Practical Data Science

LVC 2: Introduction to Bagging and Random Forest

In the last session, we discussed decision trees. **Decision Trees** are a supervised learning method used for classification (spam/not spam) and regression (pricing a car or a house). Decision trees usually work top-down, by choosing a variable at each step that **best splits** the set of items. The end nodes can have a category (classification) or a continuous number (regression).

But the drawback of decision trees is that the learning mechanism in decision trees is very sensitive to even small changes in data. Also, larger decision trees generally tend to overfit on the training data.

Why do decision trees tend to overfit?

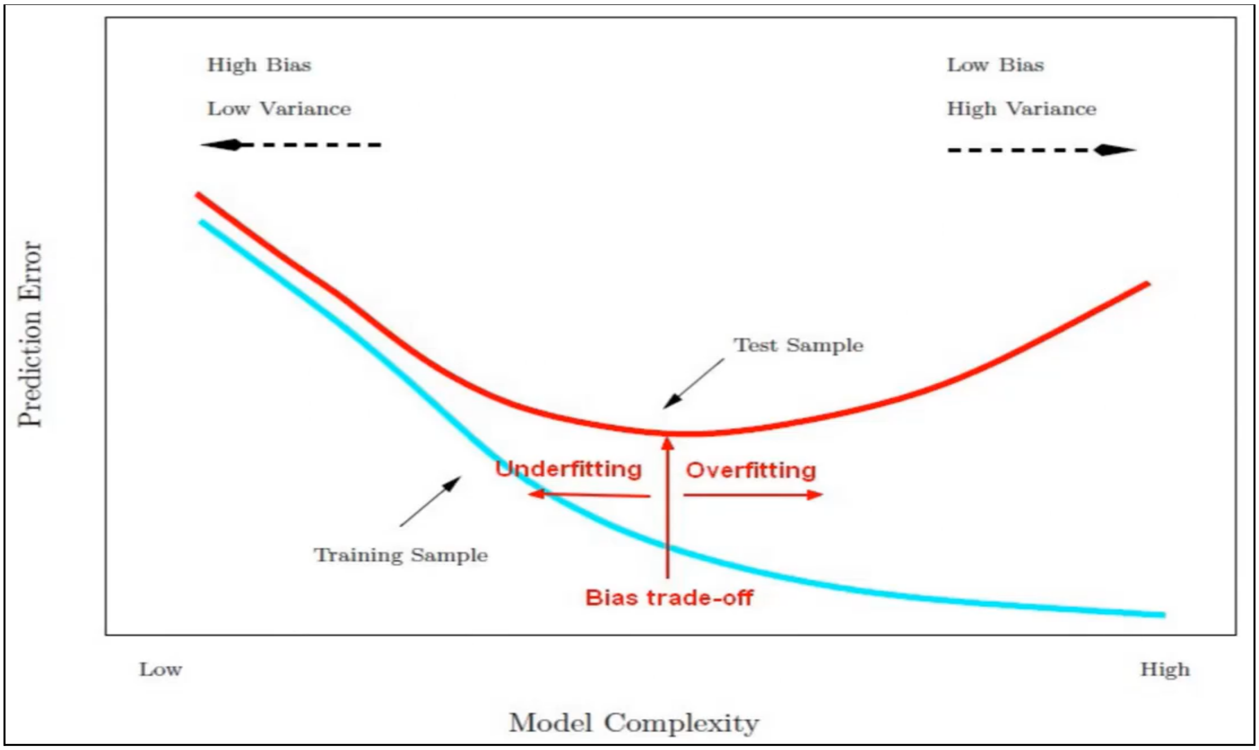
Before answering this question, let's understand the terms bias and variance.

Bias is the difference between the prediction of our model and the correct value that we are trying to predict. A model with high bias gives less attention to the training data and makes the model too general, which leads to a high error in both the training and test datasets.

Variance is the quantity that tells us about the spread of our data. A model with high variance pays a lot of attention to the training data and does not generalize well to the test data. Therefore, such models perform well on the training data but have a high error on the test data.

Now, back to our question, why do decision trees tend to overfit?

At each node, decision trees will make the decision after computing all attributes, and the path from the first node to a leaf represents **decision rules.** The deeper the tree, the more complex these rules will be.



Overfitting refers to the condition when the model completely fits the training data but fails to generalize to the unseen testing data. So, if a decision tree is fully grown, then it may lose some generalization capability. This results in **high variance**. But, decision trees have a **low bias** because they maximally overfit the training data.

The above figure shows the prediction error as a function of model complexity. On the left side, it is observed that, where both training and testing errors are very high, is the region of **high bias**, whereas, on the right side, the testing error is high, but the training error is low, which is the region of **high variance**.

The reason why a **bias-variance trade-off** exists is that an algorithm can't be more complex and less complex at the same time. If our model is too simple, then it may have a high bias and a low variance. On the other hand, if our model is too complex, then it's going to have a high variance and a low bias. So, we need to find a good balance without overfitting and underfitting the data.

How do we reduce overfitting in decision trees?

There are various ways to prevent the decision tree model from overfitting. Here, we discuss the three important techniques/algorithms to reduce overfitting in decision trees. They are

1. Pruning
2. Bagging
3. Random Forest

1. Pruning

The idea behind pruning is to let the decision tree grow fully and then remove the non-significant branches to reduce the complexity.

Pruning Algorithm:

- Create a decision tree with the maximum depth
    - Until every leaf belongs to the same class (homogeneous)
- Run iteratively until the tree reaches the optimal depth
    - Pick a subtree (a node and all leaves)
    - Aggregate the leaves all the way to the node
    - Compute new error
        - Misclassification
        - Entropy
    - Remove/keep subtrees based on the performance score

The pruning algorithm starts at the leaf node in a decision tree. Following recursively upwards, it determines the significance of each subtree using a performance metric. If the subtree is not significant, it can be replaced by a leaf and assigned the most common class at that node.

Pruning Downsides:

- **Loss of Information**: Since we throw away some information by removing subtrees, as a result, some accuracy may be lost.
- **Expensive**: Because while pruning, many sub-trees must be formed and compared.

Now, the question arises: **How about directly building a better model** instead of pruning a fully grown tree? There are two algorithms that can help with this: bagging, and random forest.

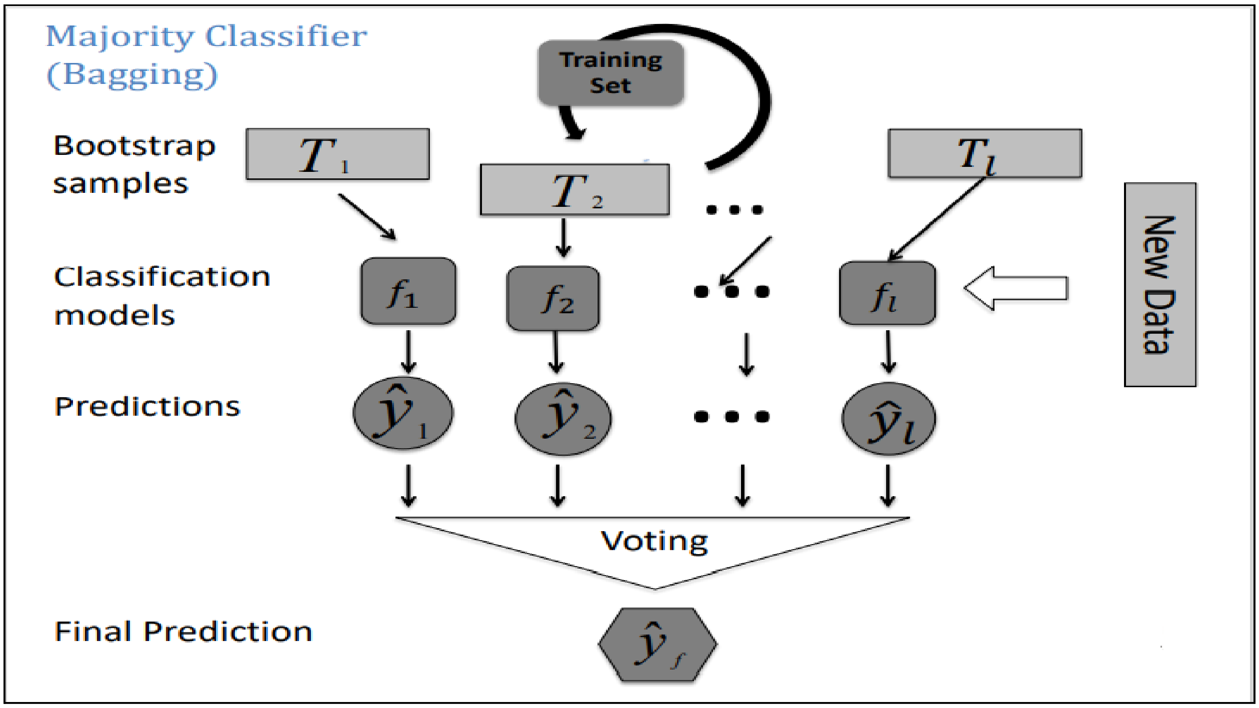
2. Bagging

Ensemble learning is the process of **combining predictions** from multiple machine learning models (base models). Bagging is an example of an ensemble algorithm composed of two parts: **Bootstrapping** and **aggregation.**

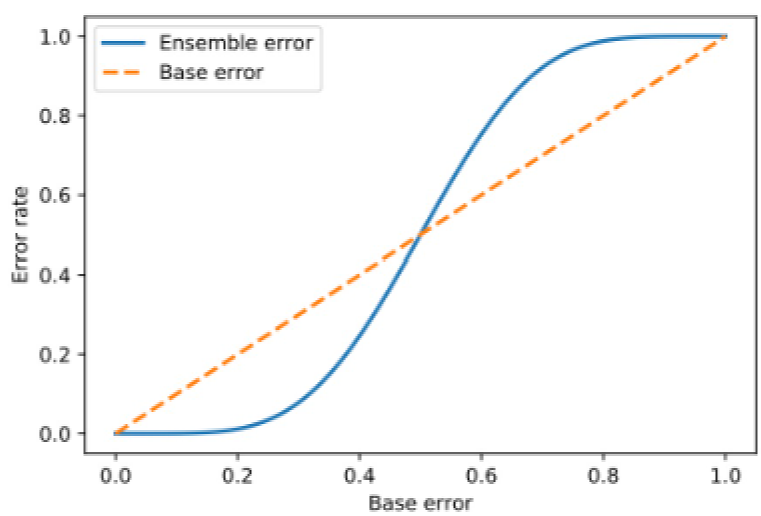
Bootstrapping creates several subsets of the original dataset chosen randomly **with replacement**. Each subset has an equal number of observations and can be used to train models in parallel. By sampling with replacement, some observations may be repeated in each new training dataset.

Data not sampled is used for cross-validation, as shown in the below figure. The probability that an observation is not sampled is approximately 0.368. Details are shown in **Appendix 1**.

Models are trained on each of these training sets independently and the results are **aggregated** for the final prediction. The final prediction is decided from these models with the **most votes** (mode) in a classification setting, as shown in the below figure. By aggregating the results from the multiple classifiers, bagging can **reduce the variance** of classification. In Regression, the final prediction is an **average** of all the predictions.



Each of these independent classifiers will make misclassifications. But, the probability of the majority of classifiers making a mistake is much **lower** than the probability of one of them making a mistake. A sample calculation is shown in **Appendix 2.** The below plot shows that if the error rate of each classifier is less than 0.5, the ensemble error will be lower than the error of a single model.



Bagging Summary:

- Bagging = Bootstrap + Aggregation
- Bootstrap: sampling with replacement
- Build a classifier with each sample
- Aggregate the results

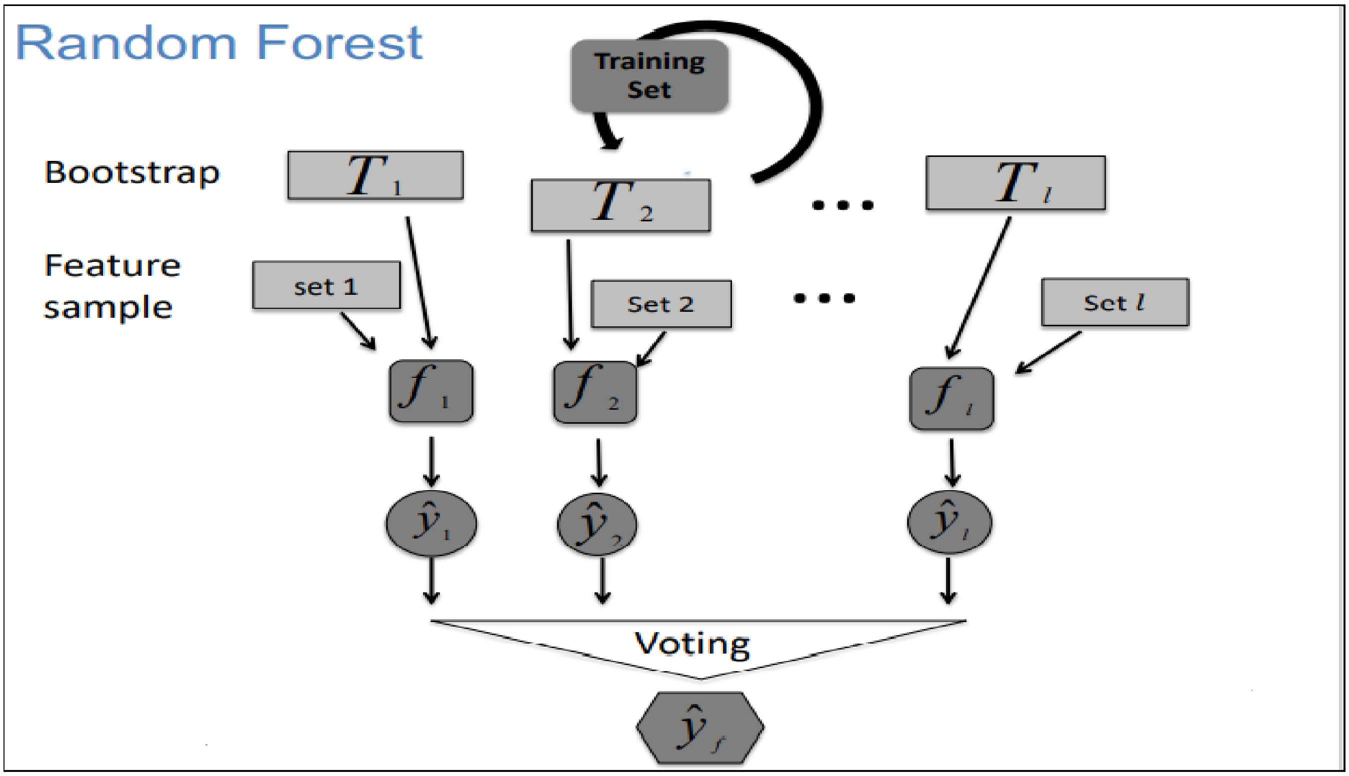
Note: When the total number of samples in the original dataset is very small and the size of each bootstrap dataset is roughly the same size as 'n', the samples are not independent or, in other words, strongly correlated. As a result, the outcome of these classifiers is also strongly correlated. So, we do not have a guarantee that the voting among dependent classifiers reduces the error.

Now, the question arises: **What if the samples are not independent? Is there a way to increase independence?** Yes. There is a way to increase independence using the Random Forest algorithm. Let's see the algorithm now.

3. Random Forest

Random forest is an **extension of bagging**. It can be used for classification as well as regression problems. It builds multiple decision trees and combines them to get a more accurate and stable prediction. Decision trees are very sensitive to even small changes in data, on the other hand, random forest prevents this by allowing a whole bunch of decision trees to work together to get a better and more robust prediction.

The bagging algorithm considers only, '**row (observations) sampling with replacement'.** That is, all training datasets are going to use all the features to decide the split at each node throughout each model. But the Random forest algorithm also considers **'column (features) sampling'** as shown in the below figure.



Instead of splitting using all the features at each node, each split in a tree can be decided based on different features. The features considered for partitioning at each node are a **random subset of the original set of features**.

Thus, the random forest algorithm increases the independence by sampling the features at each node of all trees. Both row and column sampling give a very diverse forest. Though we allow all the decision trees to grow larger, the final results have a lesser variance than a single decision tree model. While individual trees tend to overfit the training data, averaging corrects this. So, there is no need to prune the fully grown trees in random forests.

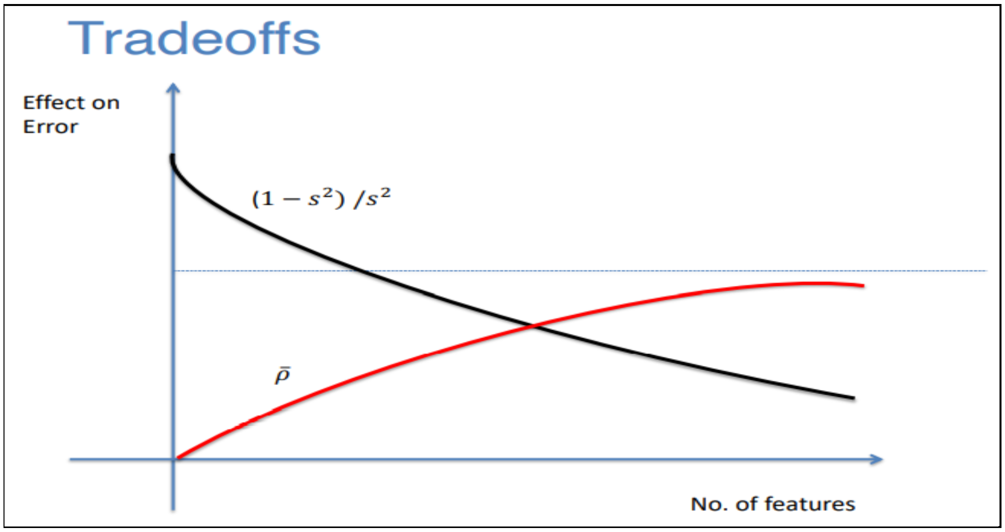
Generalization Error of Random Forests

The generalization error of the random forest classifier depends on the strength of the individual trees in the forest and the correlation between them. It is defined as below,

$$Error \leq \bar{\rho}(1 - S^2)/S^2$$

- $\bar{\rho}$: Correlation between classifiers
- S: Measure of the strength of the classifier (1 - error)

If we use a large proportion of available features at each split, even though the datasets are slightly different, the trees built from each dataset become very **correlated.** If it is very small, then the chance of actually catching one of the important variables in the splitting mechanism becomes lower, and the base classifiers have a very **weak ability** to predict. Therefore, there is a tradeoff between these two terms, correlation, and strength of the classifier as shown in the below figure. The number of features for splitting each node is to be chosen carefully to avoid any correlated or weak trees.



Random Forest summary

Algorithm:

- Random sampling with replacement
- For each subset, build a decision tree. However, only use a subset of randomly picked independent variables for each node's branching possibilities
    - Do not prune
- While predicting:
    - Use each tree to make individual predictions
    - Combine predictions using voting:
        - Mean for regression
        - Mode for classification

Benefit: More diverse, better generalization

Downside: Less interpretable

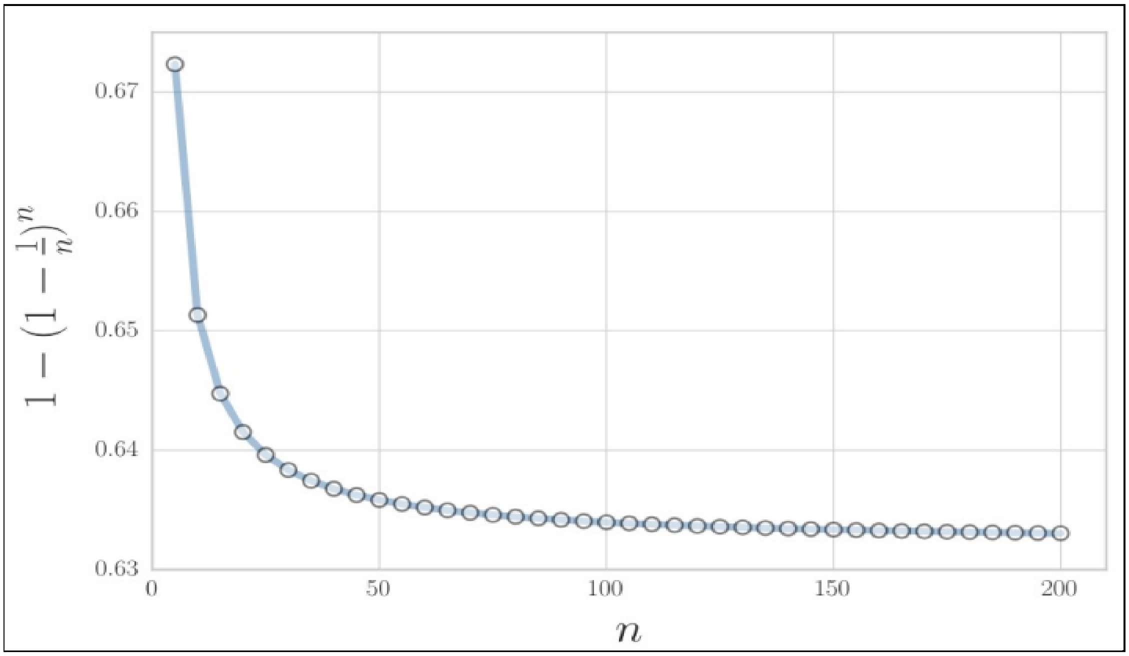
Appendix - 1

Size of Test Sets in Bootstrapping

- Assume that we have ‘n’ data points.
- What is the probability that a specific data point is not selected in ‘n’ samples with replacement?

$$(1 - 1/n)^n$$

- If ‘n’ is large, then this probability is approx 1/e = 0.368.
- Provides a reasonable percentage for cross-validation to estimate error.
- Then, each data point has a probability $1 - (1 - 1/n)^n$ of being selected as training data.



Appendix - 2

Analysis of majority voting

Let's assume that we created multiple datasets using bootstrapping and built the classifiers for each of these datasets. The results can be combined from these classifiers for final predictions.

Each of these independent classifiers will make misclassifications. But, the probability of the majority of classifiers making a mistake is much **lower** than the probability of one of them making a mistake. Let's do the analysis in a trivial way to get some intuition about how the error is reduced.

The majority of voting is defined mathematically as follows,

$$\{\hat{y}_f\} = f\ (x)\ \equiv majority\ (f_1(x), (f_2(x), \dots\ (f_l(x))$$

The probability that the majority of classifiers make a wrong prediction is defined as follows,

$$P\left(f\ (x) \neq\ \hat{y}_f\right) = \sum_{\{k \geq \frac{l}{2}\}}^{l} \binom{l}{k} \epsilon^k\ (1-\epsilon)^{l-k}$$

- $\hat{y}$ - Final prediction
- $f_i(x)$ - Classification models of each independent bootstrap training sample
- l - Total number of bootstrap training sets
- ϵ - Uniform error rate of each classifier

Let us assume,

- There are 25 bootstrap training sets, hence 25 classifiers
- Each classifier has error rate of ε = 0.35
- Errors made by classifiers are uncorrelated

Then, the probability that the majority of classifiers make a wrong prediction is,

$$= \sum_{k=13}^{25} \binom{25}{k} \epsilon^k\ (1-\epsilon)^{25-k}$$

= 0.06, which is better than 0.35.