**Decision Trees and Ensemble Learning**

**Introduction**

So far, we have looked at two machine learning algorithms, I.e., **Logistic Regression and Linear Regression**. But machine learning is not just about these two models. We have just scratched the surface of the number of algorithms that exists in Machine Learning.

*From this chapter onwards, we will build our machine learning toolbox further and study tree-based algorithms.*

Tree based algorithms are some of the most easily understood and most powerful algorithms.

They are **very easy to interpret and explain to the stakeholders**. They just ask bunch of questions about the data and split it as much as possible.

We will be looking at Decision Trees which is a way of building ML models by asking **if-else-then type questions**. They can be further combined to form much powerful models. There are two ways in which these two can be combined. One way will lead us to Random Forest algorithm while the other will lead us to Gradient Boosting Algorithm. We will be looking at both in this chapter.

Combination of Random Forest can be thought of as a way of averaging the predictions of multiple decision trees while that of gradient boosting is that of boosting every next tree iteratively rather than working at decision trees independently (It works at it in a dependent manner).

If this discussion does not fully make sense to you, just skip this part because we will be looking at all of these in much more detail in this chapter.

I hope you are ready to sit under the shade of tree. So, let us get started.

**Objective of this chapter**

In this chapter, we will be looking at the explanation and implementation of tree-based algorithms including decision trees, random forests and gradient boosting.

Before we proceed further, open [**this**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD)link. This link will open an **online hosted Jupyter Lab environment that** can be used to run the codes discussed in this lab and try the exercises. You might have to wait a few minutes for it to fully load. This will have the labs, exercises, and project notebooks of all the chapters in this book.

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

We will do a guided project in this chapter where the objective will be to predict whether a bank’s existing customer (for health insurance) will be interested in purchasing vehicle insurance from us.

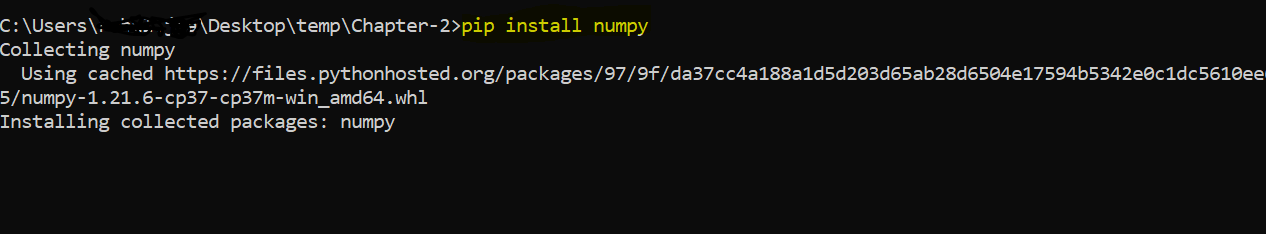
Kaggle Data Link:

**https://www.kaggle.com/datasets/anmolkumar/health-insurance-cross-sell-prediction?select=train.csv**

GitHub Data Link: **https://raw.githubusercontent.com/fenago/MLWorkshop/main/Chapter-6/CrossSell.csv**

We will be using the Python programming language to do this guided project and the libraries you will need are “Numpy”, “Skearn” , “Pandas”, XGboost Etc. **You don’t need to install anything as we will be providing you with a hosted notebook on binder which you can run to execute codes as we move further into this chapter**.

But if you are working on your local system, then make sure that you have these packages installed on your system. If it is not already installed, open a command prompt and enter the following command to install numpy:



In an equivalent way, you can install other packages used in this chapter.

**Business Understanding**

Let us assume that **you are working as a data scientist** for an insurance company that has provided Health Insurance to its customers now they need your help in **building a model to predict whether the policyholders (customers) from past year will also be interested in Vehicle Insurance provided by the insurance company**.

This model will help the company to deepen the relationship with their customers and at the same time increase their book value both in terms of revenue and portfolio.

This will further help company build its **communication strategy** to reach out to those customers differently. So, two different kinds of communication can be sent to two categories of customers:

1. **Those who might purchase vehicle insurance vs**
2. **Those who might not**

It will **h**ave an impact on their click through rate of their communication and campaign initiatives and **optimize business model** and its **revenue**.

Now, to predict, whether the customer would be interested in Vehicle insurance, we have information about **demographics (gender, age, region code type)**, Vehicles **(Vehicle Age, Damage)**, Policy **(Premium, sourcing channel)** etc.

The idea is that these variables would help us to assign a probability against each customer for the likelihood of them purchasing the insurance.

**Decision Tree Algorithm**

Before formally defining decision tree algorithm, let us ask ourselves a question about what is the purpose of any ML algorithm?

The answer to this question will differ from a specific ML task. If I have a classification problem, I want a model (algorithm) which will be able to classify a given input correctly.

We can reframe this statement. We want to classify a given observation to the same class of a group with **which its characteristics match the most**.

So, if I have a group of Data Scientists from two countries **India and United States** and I want to build a model to predict their salaries, then for a given observation (Individual) from India, I want my model to predict their salary to be close to what Indian group salary is not of United States.

So, it can also be called as the profile or cluster of our observations, and we want to make this cluster as precise as possible. Although there is a technique to do the same called **K-means clustering** but that will be the topic of another discussion.

**Let us come back to the decision tree algorithm.**

*Decision tree algorithm splits our data based on features and the objective of that splitting is to make the splits more similar within themselves and different among them.* We have two jargon to refer to these two kinds of situations. We call these as homogenous within and heterogenous from other splits.

**Let us take an example to illustrate this.**

Suppose I have rows and I split those into 7 and 3 rows separately (assuming with the help of some appropriate algorithm).

Now, after splitting they should be same within (Homogeneous within the group) and different when compared with other splits (Heterogeneous among the group).

Now, if I look at 7 data points, they should be similar in some way while the 3 rows should be different.

Now, how does decision tree make sure that this similarity is maintained? Let us discuss the algorithm in little bit more detail.

**Decision Tree Algorithm**

Every column (or feature) is a potential which can be used to split our data in two parts. **But which column?**

Because in real world, we have many features, so which one to choose? Okay, let us assume that we know how to figure out the best column to split our data on, the next question arises how to actually use it to make a split.

Suppose I have decided that Age should be the column that can be used to make a split, We can take any one age value **(call it z)** and split our whole data by asking one question., **Is age >= z or age < z**? All the rows of our data which correspond to **age >= z will go in the first split** and other goes into another split.

What about a categorical variable? For that, we can just ask if a certain value of categorical variable is present.

For example, we can have **Gender as the potential column** to split our data on. Then, we can say if the Gender is “Male” then call that first split and all the other genders will go into the second split.

So, there are two important questions we have now:

1. How to decide which column to split our data on?
2. Once column is decided, which value of the column to split our data on?

**Impurity of the data**

Any dataset can have **one numerical value for impurity**.

With this value, **we can compare two datasets**. If one has large impurity, **then that data is not desirable.**

Let us first understand the impurity from an **intuitive perspective**, then we will talk about some formal methods available out there to compute impurity.

**Intuitive understanding of Impurity**

Consider a classification task and consider I have two sets of data:

Table

Description automatically generated

**Now, let’s do a thought experiment**.

If I pick up a random observation from **set.1** and ask you to tell me if this **observation belongs to spam or no-spam**, then what will be your answer? The answer would be not sure as we are not exactly sure which class has majority of the observations.

But on the other hand, if I randomly pick an observation from set.2, them most likely (80% of the time), I will pick up a spam mail.

**So, Now, we can frame a technical question.**

Which set is purer here for a classification task? The answer is **set.2 is pure**. There can be some other set which is even purer than set.2 but out of these two sets, set2 is purer.

Two questions might arise here:

* What about regression task?
* Is there a way to compute a numerical score to quantify this pureness?

For regression tasks, instead of a class, **we will have a continuous target**.

How could we measure which set of data is purer for this task. We could look at variance of each set for the target and see which is low and that will indicate to us if one set is purer than the other.

Answer the second question, **yes there are many techniques to compute the measure of impurity of a dataset**. Next, we will look at some of these methods.

**Impurity for a classification problem**

There are two major techniques for computing the impurity of a dataset, we will look at both of these below:

**Entropy**

Let us assume we have a classification problem where we have c number of unique class. Then, for any dataset, we can compute its entropy by first calculating the probabilities of each class in that data and

Where is the probability of the ith class. We can think of negative sign as a way of converting the purity measure to an impurity one. So, the smaller the impurity is, the better it will be for a classification task.

**Its value can range from 0 to 1.**

**Gini Impurity**

Gini is one more method to compute the measure of impurity for a given data. It can be computed as:

Both and entropy calculates the same thing. The only difference between these two is the difference of scale.

**The**  **ranges from 0 to 0.5. Other than that, its functions are also similar to entropy.**

Also, can be thought of as computationally more suitable than entropy as there are no log calculations.

Now, that we have discussed the impurity, let us return to the following two questions:

1. **How to decide which column to split our data on?**
2. **Once column is decided, which value of the column to split our data on?**

Every split of data will result in two subsets of split, we can call them **left split** and **right split** for the sake of understanding. Both of them can have their own impurity measure as we discussed earlier.

For example, if I decide to split on **age >= 15** then, I will have a left split which will contain **all those rows which do have age >= 15** and right split which have all those **rows which have age < 15.**

**Information Gain**

Information gain is the **decrease in the impurity after splitting**.

We want to have that split which has the largest information gain.

Suppose we had a choice between splitting our data on two values of age which are 20 and 25. Which one do we choose?

**We will choose the one which has the largest information gain.**

After every split, we have **one dataset (call it node)** and then after split, we get two data, call them  **and** .

Let us further assume that **top level data has**  observations while the left one has  **and right one has** observations. Also, consider that the data at the node has an impurity measure of and left and right datasets have impurities and , then information gain for this split can be defined as:

So, what this formula is doing is calculating the **difference between the impurity at the top (node) and weighted impurities of the data after split**.

This information gain must be computed for all the candidates split and the one which has the maximum information gain will be chosen as a winner.

Now, let us come back to the decision tree algorithm. We defined decision tree like this:

**Decision tree algorithm splits our data based on features and the objective of that splitting is to make the splits more similar within themselves and different among them.**

So, given a dataset of **n rows and m columns**, decision tree attempts to split this data into **two left and right datasets** based on the maximum information gain out of all the **candidate splits**.

**How will candidate splits be formed?**

We will consider each column and each possible value of that columns. So, if I have m columns and let us assume that each column can take only 10 distinct values, so we will compute **information gains for possible splits** and choose the one with the **maximum information gain**.

There can be two types of columns:

* **Categorical**: For this, we will just evaluate all possible values of categorical data
* **Numeric**: Since this can have infinitely many unique values, for these can consider 10 values based on the 10 quantiles of this variable and consider the split on these

**Is that it? Is it what decision tree algorithm all about?**

Not really, what we have discussed so far is **called a decision tree of depth 1**. We did the split only once.

We can go **even further, and do the same steps for the resulting left and right data**. So, the decision tree is a recursive process where we can split the results even further.

**So, how long should we continue like this or what will be the stopping criteria?**

Technically, **we can go as deep as the data allows**. Hence, we can have a situation where we are left with only one observation at the final node (Leaf Node).

It depends on problem to problem. Some problem requires us to go for up to a depth of 10 or some even 5. It also depends on whether there is any data point left to split.

So, we must specify beforehand, what will be the stopping criteria, we can do that by specifying:

* Min number of samples in the leaf node Etc.

These are explained later in hyper parameter section.

**How do we make the prediction for a new observation?**

Once the decision tree has trained, we keep track of all the feature and their values at which split was made, and follow the same for the new observation and traverse the path through the decision tree and then once we reach the leaf node, we have two choices:

* For the classification problem, **make the prediction using the majority vote**
* For the regression problem, take the average of the targets in the leaf node for the training set.

**Let us discuss one point about overfitting and prediction.**

If we continue the process of split while training, then we might run into a situation of overfit where the leaf nodes (we call the nodes at the bottom as the leaf nodes) only contain one observation and it will not generalize well.

It won’t generalise well because if we make prediction, we are only basing our prediction on one observation which might be biased and inaccurate.

**Hyperparameters of Decision Tree Algorithm**

Hyperparameters are those values **(nobs or tweaks)** which we can make to a learning algorithm. There are no right set of values (nobs) for these hyperparameters. For getting the desired combination of hyperparameters, we have to experiment with some values first and see which results in best performance.

Some important hyperparameters for the decision tree algorithm are discussed below:

* **Max Depth**

As we have discussed, this parameter control how deep a decision tree should be and it will decide if the tree will overfit the data or not. A large maximum depth might overfit data but at the same time, it will lead to more complex models.

* **Minimum number of samples at leaf node**

It controls for the number of samples required to make a prediction. We have seen that we can technically fit a tree which has only one observation in the leaf node but that would not generalize well. So, to make sure that tree generalizes well, we can set this parameter for the minimum number of samples that should be there in a leaf node. If it is less than that, we will not be making any splits.

* **Impurity measure to use**

This can have two types of values. It can either be gini or entropy. Both give almost same results, but computationally one can be superior to the other.

**Exercise 6.1 Implementation of Decision Tree using Sklearn**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 6.1 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 6) and do the following:

1. Read the Cross Sell data using pandas:



1. Check how many records and how many columns are there using the shape command:



1. Check the top few rows using the head method of the pandas dataframe:



1. Let us check if there are any columns with missing values:



So, we can see that we don’t have any columns with missing values. Now, we can get into the correlation analysis between the feature and the target. If we had some missing data, we would have to think of some way to impute those.

1. Let us start with finding whether the data is imbalanced or not:



So, we can see that we have a situation where the target is not balance. We must keep this mind while making a train test split and also while evaluating our model. This means that we can not use accuracy as a metric for evaluating our model.

1. The problem is a binary classification problem. The objective is to predict which of the customer should be pitched for a insurance product. Let us check the response distribution for the vehicle age variable:



We can clearly see a significant difference between various vehicle age values. The new vehicles which are less than 1 years of age have an incredibly low percentage of cross sell response. This insight can be used for creating triggers for customers.

1. Now, let us look at same for gender:



We can see that male as a gender is more likely to take this insurance as a cros sell.

1. Let us now prepare the data for modelling. For this, we can use the train\_test\_split function from the sklearn library.

We first create some variables to store the list of column names:



Now, we can use this to store the data in X and y Matrix and create the splits, note that we want to have same target distribution for both train and test, so we will pass the stratify argument.



Now, we have prepared the data. But note that, we also have a categorical variable which needs to be transformed into numerical type. We can do this and modelling separately but that would be quite cumbersome. Because then we will need to do the same steps for the testing data and it gets hard to manage.

That is where pipeline comes into picture. We can create pipeline and wrap both transformers and the modelling steps in one pipeline. We can just call its fit method and it will automatically transform the feature matric and then fit the model.

1. Let us import some helper functions to do that:



1. Now, let us create a column transformer which will process our data frame and will create a feature matrix for us:



We are passing a list of tuples in this object where each tuple has:

* 1. Name of the transformer
  2. Actual transformer object
  3. Columns on which it needs to be applied.

Note the third tuple which has a “passthrough” transformer which means on these columns (others), no transformations will be applied,

1. Finally, let us wrap everything in a pipeline and train our decision tree classifier and evaluate its performance using ROC AUC score:



We can see that train and test AUC are above 0.8 which means models are good but still there is some scope of improvement. We can increase it further and also we look at decreasing the gap between train and test performance.

Please note the use of make pipeline function. In this function, we passed our transformer and also, we passed the decision tree classifier with one hyperparameter set of our choice.

**Random Forest Algorithm**

Before starting our discussion on random forest algorithm, let us first discuss ensemble methods in general.

**Ensemble Method**

Often, it has been seen from online data science competitions that, many models when combined intelligently can produce really good results than if they had been used alone.

Ensemble refers to the process of combining several base models in order to produce one powerful model.

There are several methods to combine base models. Random forest and Gradient boosting are two of the extremely popular methods to do ensemble which produce accurate models.

* **Bagging**: Bagging refers to Bootstrap aggregation which means to sample your data and use repetition to create many copies of your data. This way, we can reduce the model variance. Random Forest can be considered as an example of Bagging. Here, each model will be created independently of the other model and their predictions at last will be aggregated.
* **Boosting**: Boosting refers to a method in which multiple base learners are fit and every next model will be dependent on the previous model. The weights of the model will be adjusted according to the predictions made by the previous model.

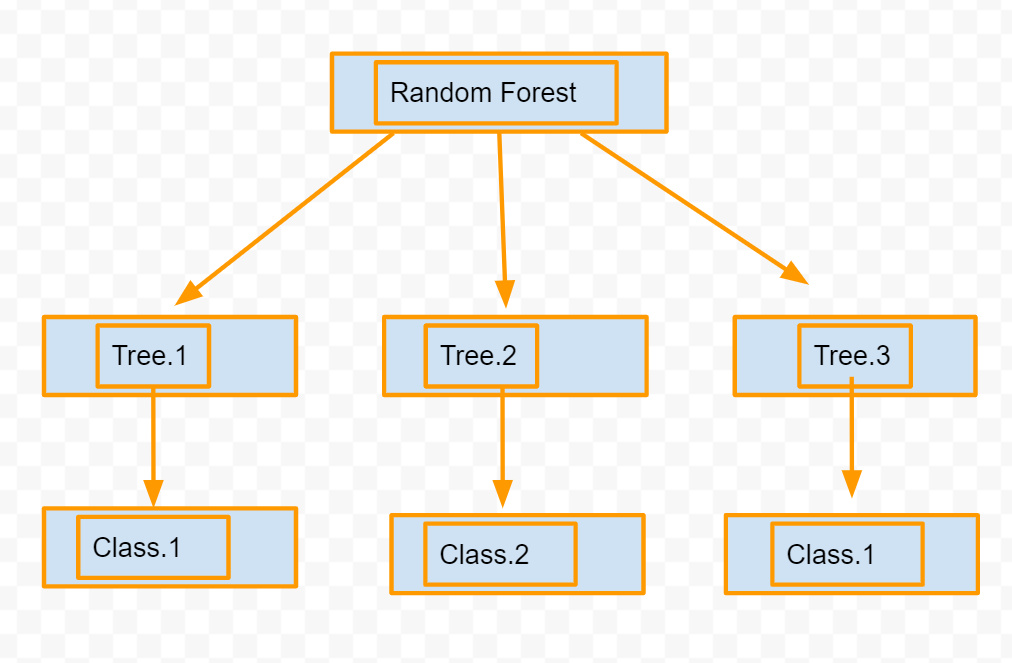
**Let us discuss Random Forest algorithm in detail.**

**Random Forest Algorithm**

Random Forest algorithm refers to a technique of modelling where there are multiple decision trees (we have already seen what a decision trees is and how do we create one). Hence, the name Forest. We can choose to select how many decision trees need to be created in random forest algorithm.

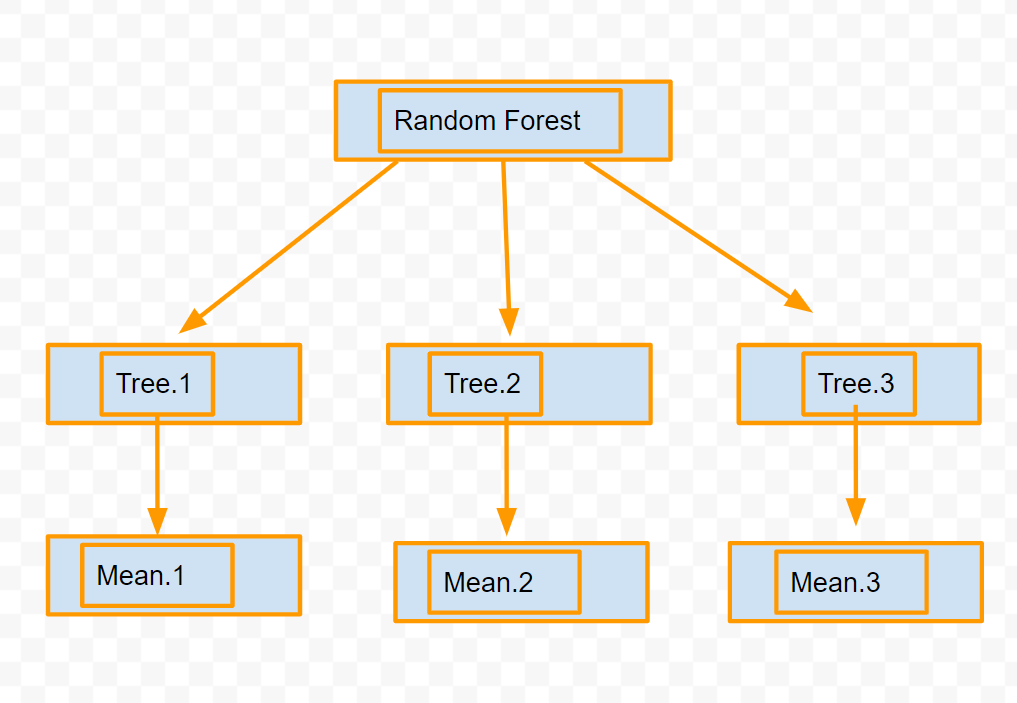
Let us say we decide to create 3 decision trees for a classification problem. How will us make a prediction in that situation? We will look at two situations for making prediction for classification and regression task.

**Classification Prediction**



We can see that the random forest has three trees and each one is making its prediction independent of the other one. For a new observation, we will follow this forest and choose the majority (or decide the prediction based on most votes) which in this case would be Class.1.

**Regression Prediction**



For a regression task, we are going to take the **average prediction of all the resulting trees**.

**How does Random Forest Make sure that each individual tree is a different one?**

The basic objective is to have **different decision trees for each fit otherwise what is the point of having many decision trees**.

We don’t want to have all the decision tree as same, then what will be the point of ensemble then.

So, the question arises that **how we make sure that data provided to each decision tree is different**. We can do that with the help of following two concepts (both of which are related to sampling):

* **Taking a bootstrapped sample instead of the original data**
* **Taking a sample of features, not all the features while making the decision trees.**

Bootstrapped sample is the same original data, but they can have repeated rows (I.e they are sampled from the original data). So, in this way, for each decision tree, we will have a different data on which to train the tree.

**Example of Bootstrapping**

The concept of bootstrapping can be seen from the following image:

Application, table, calendar

Description automatically generated with medium confidence

Here, we have an original data and from that, we create three bootstrapped version of the same data by **taking sample rows (with repetition) from the original data**.

**Sampling Features**

Next, we do not take the entire set of features for each decision tree but rather a subset of these. So, each decision tree see a different combination of features.

Hyperparameters

**Hyperparameters of Random Forest**

Just like decision trees, we also have hyperparameters for the random forest algorithm.

We have the following hyperparameters for the random forest algorithm:

* How many decision trees should be trained?

This hyperparameter control the complexity of the forest algorithm. The more we have number of decision tree, the more accurate the overall model should be.

* How many features to choose in each bootstrapped data?

This parameter controls the differentiation in the data by making sure that a subset of features are selected for each decision tree.

* Then, for each decision tree, we can have those same hyper parameters which we had for the individual decision trees which we saw in the last discussion such as:
  1. Max Depth,
  2. Impurity measure to use,
  3. Number of Samples in the leaf node Etc.

**Exercise 6.2 Implementation of Random Forest Using sklearn**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 6.2 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 6) and do the following:

1. Read the starter code. It will make the necessary splits of data for you.
2. It will also create the transformer objects for you.
3. Import the random forest classifier from the ensemble module:



1. Create a pipeline for the random forest classifier:



1. Now, let us fit and evaluate this pipeline using AUC score:



We can see that auc score is much lower but there is close to no overfit. Let us move on to second ensemble method called gradient boosting machines.

**Gradient Boosting Machines**

Gradient Boosting Machines (GBM) is a second type of ensemble we can do with our base learners.

Earlier, we saw that we can combine the predictions of many decision trees which have been constructed on a bootstrapped sample independently. There is one more paradigm where we can combine many base learners in a dependent manner.

The main difference between random forest and gradient boosting machines is the difference of dependence.

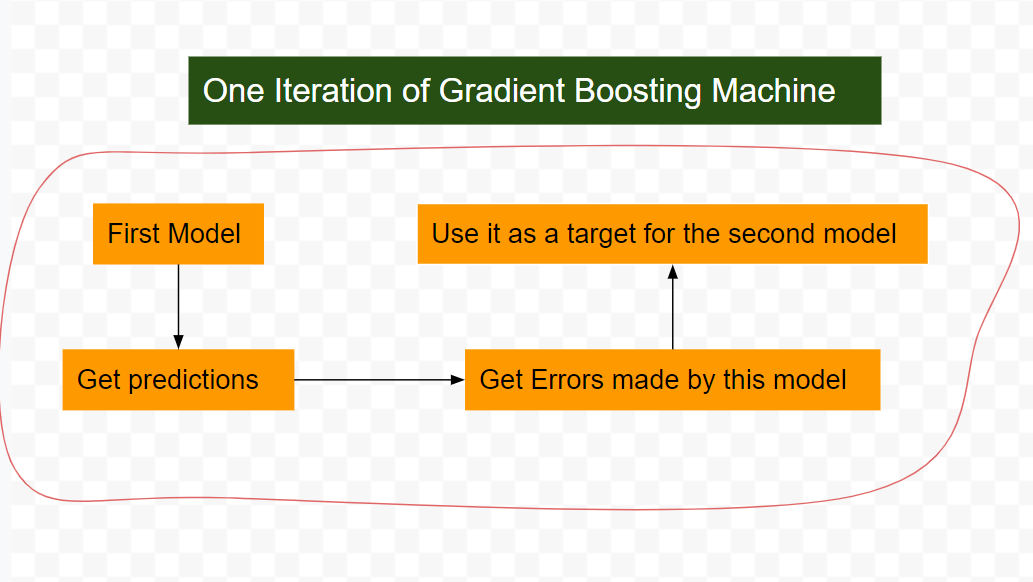
In gradient boosting, we first train our first base learner and then we look at the errors it makes and then in the next iteration, instead of modelling the target, we model the errors and then add these to the predictions made by the first tree. We continue the same thing with the errors of this second tree as well. We can do it recursively until some stopping criteria is not met.

The stopping criteria can be:

* If we have reached the limit of number of trees that should be fit
* Or if we have enabled the early stopping limit

Early stopping limit refers to a way by which we can stop the training further if the metrics are not improving anymore after each iteration.

The following image best captures what gradient boosting does for one iteration.



The basic idea is each successive model has some information about the errors made by the previous model. So, we work on boosting the performance of the previous model.

**But what is ?**

It is an open-source library just like which implements the Gradient Boosting Machine algorithm. What is the difference between implementation and implementation then? The only difference is of speed. The xgboost is fast and has capabilities build inside to make GMBs run parallelly, we can see the difference even with the small dataset.

Further, uses some advanced regularization capabilities to make it less prone to overfit.

**Exercise 4.3 Implementation of Gradient Boosting**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 4.3 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 4) and do the following:

1. Read the starter code. It will make the necessary splits of data for you.
2. It will also create the transformer objects for you.
3. Import the random forest classifier from the ensemble module:



1. Create a pipeline for the Gradient Boosting classifier:



1. Now, let us fit and evaluate this pipeline using AUC score:



We can see that GBM classifier has produced the best AUC score so far. It has been able to give us an score of 0.85 which is maximum and at the same time it is maintaining the performance for both train and test, so there is no overfit.

1. Running the same data through classifier. Let us make the pipeline:



It can be done because natively supports integration with pipeline object.

1. Let us train the model.



We can see that the performance is almost same as the GMB but the training was quite fast. It was around 6x faster than the implementation.

**Hyperparameter tuning of Tree Based Models**

In this chapter, we have discussed following three algorithms for creating machine learning models so far:

* Decision Tree
* Random Forest
* Gradient Boosting

While all of them are really powerful but they can overfit very easily any kind of data. So, it is very important to make sure that their hyperparameters are properly set before putting these models into production.

Hyperparameters, as we have already discussed, are nobs of an algorithm which every data scientist has to decide on and find their optimal value. Every machine learning algorithm have a certain unique hyperparameters and it is important that they should be understood to make the best use of them.

**Grid Search**

Grid Search is a way to evaluate all the possible values (that we think can be optimal) for a given ML algorithm. It run the algorithm for all the possible values and then return their accuracy scores (or whatever other metric you may provide, we have already discussed various metrics in chapter 4).

This method has one downside that it may take a large amount of time to try out all the possible hyperparameter and hence we try out only a sample of those.

We will learn to do the Grid Search optimization of hyperparameter in the exercises.

There are some other algorithms which offers a way to not try all the possible hyperparameter such as :

* Bayesian Optimization
* Evolutionary algorithms
* Gradient Based Optimization Etc.

**Exercise 6.4 Hyperparameter tuning for Tree Based Models**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 6.4 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 6) and do the following:

1. Read the following starter code. It will create the necessary data splits and create the transformer for you.
2. Let us tune the parameter for the decision tree using a for loop (which is just like a grid search):



This will evaluate the decision tree at each point of max depth and will print the AUC, we can see that of 8 produces similar performance as gradient boosting.

1. Let us tune the same parameter for Random Forest:



This code will take some time to run, you can fix that by setting a limit on . Again, we can see that with random forest also, we can get close to same results as gradient boosting

It shows us that if we do intelligent hyper parameter tuning than even a simple algorithm can perform just as best as the powerful one.

1. Let us tune one hyperparameter of parameter:



We see that the best value for for is 2 as for large values, it is overfitting.

**Activity 6.1**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Activity 6.1.ipynb file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 6) and do the following. The objective is to build a classification model to predict the forest cover type. Follow the below mentioned steps to do the same:

Data Link: **https://www.kaggle.com/datasets/uciml/forest-cover-type-dataset**

The dataset for the same can be found here: [**Forest Cover data**](https://www.kaggle.com/datasets/uciml/forest-cover-type-dataset)

* Read the data using pandas
* Extract all the numeric columns in a separate variable as a list and all the categorical as a separate variable
* Build a column transform to
  1. Impute the missing values of numeric data using
  2. Scale the numeric variables using
  3. Transform the categorical columns using
* Build a decision tree pipeline with the above column transformer and train the pipeline
* Make predictions using the above pipeline and compute the AUC for both train and test data.
* Tune the and parameter for this pipeline.
* Perform the steps 4 to 6 for Random Forest and Gradient Boosting
* Which model performed best?
* Use in the pipeline to predict the forest cover.
* What was the time difference between Gradient Boosting and ?

**Project**

Link: **https://archive.ics.uci.edu/ml/datasets/Letter+Recognition**

Navigate to the UCI Link [**here**](https://archive.ics.uci.edu/ml/datasets/Letter+Recognition) and build a classification model to recognize a digit. Try to include all the steps and algorithms that we used in this chapter. Build a Random Forest and Gradient Boosting Model on this data.

**Summary**

In this chapter, we looked at several tree-based machine learning models. We started our discussion with decision tree algorithm where we discussed how it can be understood as a data split algorithm where the split is done basis the impurity and information gain.

Then, we looked at two ensemble techniques called random forest and gradient booting, We also saw how one fits many trees independently while one fits them in a dependent manner.

All the while, we kept looking at practical examples of using these algorithms in and .