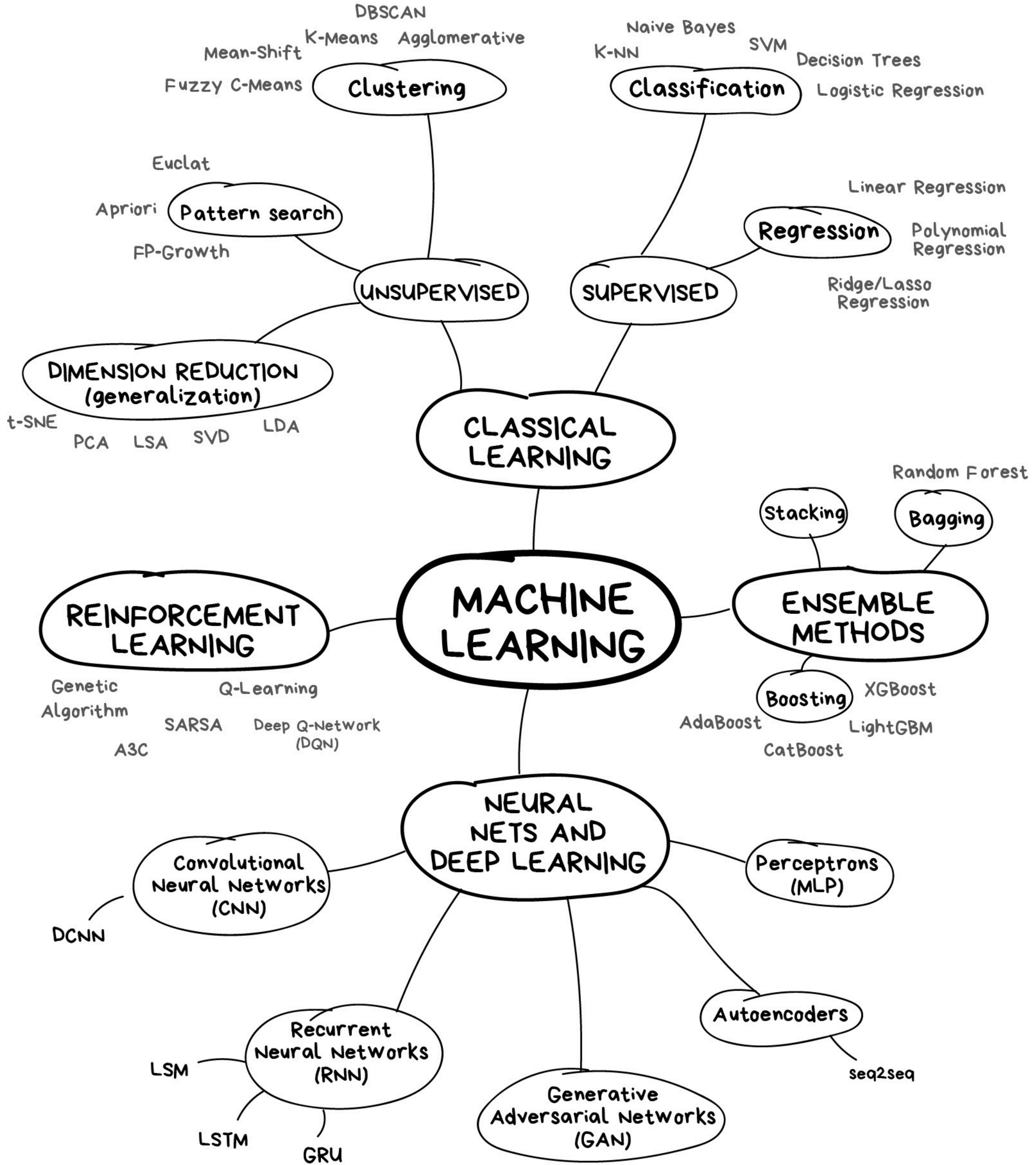
Machine Learning

Decision Tree – 2

Prof. Dr. Fazlul Hasan Siddiqui
Head, Dept. of CSE, DUET, Gazipur
BSc:IUT; MSc:BUET; PhD:ANU (Australia)
siddiqui@duet.ac.bd
Credit: www.analyticsvidhya.com/blog/2016/04/tree-based-algorithms-complete-tutorial-scratch-in-python/

Machine Learning Algorithms



Avoid/Counter Over-fitting in Decision Trees

Overfitting is one of the key challenges faced while using tree based algorithms. If there is no limit set of a decision tree, it will give you 100% accuracy on training set because in the worse case it will end up making 1 leaf for each observation. Thus, preventing overfitting is pivotal while modeling a decision tree and it can be done in 2 ways:

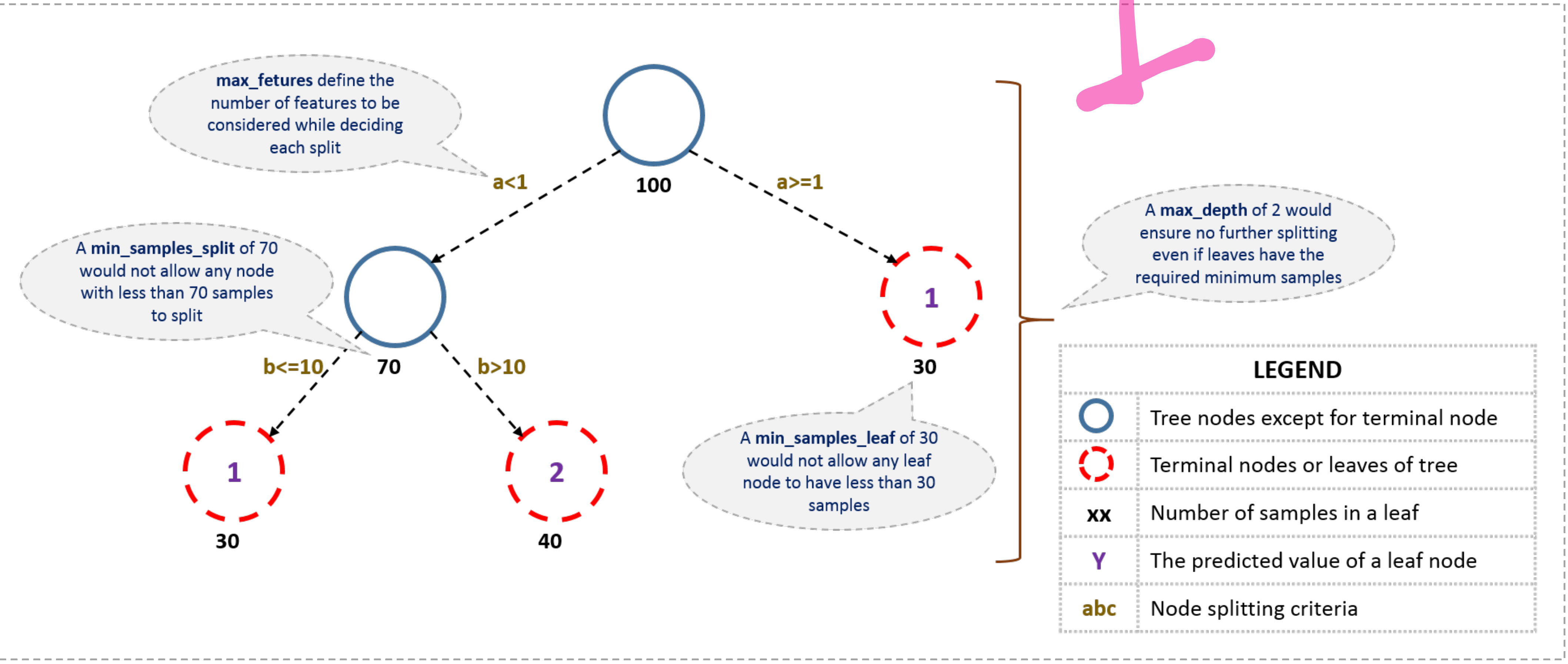
1. Setting constraints on tree size
2. Tree pruning

$\left.\right\}$ 2

Let's discuss both of these briefly.

Setting Constraints | Avoid/Counter Over-fitting

1


max_fetures define the number of features to be considered while deciding each split
a<1
100
a>=1
A min_samples_split of 70 would not allow any node with less than 70 samples to split
A max_depth of 2 would ensure no further splitting even if leaves have the required minimum samples
b<=10
70
b>10
1
30
1
30
2
40
A min_samples_leaf of 30 would not allow any leaf node to have less than 30 samples

LEGEND	
○	Tree nodes except for terminal node
◌	Terminal nodes or leaves of tree
xx	Number of samples in a leaf
Y	The predicted value of a leaf node
abc	Node splitting criteria

1. **Minimum samples for a node split**
    - Defines the minimum number of samples (or observations) which are required in a node to be considered for splitting.
    - Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
    - Too high values can lead to under-fitting hence, it should be tuned using CV.

2. **Minimum samples for a terminal node (leaf)**
    - Defines the minimum samples (or observations) required in a terminal node or leaf.
    - Used to control over-fitting similar to min_samples_split.
    - Generally lower values should be chosen for imbalanced class problems because the regions in which the minority class will be in majority will be very small.

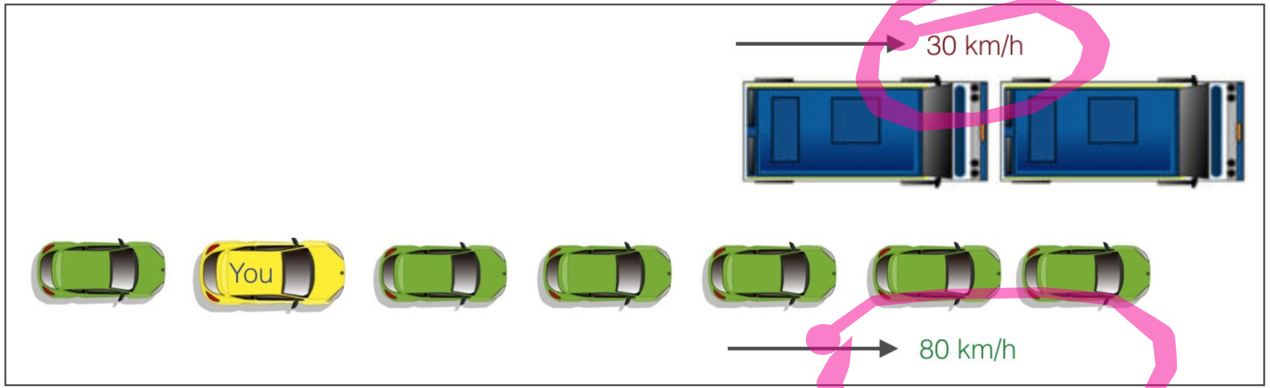
3. **Maximum depth of tree (vertical depth)**
    - The maximum depth of a tree.
    - Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.
    - Should be tuned using CV.

4. **Maximum number of terminal nodes**
    - The maximum number of terminal nodes or leaves in a tree.
    - Can be defined in place of max_depth. Since binary trees are created, a depth of 'n' would produce a maximum of 2^n leaves.

5. **Maximum features to consider for split**
    - The number of features to consider while searching for a best split. These will be randomly selected.
    - As a thumb-rule, square root of the total number of features works great but we should check upto 30-40% of the total number of features.
    - Higher values can lead to over-fitting but depends on case to case.

Pruning | Avoid/Counter Over-fitting

As discussed earlier, the technique of setting constraint is a greedy-approach. In other words, it will check for the best split instantaneously and move forward until one of the specified stopping condition is reached. Let's consider the following case when you're driving:



There are 2 lanes:

1. A lane with cars moving at 80km/h
2. A lane with trucks moving at 30km/h

At this instant, you are the yellow car and you have 2 choices:

1. Take a left and overtake the other 2 cars quickly
2. Keep moving in the present lane

Let's analyze these choice. In the former choice, you'll immediately overtake the car ahead and reach behind the truck and start moving at 30 km/h, looking for an opportunity to move back right. All cars originally behind you move ahead in the meanwhile. This would be the optimum choice if your objective is to maximize the distance covered in next say 10 seconds. In the later choice, you sale through at same speed, cross trucks and then overtake maybe depending on situation ahead. Greedy you!

Pruning | Avoid/Counter Over-fitting

Let's analyze these choice. In the former choice, you'll immediately overtake the car ahead and reach behind the truck and start moving at 30 km/h, looking for an opportunity to move back right. All cars originally behind you move ahead in the meanwhile. This would be the optimum choice if your objective is to maximize the distance covered in next say 10 seconds. In the later choice, you sale through at same speed, cross trucks and then overtake maybe depending on situation ahead. Greedy you!

This is exactly the difference between normal decision tree & pruning. A decision tree with constraints won't see the truck ahead and adopt a greedy approach by taking a left. On the other hand if we use pruning, we in effect look at a few steps ahead and make a choice.

So we know pruning is better. But how to implement it in decision tree? The idea is simple.

1. We first make the decision tree to a large depth.
2. Then we start at the bottom and start removing leaves which are giving us negative returns when compared from the top.
3. Suppose a split is giving us a gain of say -10 (loss of 10) and then the next split on that gives us a gain of 20. A simple decision tree will stop at step 1 but in pruning, we will see that the overall gain is +10 and keep both leaves.

Are tree based algorithms better than linear models?

“If I can use logistic regression for classification problems and linear regression for regression problems, why is there a need to use trees”? Many of us have this question. And, this is a valid one too.

Actually, you can use any algorithm. It is dependent on the type of problem you are solving. Let’s look at some key factors which will help you to decide which algorithm to use:

1. If the relationship between dependent & independent variable is well approximated by a linear model, linear regression will outperform tree based model.
2. If there is a high non-linearity & complex relationship between dependent & independent variables, a tree model will outperform a classical regression method.
3. If you need to build a model which is easy to explain to people, a decision tree model will always do better than a linear model. Decision tree models are even simpler to interpret than linear regression!

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.

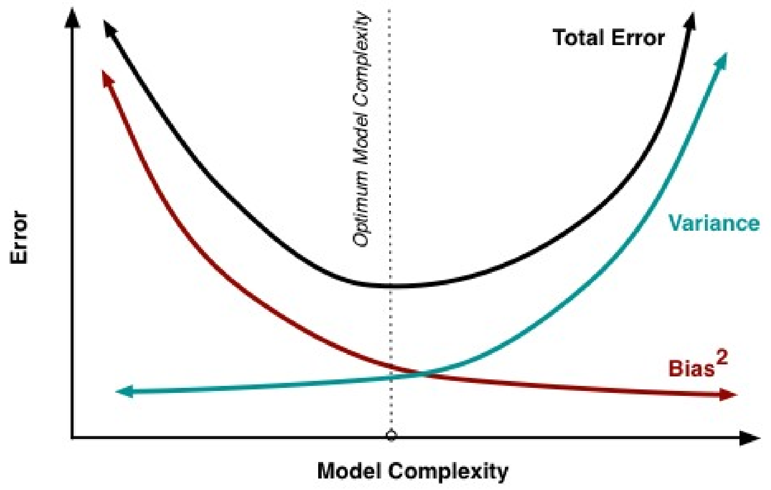
Like every other model, a tree based algorithm also suffers from the plague of bias and variance. Bias means, 'how much on an average are the predicted values different from the actual value.' Variance means, 'how different will the predictions of the model be at the same point if different samples are taken from the same population'.

You build a small tree and you will get a model with low variance and high bias. How do you manage to balance the trade off between bias and variance ?

Normally, as you increase the complexity of your model, you will see a reduction in prediction error due to lower bias in the model. As you continue to make your model more complex, you end up over-fitting your model and your model will start suffering from high variance.

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.

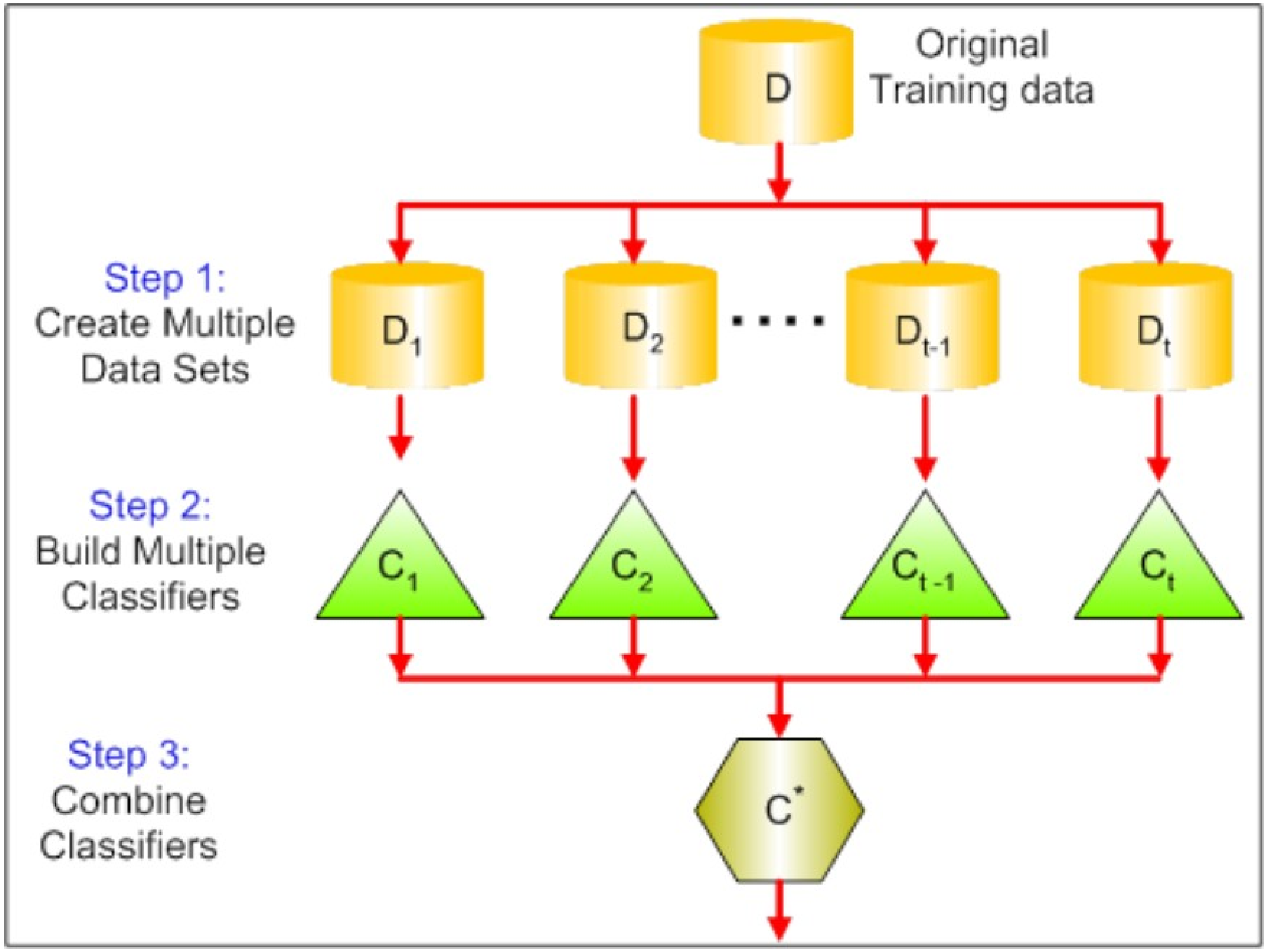
Ensemble Methods



Some of the commonly used ensemble methods include: Bagging, Boosting and Stacking

Bagging | Ensemble

Bagging is an ensemble technique used to reduce the variance of our predictions by combining the result of multiple classifiers modeled on different sub-samples of the same data set. The following figure will make it clearer:


D
Original Training data
Step 1: Create Multiple Data Sets
D_1
D_2
D_{t-1}
D_t
Step 2: Build Multiple Classifiers
C_1
C_2
C_{t-1}
C_t
Step 3: Combine Classifiers
C^*

Bagging | Ensemble

The steps followed in bagging are:

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 modeled 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.

Note that, here the number of models built is not a hyper-parameters. Higher number of models are always better or may give similar performance than lower numbers. It can be theoretically shown that the variance of the combined predictions are reduced to 1/n (n: number of classifiers) of the original variance, under some assumptions.

There are various implementations of bagging models. Random forest is one of them and we'll discuss it next.

Random Forest | Bagging | Ensemble

Random Forest is considered to be a *panacea* of all data science problems. On a funny note, when you can't think of any algorithm (irrespective of situation), use random forest!

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.

Remember the quality of your inputs decide the quality of your output. So, once you have got your business hypothesis ready, it makes sense to spend lot of time and efforts here. With my personal estimate, data exploration, cleaning and preparation can take up to 70% of your total project time.

Below are the steps involved to understand, clean and prepare your data for building your predictive model:

1. Variable Identification
2. Univariate Analysis
3. Bi-variate Analysis
4. Missing values treatment
5. Outlier treatment
6. Variable transformation
7. Variable creation

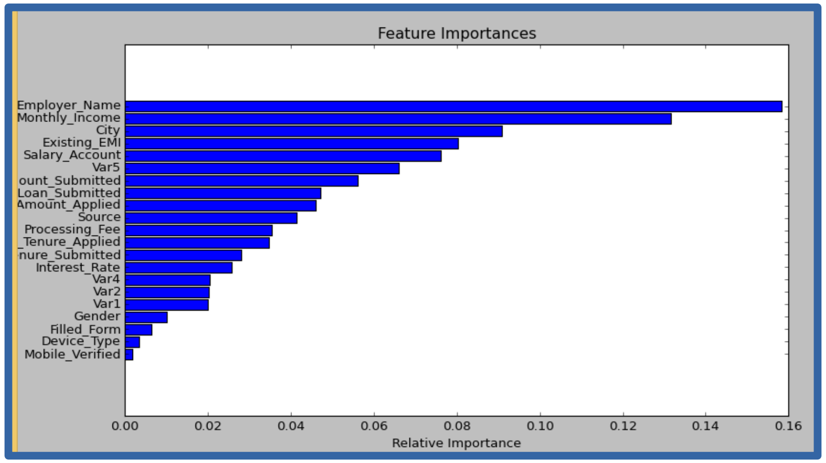
** analyticsvidhya.com/blog/2016/01/guide-data-exploration/

How Random Forest Works

In Random Forest, we grow multiple trees as opposed to a single tree in CART model (see comparison between CART and Random Forest here, part1 and part2). 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.

1. 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.
2. 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.
3. Each tree is grown to the largest extent possible and there is no pruning.
4. Predict new data by aggregating the predictions of the ntree trees (i.e., majority votes for classification, average for regression).

Advantages of Random Forest



- 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 which excites me most 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.
- It has methods for balancing errors in data sets where classes are imbalanced.
- The capabilities of the above can be extended to unlabeled 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.

Disadvantages of Random Forest

- 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 modelers – you have very little control on what the model does. You can at best – try different parameters and random seeds!

R Code

```
> library(randomForest)

> x <- cbind(x_train,y_train)

# Fitting model

> fit <- randomForest(Species ~ ., x,ntree=500)

> summary(fit)

#Predict Output

> predicted= predict(fit,x_test)
```

Boosting | Ensemble

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

Let's understand this definition in detail by solving a problem of spam email identification:

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

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

Above, we've defined multiple rules to classify an email into 'spam' or 'not spam'. But, do you think these rules individually are strong enough to successfully classify an email? No.

Individually, these rules are not powerful enough to classify an email into 'spam' or 'not spam'. Therefore, these rules are called as **weak learner**.

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

- 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'.

How Boosting Works

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

Here's another question which might haunt you, '*How do we choose different distribution for each round?*'

For choosing the right distribution, here are the following steps:

Step 1: The base learner takes all the distributions and assign equal weight or attention to each observation.

Step 2: If there is any prediction error caused by first base learning algorithm, then we pay higher attention to observations having prediction error. Then, we apply the next base learning algorithm.

Step 3: Iterate Step 2 till the limit of base learning algorithm is reached or higher accuracy is achieved.

Finally, it combines the outputs from weak learner and creates a strong learner which eventually improves the prediction power of the model. Boosting pays higher focus on examples which are mis-classified or have higher errors by preceding weak rules.
There are many boosting algorithms which impart additional boost to model's accuracy. In this tutorial, we'll learn about the two most commonly used algorithms i.e. Gradient Boosting (GBM) and XGboost.