Statistics Foundation

Machine Learning – Decision Tree

Digital Vidya

Decision Tree

- **Why Decision Tree**
  - They are supervised learning algorithm which has a pre-defined target variable
  - They are mostly used in non-linear decision making with simple linear decision surface
  - They are adaptable for solving both classification or regression kind of problems

Decision Tree

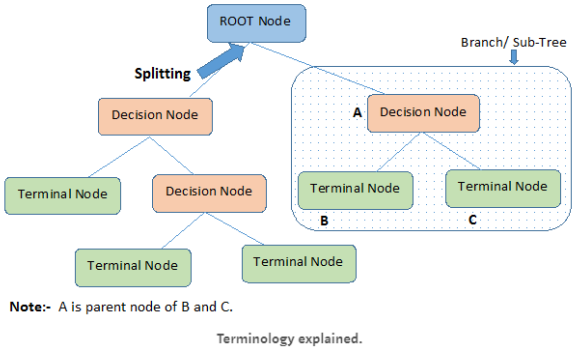
- **Why Decision Tree**

  - They empower predictive modeling with higher accuracy, better stability and provide ease of interpretation.
  - Unlike linear modeling techniques, they map non-linear relationships quite well.

Decision Tree

- **How Decision Tree works**

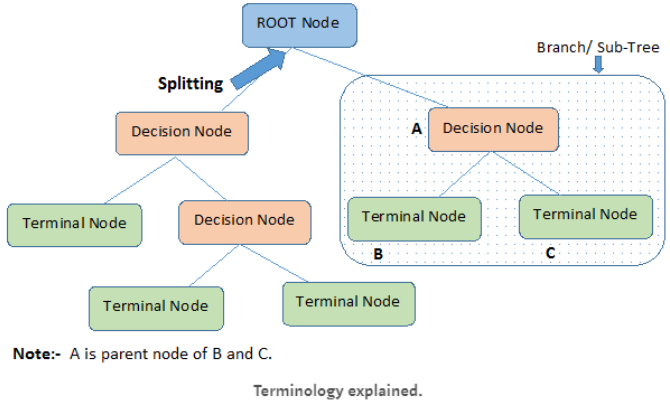
  - It breaks a dataset into smaller subsets, and at the same time, an associated decision tree is incrementally developed.
  - As it happens in real life, we consider the most important factor, and divide possibilities according to it.
  - Similarly, tree building starts by finding the variable/feature for best split.



Decision Tree

- **Decision Tree Terminologies**

  - Root Node: Entire population or sample, further gets divided into two or more homogeneous sets.
  - Parent and Child Node: Node which is divided into sub-nodes is called parent node, whereas sub-nodes are the child of parent node.
  - Splitting: Process of dividing a node into two or more sub-nodes.
  - Decision Node: A sub-node that splits into further sub-nodes.
  - Leaf/ Terminal Node: Nodes that do not split.
  - Pruning: When we remove sub-nodes of a decision node, this process is called pruning. (Opposite of Splitting)
  - Branch/Sub-Tree: Sub-section of entire tree.



Decision Tree

- **What is a Decision Tree**
  - A decision tree consists of the root /Internal node which further splits into decision nodes/branches, depending on the outcome of the branches the next branch or the terminal /leaf nodes are formed

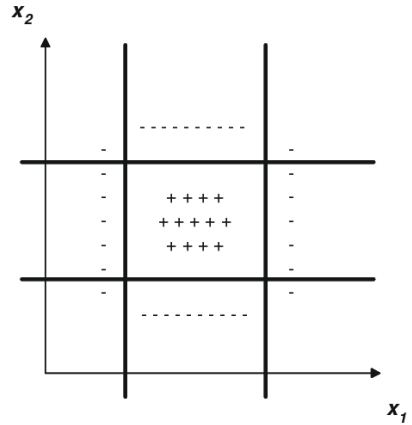
Alternatingly think that decision trees are a group of nested IF-ELSE conditions which can be modeled as a tree wherein the decision are made in the internal node and output is obtained in the leaf node.

```
if (Height > 5.6):
    output = TOUCHDOWN
elif (speed < 13.2 m.p.h):
    output = INTERCEPTION
elif (weight > 200 lbs):
    output = TOUCHDOWN
else:
    output = INTERCEPTION
```

Nested if-else

Decision Tree

- **Geometrical analogy:**
  - Geometrically we can think decision trees as a set of a number of axis parallel hyperplanes which divides the space into the number of hypercuboids during the inference. we classify the point based on which hypercuboid it falls into.



Axis-parallel hyperplanes

Decision Tree

- **What is Entropy method ?**
  - Definition: Entropy is the measures of impurity, disorder or uncertainty in a bunch of examples.
  - Entropy controls how a Decision Tree decides to split the data. It actually effects how a Decision Tree draws its boundaries.

The Equation of Entropy:

$$Entropy = -\sum p(X) \log p(X)$$

here p(x) is a fraction of examples in a given class

Equation of Entropy

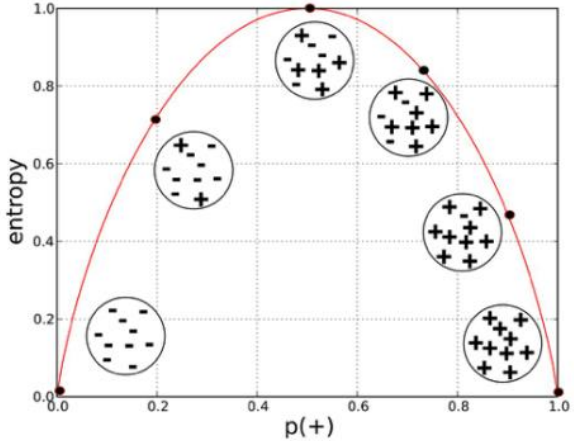
Decision Tree

- **What is Entropy method ?**
  - Let's say we only have two classes , a positive class and a negative class. Therefore 'i' here could be either + or (-).
  - So if we had a total of 100 data points in our dataset with 30 belonging to the positive class and 70 belonging to the negative class
  - Then 'P+' would be 3/10 and 'P-' would be 7/10

$$-\frac{3}{10}\times\log_2\left(\frac{3}{10}\right)-\frac{7}{10}\times\log_2\left(\frac{7}{10}\right)\approx 0.88$$

Entropy in this example

Decision Tree



Entropy variation for Pure/Impure Nodes

- **What is Entropy method ?**
  - The x-axis measures the proportion of data points belonging to the positive class in each bubble and the y-axis axis measures their respective entropies.
  - We can see the inverted ‘U’ shape of the graph. Entropy is lowest at the extremes, when the bubble either contains no positive instances or only positive instances.
  - That is, when the bubble is pure the disorder is 0. Entropy is highest in the middle when the bubble is evenly split between positive and negative instances. Extreme disorder , because there is no majority.

Decision Tree

$$IG(Y, X) = E(Y) - E(Y|X)$$

Information Gain from X on Y

- **What is Information Gain ?**
  - Now we know how to measure disorder (using Entropy).
  - Next we need a metric to measure the reduction of this disorder in our target variable/class given additional information(features/independent variables) about it. This is where Information Gain comes in

Decision Tree

$$IG(Y, X) = E(Y) - E(Y|X)$$

Information Gain from X on Y

- **What is Information Gain ?**
  - We simply subtract the entropy of Y given X from the entropy of just Y to calculate the reduction of uncertainty about Y given an additional piece of information X about Y
  - This is called Information Gain
  - The greater the reduction in this uncertainty, the more information is gained about Y from X.

Decision Tree

- **Example: Decision Tree**

  Consider an example where we are building a decision tree to predict whether a loan given to a person would result in a write-off or not :

  - Our entire population consists of 30 instances
  - 16 belong to the write-off class
  - Other 14 belong to the non-write-off class
  - We have two features, namely “Balance” that can take on two values -> “< 50K” or “>50K”
  - “Residence” that can take on three values -> “OWN”, “RENT” or “OTHER”

  Let us see how a decision tree algorithm would decide what attribute to split on first and what feature provides more information, or reduces more uncertainty about our target variable out of the two using the concepts of Entropy and Information Gain.

Decision Tree

- **Example: Decision Tree**

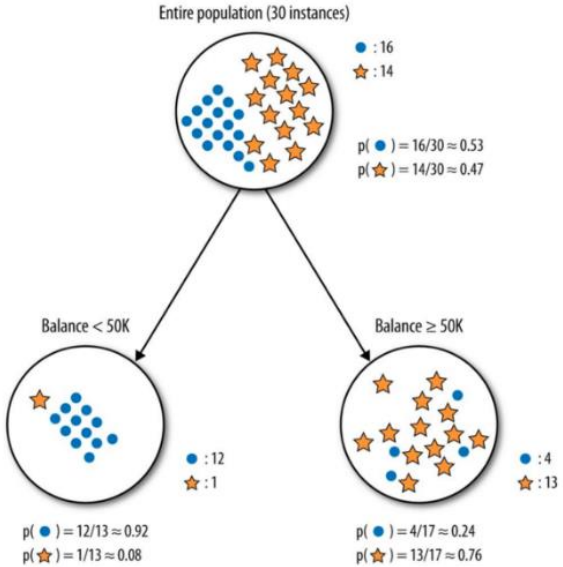
Split on Feature 1: Balance

The dots are the data points with class right-off and the stars are the non-write-offs

Splitting the parent node on attribute balance gives us 2 child nodes

The left node gets 13 of the total observations with 12/13 (0.92 probability) observations from the write-off class and only 1/13(0.08 probability) observations from the non-write of class

The right node gets 17 of the total observation with 13/17(0.76 probability) observations from the non-write-off class and 4/17 (0.24 probability) from the write-off class



Decision Tree

- **Example: Decision Tree**

Split on Feature 1: Balance

Let's calculate the entropy for the parent node and see how much uncertainty the tree can reduce by splitting on Balance

$$E(Parent) = -\frac{16}{30}\log_2\left(\frac{16}{30}\right) - \frac{14}{30}\log_2\left(\frac{14}{30}\right) \approx 0.99$$

$$E(Balance < 50K) = -\frac{12}{13}\log_2\left(\frac{12}{13}\right) - \frac{1}{13}\log_2\left(\frac{1}{13}\right) \approx 0.39$$

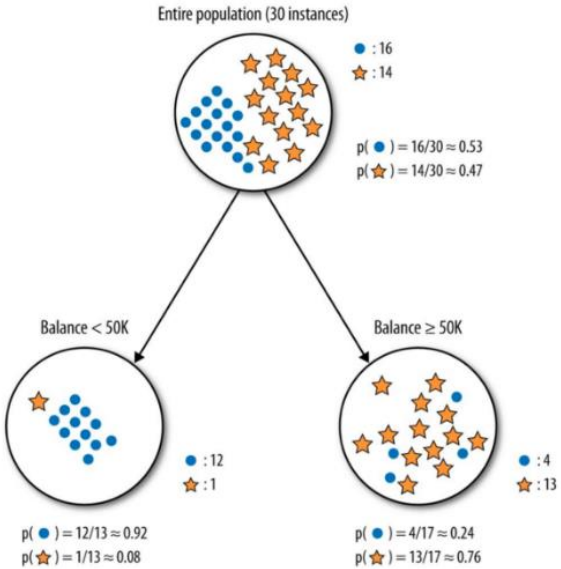
$$E(Balance > 50K) = -\frac{4}{17}\log_2\left(\frac{4}{17}\right) - \frac{13}{17}\log_2\left(\frac{13}{17}\right) \approx 0.79$$

Weighted Average of entropy for each node:

$$E(Balance) = \frac{13}{30} \times 0.39 + \frac{17}{30} \times 0.79$$
$$= 0.62$$

Information Gain:

$$IG(Parent, Balance) = E(Parent) - E(Balance)$$
$$= 0.99 - 0.62$$
$$= 0.37$$



Decision Tree

- **Example: Decision Tree**

Split on Feature 2: Residence

Let's calculate the entropy for the parent node and see how much uncertainty the tree can reduce by splitting on Residence

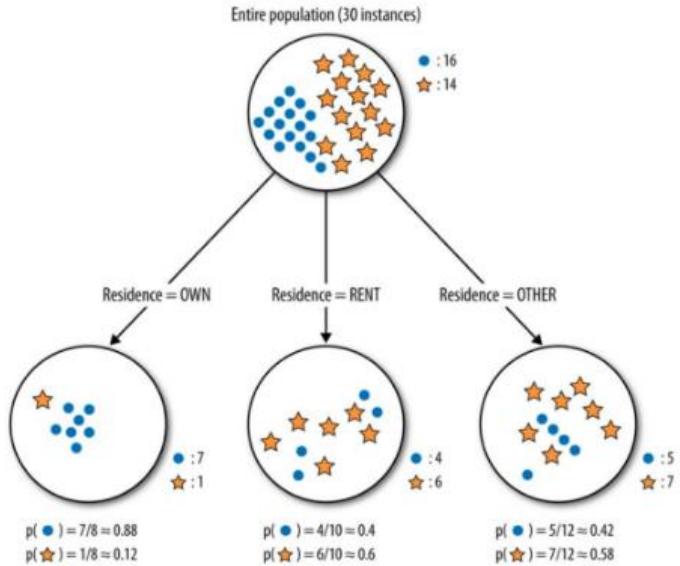
$$E(Residence = OWN) = -\frac{7}{8}\log_2\left(\frac{7}{8}\right) - \frac{1}{8}\log_2\left(\frac{1}{8}\right) \approx 0.54$$

$$E(Residence = RENT) = -\frac{4}{10}\log_2\left(\frac{4}{10}\right) - \frac{6}{10}\log_2\left(\frac{6}{10}\right) \approx 0.97$$

$$E(Residence = OTHER) = -\frac{5}{12}\log_2\left(\frac{5}{12}\right) - \frac{7}{12}\log_2\left(\frac{7}{12}\right) \approx 0.98$$

Weighted Average of entropies for each node:

$$E(Residence) = \frac{8}{30} \times 0.54 + \frac{10}{30} \times 0.97 + \frac{12}{30} \times 0.98 = 0.86$$



Information Gain:

$$IG(Parent, Residence) = E(Parent) - E(Residence)$$
$$= 0.99 - 0.86$$
$$= 0.13$$

Decision Tree

- By itself the feature, Balance provides more information about our target variable than Residence
- It reduces more disorder in our target variable.
- A decision tree algorithm would use this result to make the first split on our data using Balance
- From here on, the decision tree algorithm would use this process at every split to decide what feature it is going to split on next
- In a real world scenario , with more than two features the first split is made on the most informative feature and then at every split the information gain for each additional feature needs to be done
- A decision tree would repeat this process as it grows deeper and deeper till either it reaches a pre-defined depth or no additional split can result in a higher information gain beyond a certain threshold which can also usually be specified as a hyper-parameter

Decision Tree

- **What is Gini Impurity**

  Gini impurity can be considered as an alternative for the entropy method. Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset.

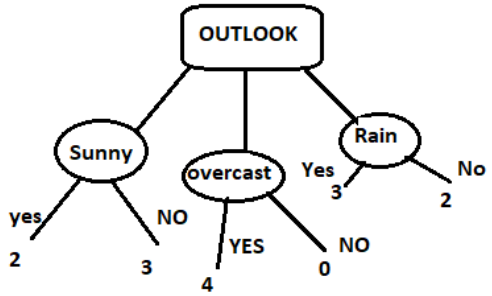
$$G.I\ (Y) = 1 - \sum_{i=1}^{k} [p(Y)]^2$$

  Calculation of Entropy in Gini method behaves in the same way but entropy involves a log computation and Gini impurity involved a square computation. Since computing square is cheaper than logarithmic function we prefer Gini impurity over entropy.

Decision Tree

- **What is Pure Node**

  Pure node is a node wherein all the data points belong to the same class and thus it is very easy to make the prediction at such node.



Here overcast is a pure node

Decision Tree

- **Overfitting & Underfitting in CART**
- If we continue to grow the tree fully until each leaf node corresponds to the lowest impurity, then the data have typically been overfitted
- If splitting is stopped too early, error on training data is not sufficiently high and performance will suffer due to bias
- Thus, preventing overfitting & underfitting are pivotal while modeling a decision tree and it can be done in 2 ways:
    - **Setting constraints on tree size**
    - **Tree pruning**

Decision Tree

- **Overfitting & Underfitting in CART**

  - **Setting constraints on tree size**

    - Providing minimum number of samples for a node split.
    - Deploying the minimum number of samples for a terminal node (leaf).
    - Allowing maximum depth of tree (vertical depth).
    - Maximum number of terminal nodes.
    - Maximum features to consider for the split.

Decision Tree

- **Overfitting & Underfitting in CART**

  - **Tree Pruning**

    - Pruning is a technique in machine learning that reduces the size of decision trees by removing sections of the tree
    - It also reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction of overfitting
    - Tree prunning can be done in two ways by pre-prunning or by post-prunning

Decision Tree

- **Overfitting & Under-fitting in CART**

  - **Pre-pruning:**

  — Stop splitting the current node if it does not improve the entropy by at least some pre-set(threshold) value

  — Stop partitioning if the number of data points are less then some pre-set (Threshold) values

  — Restricting the depth of the tree to some pre-set(Threshold) value

  - **Post-pruning:**

  It can be done by first allowing the tree to grow to its full potential and then pruning the tree at each level after calculating the cross-validation accuracy at each level.

Decision Tree

- **Advantages of CART:**
  - Decision trees can inherently perform multiclass classification
  - They provide most model interpretability because they are simply series of if-else conditions
  - They can handle both numerical and categorical data
  - Nonlinear relationships among features do not affect the performance of the decision trees

Decision Tree

- **Disadvantages of CART:**
  - A small change in the dataset can make the tree structure unstable which can cause high variance
  - Decision tree learners create underfit trees if some classes are imbalanced. It is therefore recommended to balance the data set prior to fitting with the decision tree

Decision Tree

- **Preparing data for CART:**
  - The splitting of numerical features can be performed by sorting the features in the ascending order and trying each value as the threshold point and calculating the information gain for each value as the threshold. Finally, if that value obtained is equal to the threshold which gives the maximum I.G value then that point is chosen as the split point
  - Feature scaling(column standardization) not necessary to perform in decision trees. However, it helps with data visualization/manipulation and might be useful if you intend to compare performance with other data or other methods like SVM.
  - In order to handle categorical features in Decision trees, we must never perform one hot encoding on a categorical variable even if the categorical variables are nominal since most of the libraries can handle categorical variables automatically. we can still assign a number for each variable if desired.
  - If height or depth of the tree is exactly one then such a tree is called as a decision stump.

Decision Tree

- **Preparing data for CART:**
  - Imbalanced class does have a detrimental impact on the tree's structure so it can be avoided by either using up-sampling or by using down-sampling depending upon the dataset
  - Apart from skewed classes, high dimensionality can also have an adverse effect on the structure of the tree if dimensionality is very high that means we have a lot of features which means that to find the splitting criterion on each node it will consume a lot of time
  - Outliers also impact the tree's structure as the depth increases the chance of outliers in the tree increases

Decision Tree

Python Code

```
#Import Library

#Import other necessary libraries like pandas, numpy...

from sklearn import tree

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset

# Create tree object

model = tree.DecisionTreeClassifier(criterion='gini') # for classification, here you can change the algorithm as gini or entropy (information gain) by default it is gini

# model = tree.DecisionTreeRegressor() for regression

# Train the model using the training sets and check score

model.fit(X, y)

model.score(X, y)

#Predict Output

predicted= model.predict(x_test)
```

Ensemble Methods

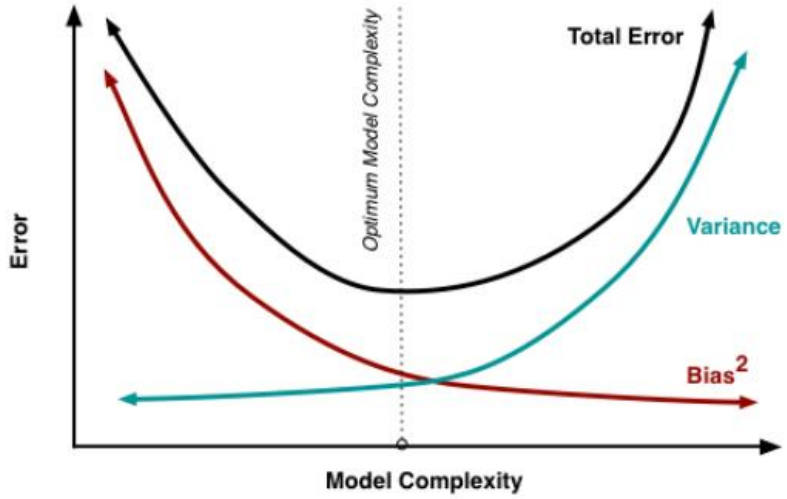
Ensemble Methods

- The literary meaning of word ‘ensemble’ is group
- Ensemble methods involve group of predictive models to achieve a better accuracy and model stability
- Ensemble methods are known to impart supreme boost to tree based models
- Like every other model, a tree based model also suffers from the plague of bias and variance.
- Bias means, ‘how much on an average are the predicted values different from the actual value.’
- Variance means, ‘how different will the predictions of the model be at the same point if different samples are taken from the same population’.
- With a small tree we will get a model with low variance and high bias
- How do we balance the trade off between bias and variance ?
- Normally, as the complexity of model increases, there is a reduction in prediction error due to lower bias in the model
- As the model grows more complex, we end up over-fitting model and the model will start suffering from high variance

Ensemble Methods

Ensemble Methods

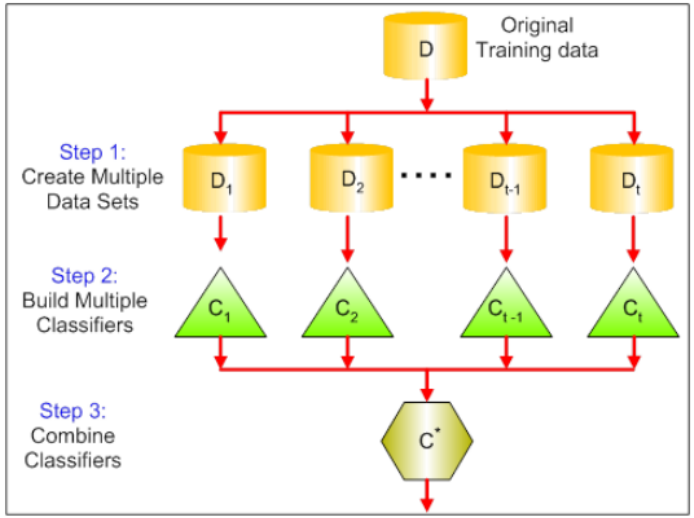
A champion model should maintain a balance between these two types of errors. This is known as the trade-off management of bias-variance errors. Ensemble learning is one way to execute this trade off analysis.



Ensemble Methods

Bagging

Bagging is a technique used to reduce the variance of our predictions by combining the result of multiple classifiers modelled on different sub-samples of the same data set.



Ensemble Methods

Steps to perform Bagging

1. Create Multiple DataSets:

- Sampling is done with replacement on the original data and new datasets are formed.
- The new data sets can have a fraction of the columns as well as rows, which are generally hyper-parameters in a bagging model
- Taking row and column fractions less than 1 helps in making robust models, less prone to overfitting

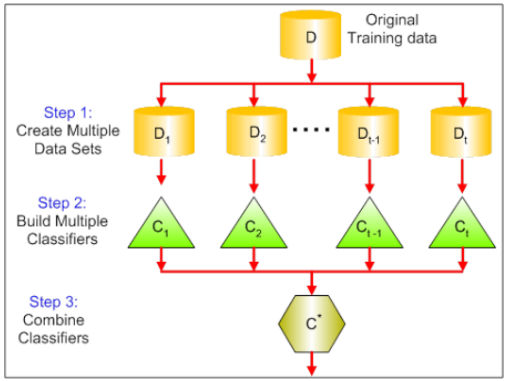
2. Build Multiple Classifiers:

- Classifiers are built on each data set.
- Generally the same classifier is modelled on each data set and predictions are made.

3. Combine Classifiers:

- The predictions of all the classifiers are combined using a mean, median or mode value depending on the problem at hand.
- The combined values are generally more robust than a single model

There are various implementations of bagging models. Random forest is one of them



Random Forests

- **Random Forests**

  Random Forest is a versatile machine learning method capable of performing both regression and classification tasks. It also undertakes dimensional reduction methods, treats missing values, outlier values and other essential steps of data exploration, and does a fairly good job. It is a type of ensemble learning method, where a group of weak models combine to form a powerful model.

- In Random Forest, we grow multiple trees as opposed to a single tree in CART model
- To classify a new object based on attributes, each tree gives a classification and we say the tree “votes” for that class
- The forest chooses the classification having the most votes (over all the trees in the forest) and in case of regression, it takes the average of outputs by different trees.

Random Forests

- **Steps to build Random Forest**

  - Assume number of cases in the training set is N. Then, sample of these N cases is taken at random but with replacement. This sample will be the training set for growing the tree
  - If there are M input variables, a number m<M is specified such that at each node, m variables are selected at random out of the M.
  - The best split on these m is used to split the node. The value of m is held constant while we grow the forest
  - Each tree is grown to the largest extent possible and there is no pruning
  - Predict new data by aggregating the predictions of the ntree trees (i.e., majority votes for classification, average for regression)

Random Forests

- **Advantages**

  - This algorithm can solve both type of problems i.e. classification and regression and does a decent estimation at both fronts
  - One of benefits of Random forest is, the power of handle large data set with higher dimensionality
  - It can handle thousands of input variables and identify most significant variables so it is considered as one of the dimensionality reduction methods
  - Further, the model outputs Importance of variable, which can be a very handy feature (on some random data set).
  - It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.

Random Forests

- **Advantages**

  - It has methods for balancing errors in data sets where classes are imbalanced.
  - The capabilities of the above can be extended to unlabelled data, leading to unsupervised clustering, data views and outlier detection
  - Random Forest involves sampling of the input data with replacement called as bootstrap sampling. Here one third of the data is not used for training and can be used to testing. These are called the out of bag samples
  - Error estimated on these out of bag samples is known as out of bag error. Study of error estimates by Out of bag, gives evidence to show that the out-of-bag estimate is as accurate as using a test set of the same size as the training set
  - Therefore, using the out-of-bag error estimate removes the need for a set aside test set

Random Forests

- **Disadvantages**

  - It surely does a good job at classification but not as good as for regression problem as it does not give precise continuous nature predictions.
  - In case of regression, it doesn't predict beyond the range in the training data, and that they may over-fit data sets that are particularly noisy.
  - Random Forest can feel like a black box approach for statistical modellers – we have very little control on what the model does. We can at best – try different parameters and random seeds

Random Forest

Python Code

```
#Import Library
from sklearn.ensemble import RandomForestClassifier #use RandomForestRegressor for regression problem
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset
# Create Random Forest object
model= RandomForestClassifier(n_estimators=1000)
# Train the model using the training sets and check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

Ensemble Method - Boosting

- **Boosting**

  Definition: The term 'Boosting' refers to a family of algorithms which converts weak learner to strong learners.

  Let's solve 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. If:

  - Email has only one image file (promotional image), It's a SPAM
  - Email has only link(s), It's a SPAM
  - Email body consist of sentence like "You won a prize money of $ xxxxxx", It's a SPAM
  - Email from our official domain "Analyticsvidhya.com" , Not a SPAM
  - 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.

Ensemble Method - Boosting

- **Boosting**

  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:

  - Using average/ weighted average
  - 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) vote for ‘SPAM’.

Ensemble Method - Boosting

- **Boosting**
  - An initial model F0 is defined to predict the target variable y
  - This model will be associated with a residual (y – F0)
  - A new model h1 is fit to the residuals from the previous step
  - Now, F0 and h1 are combined to give F1, the boosted version of F0. The mean squared error from F1 will be lower than that from F0:

$$F_1(x) <- F_0(x) + h_1(x)$$

  - To improve the performance of F1, we could model after the residuals of F1 and create a new model F2:

$$F_2(x) <- F_1(x) + h_2(x)$$

  - This can be done for ‘m’ iterations, until residuals have been minimized as much as possible:

$$F_m(x) <- F_{m-1}(x) + h_m(x)$$

Here, the additive learners do not disturb the functions created in the previous steps.

Instead, they impart information of their own to bring down the errors.

Gradient Boosting

Python Code

```
#import libraries
from sklearn.ensemble import GradientBoostingClassifier #For Classification
from sklearn.ensemble import GradientBoostingRegressor #For Regression
#use GBM function
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1)
clf.fit(X_train, y_train)
```

XGBoost

Python Code

```
# First XGBoost model for Pima Indians dataset
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# load data
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=",")
# split data into X and y
X = dataset[:,0:8]
Y = dataset[:,8]
# split data into train and test sets
seed = 7
test_size = 0.33
```

Python Code

```
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=test_size, random_state=seed)
# fit model no training data
model = XGBClassifier()
model.fit(X_train, y_train)
# make predictions for test data
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

XGBoost Features

- **Unique features of XGBoost**
  - XGBoost is a popular implementation of gradient boosting. Let's discuss some features of XGBoost that make it so interesting
  - Regularization: XGBoost has an option to penalize complex models through both L1 and L2 regularization. Regularization helps in preventing overfitting
  - Handling sparse data: Missing values or data processing steps like one-hot encoding make data sparse. XGBoost incorporates a sparsity-aware split finding algorithm to handle different types of sparsity patterns in the data
  - Weighted quantile sketch: Most existing tree based algorithms can find the split points when the data points are of equal weights (using quantile sketch algorithm). However, they are not equipped to handle weighted data. XGBoost has a distributed weighted quantile sketch algorithm to effectively handle weighted data

XGBoost Features

- **Unique features of XGBoost**
  - Block structure for parallel learning: For faster computing, XGBoost can make use of multiple cores on the CPU. This is possible because of a block structure in its system design. Data is sorted and stored in in-memory units called blocks. Unlike other algorithms, this enables the data layout to be reused by subsequent iterations, instead of computing it again. This feature also serves useful for steps like split finding and column sub-sampling
  - Cache awareness: In XGBoost, non-continuous memory access is required to get the gradient statistics by row index. Hence, XGBoost has been designed to make optimal use of hardware. This is done by allocating internal buffers in each thread, where the gradient statistics can be stored
  - Out-of-core computing: This feature optimizes the available disk space and maximizes its usage when handling huge datasets that do not fit into memory

Thank You

Digital Vidya