residuals as output variable.
7. Again predict new output using learning rate.

$$\text{prediction} = \text{1st prediction} + (\text{LR X residual}) + (\text{LR X new residual})$$

8. Again calculate new pseudo-residuals for each data point (observed values — new predicted values)
9. Repeat until a stopping criteria is met.
10. Consecutive residual values will keep on reducing and the speed of reduction will depend on learning rate.

$$\text{prediction} = \text{1st prediction} + \Sigma\ (\text{LR X residuals})$$

Working [Binary Classification]

1. Calculate log(odds) for output variable [1st prediction]
2. Calculate probability of log(odds) for calculating pseudo-residuals.
3. Calculate pseudo-residuals for each data point with respect to probability of log(odds). Example, a data point has output 1 and probability of log(odds) is 0.7 so pseudo-residual is equal to 1-0.7 = 0.3
4. Create a decision tree using predictor variables and pseudo-residuals as output variable.
5. Predict values using learning rate.
6. But we can not predict values directly because leaf nodes have probability values and 1st prediction is a log(odds) value. Convert probability to log(odds) and then predict values.

7. Again calculate probability for predicted log(odds) values and again calculate pseudo-residuals.
8. Repeat until a stopping criteria is met.

Application

1. Gradient boosting is known for its high predictive accuracy to complicated datasets.
2. Gradient boosting is less sensitive to outliers than AdaBoost. Because gradient boosting minimises the overall loss function and adaboost calculates alpha using error to update the weights on data points and outliers have a significant influence on error.
3. Like other boosting methods, gradient boosting is also prone to overfit if not tuned properly [number of weak learners is very high]
4. Gradient boosting can often get computationally very expensive for big datasets.
5. There are popular variants of gradient boosting, such as XGBoost which offer advancements in terms of speed, performance, and flexibility.

3. Extreme Gradient Boosting

- XGBoost is an ensemble learning technique built on gradient boosting. It is an optimised implementation of gradient boosting.
- Hyperparameter tuning is an essential part of XGBoost
- Additional features of XGBoost over Gradient boosting.
    1. Regularisation
        1. Uses L1and L2 regularisation methods to prevent overfitting.
        2. L1 — Lasso Reg. | L2 — Ridge Reg.
        3. Regularisation adds a penalty term to loss functions to reduce complexity.
        4. L1 due to its stronger penalty term reduces the number of features to prevent overfitting but it can also increase bias if important features are removed.
        5. L2 on the other hand has a relatively weaker penalty term to prevent overfitting. It can also reduce bias by understanding the underlying trend now.
        6. Model hyperparameter, L2 — lambda | L1 — alpha.
    2. Handling Missing Values
        1. XGBoost has a built-in feature to deal with missing values called Sparsity-Aware Split Finding. Missing value imputation is not required.
    3. Parallelisation
        1. XGBoost was designed to build scalable complicated models on extremely large datasets. which are computationally expensive and time consuming.
        2. As a solution, XGBoost does parallel computing, it uses multiple cores of the CPU to train models which significantly reduces training time.
    4. Built-In Cross Validation
        1. xgboost.cv
        2. Normally, a machine algorithm would require external help to implement the Cross-Validation, but XGBoost has an in-built Cross-Validation that could be used for training.
        3. The Cross-Validation would be performed at each boosting iteration and ensure the produced tree is robust.

Application

1. XGBoost is known for building complicated models on extremely large and complicated datasets which give high predictive accuracy and allows scalability.
2. XGBoost gives significant reduction in variance due to its added regularisation feature.
3. Provision to use parallel computing to save time and computing power.
4. Flexible model providing many options for hyperparameter tuning.