**CHAPTER 4: Decision Trees and Ensemble Learning**

**Theory**

In the previous chapter, we studied what Unsupervised Learning is, K-Means Clustering, Hierarchical Clustering, and Principal Component Analysis. In this chapter, we will study what is decision tree, what are ensemble learning techniques and gradient boosting.

Let’s start our chapter with decision trees.

**Decision Trees**

The decision tree algorithm is one of the most versatile algorithms in machine learning which can perform both classification and regression analysis. It is very powerful and works great with complex datasets. Apart from that, it is very easy to understand and read. That makes it more popular to use. When coupled with ensemble techniques – which we will learn very soon- it performs even better. As the name suggests, this algorithm works by dividing the whole dataset into a tree-like structure based on some rules and conditions and then gives predictions based on those conditions. Let’s understand the approach to decision tree with a basic scenario. Suppose it’s Friday night and you are not able to decide if you should go out or stay at home. Let the decision tree decide it for you.

**Diagram

Description automatically generated**

Although we may or may not use the decision tree for such decisions, this was a basic example to help you understand how a decision tree makes a decision. So how did it work?

* It selects a root node based on a given condition, e.g. our root node was chosen as time >10 pm.
* Then, the root node was split into child notes based on the given condition. The right child node in the above figure fulfilled the condition, so no more questions were asked.
* The left child node didn’t fulfill the condition, so again it was split based on a new condition.
* This process continues till all the conditions are met or if you have predefined the depth of your tree, e.g. the depth of our tree is 3, and it reached there when all the conditions were exhausted.



Let’s see how the parent nodes and condition is chosen for the splitting to work.

**Decision Tree for Regression**

When performing regression with a decision tree, we try to divide the given values of X into distinct and non-overlapping regions, e.g. for a set of possible values X1, X2,……XP; we will try to divide them into J distinct and non-overlapping regions R1, R2, . . . , RJ. For a given observation falling into the region Rj, the prediction is equal to the mean of the response(y) values for each training observation(x) in the region Rj. The regions R1, R2, . . . , RJ is selected in a way to reduce the following sum of squares of residuals :

Text

Description automatically generated

Where yrj (second term) is the mean of all the response variables in the region ‘j’.

**Recursive Binary Splitting (Greedy Approach)**

As mentioned above, we try to divide the X values into j regions, but it is very expensive in terms of computational time to try to fit every set of X values into j regions. Thus, the decision tree opts for a top-down greedy approach in which nodes are divided into two regions based on the given condition, i.e. not every node will be split but the ones which satisfy the condition are split into two branches. It is called greedy because it does the best split at a given step at that point of time rather than looking for splitting a step for a better tree in upcoming steps. It decides a threshold value (say s) to divide the observations into different regions(j) such that the RSS for Xj>= s and Xj <s is minimum.

Diagram

Description automatically generated with medium confidence

Here for the above equation, j and s are found such that this equation has the minimum value. The regions R1, R2 are selected based on that value of s and j such that the equation above has the minimum value. Similarly, more regions are split out of the regions created above based on some condition with the same logic. This continues until a stopping criterion (predefined) is achieved. Once all the regions are split, the prediction is made based on the mean of observations in that region.

The process mentioned above has a high chance of overfitting the training data as it will be very complex.

**Tree Pruning**

Tree pruning is the method of trimming down a full tree (obtained through the above process) to reduce the complexity and variance in the data. Just as we regularised linear regression, we can also regularise the decision tree model by adding a new term.

Text, letter

Description automatically generated

Where T is the subtree which is a subset of the full tree T0 And α is the non-negative tuning parameter which penalizes the MSE with an increase in tree length. By using cross-validation, such values of α and T are selected for which our model gives the lowest test error rate. This is how the decision tree regression model works. Let’s now see the working algorithm of doing classification using a decision tree. Greedy Algorithm As per the Hands-on machine learning book “the greedy algorithm greedily searches for an optimum split at the top level, then repeats the process at each level. It does not check whether or not the split will lead to the lowest possible impurity several levels down. A greedy algorithm often produces a reasonably good solution, but it is not guaranteed to be the optimal solution.”

**Post-pruning**

Post-pruning, also known as backward pruning, is the process where the decision tree is generated first and then the non-significant branches are removed. A cross-validation set of data is used to check the effect of pruning and tests whether expanding a node will make an improvement or not. If any improvement is there then we continue by expanding that node else if there is a reduction in accuracy then the node not be expanded and should be converted into a leaf node.

**Pre-pruning**

Pre-pruning, also known as forwarding pruning, stops the non-significant branches from generating. It uses a condition to decide when it should terminate splitting of some of the branches prematurely as the tree is generated.

**Classification Trees**

Regression trees are used for quantitative data. In the case of qualitative data or categorical data, we use classification trees. In regression trees, we split the nodes based on RSS criteria, but in classification, it is done using classification error rate, Gini impurity, and entropy. Let’s understand these terms in detail.

There are multiple algorithms that are used by the decision tree to decide the best split for the problem. Like IDE while using Entropy and Information gain. The second one is Gini Impunity.

Let’s first look at the most common and popular out of all them, which is IDE

**IDE**

**Entropy**

Entropy is the measure of randomness in the data. In other words, it gives the impurity present in the dataset.

Bubble chart

Description automatically generated with medium confidence

When we split our nodes into two regions and put different observations in both the regions, the main goal is to reduce the entropy i.e. reduce the randomness in the region and divide our data cleanly than it was in the previous node. If splitting the node doesn’t lead to entropy reduction, we try to split based on a different condition, or we stop. A region is clean (low entropy) when it contains data with the same labels and random if there is a mixture of labels present (high entropy). Let’s suppose there are ‘m’ observations and we need to classify them into categories 1 and 2. Let’s say that category 1 has ‘n’ observations and category 2 has ‘m-n’ observations.

p= n/m and q = m-n/m = 1-p

then, entropy for the given set is:



When all the observations belong to category 1, then p = 1 and all observations belong to category 2, then p =0, in both cases E =0, as there is no randomness in the categories. If half of the observations are in category 1 and another half in category 2, then p =1/2 and q =1/2, and the entropy is maximum, E =1.

Chart, line chart

Description automatically generated

**Information Gain**

Information gain calculates the decrease in entropy after splitting a node. It is the difference between entropies before and after the split. The more information gain, the more entropy is removed.

Text

Description automatically generated

Where T is the parent node before split and X is the split node from T.

A tree that split on basis of entropy and information gain value looks like this:

A picture containing text, clock

Description automatically generated

Now let’s look at the Gini Impurity method to build the decision tree.

**Gini Impurity**

It measures the impurity of the nodes and is calculated as:

A picture containing text

Description automatically generated

Were,

A picture containing text, watch, clock

Description automatically generated

Let’s first understand what Gini is and then I’ll show you how you can calculate the Gini impurity for split and decide the right split. Let’s say we have a node like this

Qr code

Description automatically generated

So, what Gini says is that if we pick two points from a population at random, the pink ones highlighted here, then they must be from the same class. Let’s say we have a completely pure node

Qr code

Description automatically generated

Can you guess what would be the probability that a randomly pricked point will be below the same class? Well, obviously it will be 1 since all the points here belong to the same class. So, no matter which two points you picked, they will always belong to that one class and hence the probability will always be 1 if the node is pure. And that is what we want to achieve using Gini.

Gini ranges from zero to one, as it is a probability and the higher this value, the more will be the purity of the nodes. And of course, a lesser value means lesser pure nodes.

**How to calculate the Gini Impurity for a split**

1. Calculate Gini for sub-nodes using the aforementioned success(p) and failure(q) formulas (p2+q2).
2. Calculate the Gini Impurity for each split node using the weighted Gini score.

**Example:**

Let’s see a small climate dataset and understand how the decision tree algorithm works.

There are 14 instances of golf playing decisions based on outlook, temperature, and wind factors.

Table

Description automatically generated

Gini Index is a metric for classification tasks in CART. It stores the sum of the squared probabilities of each class. We can formulate it as illustrated below.



**Outlook**

Outlook is a nominal feature. It can be sunny. Overcast or rain. We will summarize the final decisions for the outlook feature.

A screenshot of a computer

Description automatically generated with medium confidence

Gini (Outlook = Sunny) 🡺 1-(2/5)2- (3/5)2 = 1 – 0.16 - 0.36 = 0.48

Gini (Outlook = Overcast) 🡺 1-(4/4)2- (0/4)2 = 0

Gini (Outlook = Sunny) 🡺 1-(3/5)2- (2/5)2 = 1 – 0.36 - 0.16 = 0.48

Then, we will calculate the weighted sum of Gini indexes for the outlook feature.

Gini (Outlook) 🡺 (5/14) x 0.48 + (4/14) x 0 + (5/14) x 0.48 = 0.171 + 0 + 0.171 🡺 0.342

**Temperature**

Similarly, the temperature is a nominal feature and it could have 3 different values: Cool, Hot, and Mild. Let’s summarize decisions for the temperature features.

A screenshot of a computer

Description automatically generated with medium confidence

Gini (Temp=Hot) 🡺 1 – (2/4)2 – (2/4)2 = 0.5

Gini (Temp=Cool) 🡺 1 – (3/4)2 – (1/4)2= 1 – 0.5625 – 0.0625 = 0.375

Gini (Temp=Mild) 🡺 1 – (4/6)2 – (2/6)2 = 1 – 0.444 – 0.111 = 0.445

We’ll calculate weighted sum of gini index for temperature feature

Gini (Temp) 🡺 (4/14) x 0.5 + (4/14) x 0.375 + (6/14) x 0.445 = 0.142 + 0.107 + 0.190 🡺 0.439

**Humidity**

Humidity is a binary class feature. It can be high or normal.

Table

Description automatically generated

Gini (Humidity=High) 🡺 1 – (3/7)2 – (4/7)2 = 1 – 0.183 – 0.326 = 0.489

Gini (Humidity=Normal) 🡺 1 – (6/7)2 – (1/7)2 = 1 – 0.734 – 0.02 = 0.244

Weighted sum for humidity feature will be calculated next

Gini (Humidity) 🡺 (7/14) x 0.489 + (7/14) x 0.244 🡺 0.367

**Wind**

The wind is a binary class like humidity. It can be weak and strong.

Table

Description automatically generated

Gini (Wind=Weak) 🡺 1 – (6/8)2 – (2/8)2 = 1 – 0.5625 – 0.062 = 0.375

Gini (Wind=Strong) 🡺 1 – (3/6)2 – (3/6)2 = 1 – 0.25 – 0.25 = 0.5

Gini (Wind) 🡺 (8/14) x 0.375 + (6/14) x 0.5 🡺 0.428

**Time to decide**

We’ve calculated Gini index values for each feature. The winner will be the outlook feature because its cost is the lowest.

Table

Description automatically generated

We’ll put the outlook decision at the top of the tree.

Diagram

Description automatically generated

The first decision would be the outlook feature.

You might realize that the sub dataset in the overcast leaf has only yes decisions. This means that the overcast leaf is over.

Diagram

Description automatically generated

We will apply the same principles to those sub-datasets in the following steps.

Focus on the sub dataset for sunny outlook. We need to find the Gini index scores for temperature. Humidity and wind feature respectively.

Table

Description automatically generated

**Gini of temperature for a sunny outlook**

Table, calendar

Description automatically generated

Gini (Outlook=Sunny and Temp.=Hot) 🡺 1 – (0/2)2 – (2/2)2 = 0

Gini (Outlook=Sunny and Temp.=Cool) 🡺 1 – (1/1)2 – (0/1)2 = 0

Gini (Outlook=Sunny and Temp.=Mild) 🡺 1 – (1/2)2 – (1/2)2 = 1 – 0.25 – 0.25 = 0.5

Gini (Outlook=Sunny and Temp.) 🡺 (2/5)x0 + (1/5)x0 + (2/5)x0.5 = 0.2

**Gini of humidity for sunny outlook**

Table

Description automatically generated

Gini (Outlook=Sunny and Humidity=High) 🡺 1 – (0/3)2 – (3/3)2 = 0

Gini (Outlook=Sunny and Humidity=Normal) 🡺 1 – (2/2)2 – (0/2)2 = 0

Gini (Outlook=Sunny and Humidity) 🡺 (3/5)x0 + (2/5)x0 = 0

**Gini of wind for a sunny outlook**

Table

Description automatically generated

Gini (Outlook=Sunny and Wind=Weak) 🡺 1 – (1/3)2 – (2/3)2 = 0.266

Gini (Outlook=Sunny and Wind=Strong) 🡺 1- (1/2)2 – (1/2)2 = 0.2

Gini (Outlook=Sunny and Wind) 🡺 (3/5)x0.266 + (2/5)x0.2 = 0.466

**Decision for sunny outlook**

We’ve calculated Gini index scores for features when the outlook is sunny. The winner is humidity because it has the lowest value.

Table

Description automatically generated

We’ll put a humidity check at the extension of the sunny outlook.

Diagram

Description automatically generated

Sub datasets for high and normal humidity

As seen, a decision is always no for high humidity and sunny outlook. On the other hand, a decision will always be yes for normal humidity and a sunny outlook. This branch is over.

Diagram

Description automatically generated

Now, we need to focus on the rain outlook.

**Rain outlook**

Table

Description automatically generated

We’ll calculate Gini index scores for temperature, humidity, and wind features when the outlook is rain.

**Gini of temperature for rain outlook**

Table

Description automatically generated

Gini (Outlook=Rain and Temp.=Cool) 🡺 1 – (1/2)2 – (1/2)2 = 0.5

Gini (Outlook=Rain and Temp.=Mild) 🡺 1 – (2/3)2 – (1/3)2 = 0.444

Gini (Outlook=Rain and Temp.) 🡺 (2/5)x0.5 + (3/5)x0.444 = 0.466

**Gini of wind for rain outlook**

Table

Description automatically generated

Gini (Outlook=Rain and Wind=Weak) 🡺 1 – (3/3)2 – (0/3)2 = 0

Gini (Outlook=Rain and Wind=Strong) 🡺 1 – (0/2)2 – (2/2)2 = 0

Gini (Outlook=Rain and Wind) 🡺 (3/5)x0 + (2/5)x0 = 0

**Decision for rain outlook**

The winner is the wind feature for rain outlook because it has the minimum Gini index score in features.

Table

Description automatically generated

Put the wind features for the rain outlook branch and monitor the new sub data sets.

Diagram

Description automatically generated

Sub data sets for weak and strong wind and rain outlook

As seen, the decision is always yes when the wind is weak. On the other hand, the decision is always no if the wind is strong. This means that this branch is over.

Diagram

Description automatically generated

The final form of the decision tree built by the CART algorithm

Entropy and Gini impurity can be used reversibly. It doesn’t affect the result much. Although, Gini is easier to compute than entropy since entropy has a ‘log’ term calculation. That’s why the CART algorithm uses

If we plot Gini vs entropy graph, we can see there is not much difference between them:

Diagram

Description automatically generated with medium confidence

**Advantages of Decision Tree:**

* It can be used for both Regression and Classification problems.
* Decision Trees are very easy to grasp as the rules of splitting are clearly mentioned.
* Complex decision tree models are very simple when visualized. It can be understood just by visualizing.
* Scaling and normalizing are not needed.

**Disadvantages of Decision Tree:**

* A small change in data can cause instability in the model because of the greedy approach.
* The probability of overfitting is very high for Decision Trees.
* It takes more time to train a decision tree model than other classification algorithms.

**Exercise**

**Implementation in Python**

We will use the Sklearn module to implement a decision tree algorithm. Sklearn uses the CART (Classification and Regression Trees) algorithm and by default, it uses Gini impurity as a criterion to split the nodes.

There are other algorithms like ID3, C4.5, Chi-Square, etc.

We will see the use of CART in the following implementation

Graphical user interface, text, application

Description automatically generated

A picture containing logo

Description automatically generated

Graphical user interface

Description automatically generated

The data set consists following input variables: 1 – fixed acidity 2 - volatile acidity 3 - citric acid 4 - residual sugar 5 - chlorides 6 - free sulfur dioxide 7 - total sulfur dioxide 8 - density 9 - pH 10 - sulphates 11 – alcohol and the Output variable gives the quality of the wine based on the input variables: 12 - quality (score between 0 and 10)

Shape

Description automatically generated with low confidence

Graphical user interface, application

Description automatically generated

We can see there is no missing data in the columns. Great!

Let’s split and fit the data without any pre-processing

A picture containing graphical user interface

Description automatically generated





Let’s see the performance of the model

Graphical user interface, application, Word

Description automatically generated

The train score is 100%. Let’s see the test score



Text, application

Description automatically generated

Let’s scale the data and check the model performance

A picture containing chart

Description automatically generated



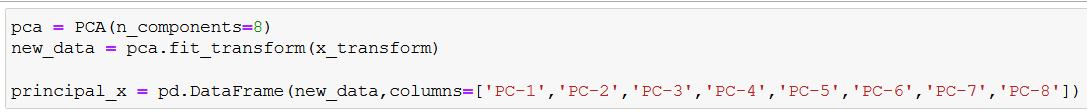
Text

Description automatically generated

Chart, line chart

Description automatically generated

We can see that around 95% of the variance is being explained by 8 components. So, instead of giving all columns as input in our algorithm let’s use these 8 principal components instead.



Table

Description automatically generated

Let’s see how well our model performs on this new data. let's first visualize the tree on the data without doing any pre-processing

Text

Description automatically generated

The performance of the model after PCA is 59%

Let’s do some hyper-parameter tuning. We are tuning three hyperparameters right now, we are passing the different values for both parameters.

Text

Description automatically generated

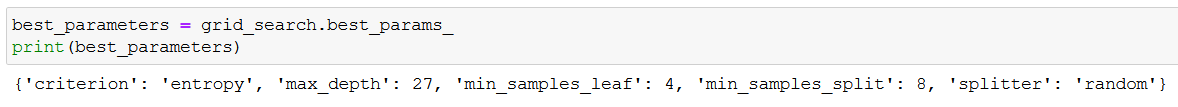
Text

Description automatically generated

Text

Description automatically generated

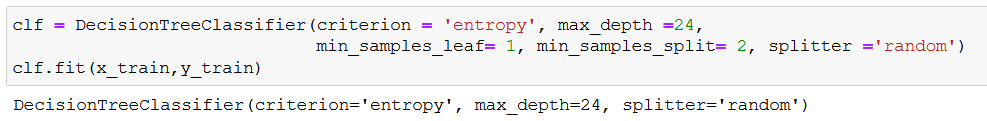
Let’s check the best parameters from the grid search CV.



A picture containing text

Description automatically generated

Let’s build the model with updated parameters.



Graphical user interface, application, Word

Description automatically generated with medium confidence

The performance score after the hyper-parameter tuning is 59%.

**Ensemble Techniques**

We regularly come across many game shows on television and you must have noticed an option of “Audience Poll”. Most of the time a contestant goes with the option which has the highest vote from the audience and most of the time they win. We can generalize this in real life as well where taking opinions from most people is much more preferred than the opinion of a single person. The Ensemble technique has a similar underlying idea where we aggregate predictions from a group of predictors, which may be classifiers or regressors, and most of the time the prediction is better than the one obtained using a single predictor. Such algorithms are called Ensemble methods and such predictors are called Ensembles.

Let’s suppose we have ‘n’ predictors:

Z1, Z2, Z3, ......., Zn with a standard deviation of σ

Var(z) = σ^2

If we use single predictors Z1, Z2, Z3, ......., Zn the variance associated with each will be σ2 but the expected value will be the average of all the predictors.

Let’s consider the average of the predictors:

µ = (Z1 + Z2 + Z3+.......+ Zn)/n

if we use µ as the predictor then the expected value still remains the same but see the variance now:

variance(µ) = σ^2/n

So, the expected value remained ‘µ’ but variance decreases when we use an average of all the predictors.

This is why taking means is preferred over using single predictors.

Ensemble methods take multiple small models and combine their predictions to obtain a more powerful predictive power.

There are a few very popular Ensemble techniques which we will talk about in detail such as Bagging, Boosting, stacking, etc.

Diagram

Description automatically generated

Diagram

Description automatically generated

**Bagging (Bootstrap Aggregation)**

In real-life scenarios, we don’t have multiple different training sets on which we can train our model separately and in the end combine their result. Here, bootstrapping comes into the picture. Bootstrapping is a technique of sampling different sets of data from a given training set by using replacement. After bootstrapping the training dataset, we train the model on all the different sets and aggregate the result. This technique is known as Bootstrap Aggregation or Bagging.

Let’s see the definition of bagging:

Bagging is the type of ensemble technique in which a single training algorithm is used on different subsets of the training data where the subset sampling is done with replacement (bootstrap). Once the algorithm is trained on all the subsets, then bagging makes the prediction by aggregating all the predictions made by the algorithm on different subsets. In the case of regression, bagging prediction is simply the mean of all the predictions and in the case of classifier, bagging prediction is the most frequent prediction (majority vote) among all the predictions.

Bagging is also known as a parallel model since we run all models parallelly and combine their results at the end.

Diagram

Description automatically generated

Diagram

Description automatically generated

**Advantages of Bagging Model**

* Bagging significantly decreases the variance without increasing bias.
* Bagging methods work so well because of diversity in the training data since the sampling is done by bootstrapping
* Also, if the training set is very huge, it can save computational time by training the model on the relatively smaller dataset and still can increase the accuracy of the model.
* Works well with small datasets as well.

**Disadvantages of Bagging Model**

* The main disadvantage of bagging is that it improves the accuracy of the model at the expense of interpretability. i.e., if a single tree was being used as the base model, then it would have a more attractive and easily interpretable diagram, but with the use of bagging this interpretability gets lost.

**Pasting**

Pasting is an ensemble technique similar to bagging with the only difference being that there is no replacement done while sampling the training dataset. This causes less diversity in the sampled datasets and data ends up being correlated. That's why bagging is more preferred than pasting in real scenarios.

**Out-of-Bag Evaluation**

In bagging, when different samples are collected, no sample contains all the data but a fraction of the original dataset. There might be some data that are never sampled at all. The remaining data which are not sampled are called out of bag instances. Since the model never trains over these data, they can be used for evaluating the accuracy of the model by using these data for prediction. We do not need a validation set or cross-validation and can use out-of-bag instances for that purpose.

**Exercise**

We are using the ‘breast\_cancer’ dataset to demonstrate the bagging.

**Importing the libraries and data**

Text

Description automatically generated

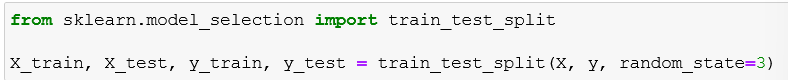
Text

Description automatically generated with low confidence

Let’s split the train and test dataset.

Text

Description automatically generated



Now. Let’s create a ‘bagging classifier’ object with a decision tree classifier.



In Scikit-Learn, you can set oob\_score=True when creating a ‘BaggingClassifier’ to request an automatic oob evaluation after training. The following code demonstrates this. The resulting evaluation score is available through the oob\_score\_ variable:

Text

Description automatically generated with medium confidence

Text

Description automatically generated

According to this oob evaluation, this ‘BaggingClassifier’ is likely to achieve about 96% accuracy on the test set. Let’s verify this:

Graphical user interface, text

Description automatically generated

**Random Forests**

Decision trees are one of such models which have low bias but high variance. We have studied that decision trees tend to overfit the data. So bagging technique becomes a very good solution for decreasing the variance in a decision tree. Instead of using a bagging model with the underlying model as a decision tree, we can also use Random forest which is more convenient and well optimized for decision trees. The main issue with bagging is that there is not much independence among the sampled datasets i.e. there is a correlation. The advantage of random forests over bagging models is that the random forests make a tweak in the working algorithm of the bagging model to decrease the correlation in trees. The idea is to introduce more randomness while creating trees which will help in reducing correlation.

Let’s understand how the algorithm works for a random forest model:

1. Just like in bagging, different samples are collected from the training dataset using bootstrapping.
2. On each sample, we train our tree model, and we allow the trees to grow with high depths.

Now, the difference within the random forest is how the trees are formed. In bootstrapping, we allow all the sample data to be used for splitting the nodes but not with random forests. When building a decision tree, each time a split is to happen, a random sample of ‘m’ predictors are chosen from the total ‘p’ predictors. Only those ‘m’ predictors are allowed to be used for the split.

Why is that?

Suppose in those ‘p’ predictors, 1 predictor is very strong. Now each sample this predictor will remain the strongest. So, whenever trees will be built for these sampled data, this predictor will be chosen by all the trees for splitting and thus will result in a similar kind of tree formation for each bootstrap model. This introduces correlation in the dataset and averaging correlated dataset results do not lead to low variance. That’s why in the random forest the choice for selecting node for split is limited and it introduces randomness in the formation of the trees as well.

Most of the predictors are not allowed to be considered for the split. Generally, the value of ‘m’ is taken as m ≈√p, where ‘p’ is the number of predictors in the sample.

When m=p, the random forest model becomes the bagging model.

\*This method is also referred to as “Feature Sampling”

Chart, histogram

Description automatically generated

The above graph represents the decrease in test classification error as we select different values of ‘m’.

1. Once the trees are formed, a prediction is made by the random forest by aggregating the predictions of all the models. For the regression model, the mean of all the predictions is the final prediction and for classification mode, the mode of all the predictions is considered the final predictions.

**Working of Random Forest Model**

From the given dataset different samples are created by bootstrapping and these samples are used to train different decision trees. Once the training is complete, prediction is made using all the different models

**Predicting Outcome**

Random forest makes the prediction by taking the mode of all the predictions made by all the models since this is the case of classification. This process is also known as “Majority voting”. We can also use prediction probability to make the final prediction. We can use the predict\_proba method, which will predict a probability from 0 to 1 that a given class is the right one for a row. For a problem with the output being only 0 and 1, we'll get a matrix with as many rows as there is in the data and 2 columns. predict\_proba will return something like this:

Table

Description automatically generated

Each row corresponds to a prediction. The first column is the probability that the prediction is a 0, the second column is the probability that the prediction is a 1. Each row adds up to 1.

If we just take the second column, we get the average value that the classifier would predict for that row. If there's a .9 probability that the correct classification is 1, we can use the .9 as the value the classifier is predicting. This will give us a continuous output in a single vector instead of just 0 or 1. We can then add all the vectors we get through this method together and divide by the number of vectors to get the mean prediction by all the members of the ensemble. We can then round off to get 0 or 1 predictions. Similarly, in the case of regression, Random Forest makes the prediction by taking the mean of all the predictions made by different models.

**Advantages and Disadvantages of Random Forest:**

1. It can be used for both regression and classification problems.
2. Since the base model is a tree, handling of missing values is easy.
3. It gives very accurate results with very low variance.
4. Results of a random forest are very hard to interpret in comparison with decision trees.
5. High computational time than other respective models.

Random Forest should be used where accuracy is up utmost priority and interpretability is not very important. Also, the computational time is less expensive than the desired outcome.

**Exercise**

Let’s see the python implementation of Random Forest.

**Importing the necessary libraries and data**

Graphical user interface, text, application

Description automatically generated

A picture containing text

Description automatically generated

Graphical user interface

Description automatically generated

**Train & Test Split**

A picture containing graphical user interface

Description automatically generated



Creating Random Forest object

Text

Description automatically generated with medium confidence

Let’s fit the model with default parameters

Graphical user interface, text, application

Description automatically generated with medium confidence

Performance evaluation

Graphical user interface, text

Description automatically generated

The accuracy score of the random forest algorithm is 67% with default parameters.

Let’s do the hyperparameter tuning with grid searchcv and check the accuracy of the model

Text

Description automatically generated

A picture containing diagram

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

Let’s check the best parameters

A picture containing text

Description automatically generated

Text

Description automatically generated

We are going the build the model using the best parameters

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated with medium confidence

Let’s check the performance of the model.

Graphical user interface, text, application

Description automatically generated

The random forest classifier gave us 67% accuracy on the test data.

**Boosting**

Boosting is an ensemble approach (meaning it involves several trees) that starts from a weaker decision and keeps on building the models such that the final prediction is the weighted sum of all the weaker decision-makers. The weights are assigned based on the performance of an individual tree.

Diagram

Description automatically generated

Ensemble parameters are calculated in a stagewise way which means that while calculating the subsequent weight, the learning from the previous ‘tree’ is considered as well.

Weak classifier – why a tree?

First, what is a weak classifier? Weak classifier – slightly better than random guessing.

Any algorithm could have been used as a base for the boosting technique, but the reason for choosing trees are:

**Pro’s**

* Computational scalability
* Handles missing values
* Robust to outliers
* Does not require feature scaling
* Can deal with irrelevant inputs
* Interpretable (if small)
* Handles mixed predictors as well (quantitative and qualitative)

**Con’s**

* Inability to extract a linear combination of features
* High variance leading to a small computational power

And that’s where boosting comes into the picture. It minimizes the variance by taking into consideration the results from various trees.

In every machine learning model, the training objective is a sum of a loss function **L** and regularisation **Ω**:



The loss function controls the predictive power of an algorithm and the regularisation term controls its simplicity.

There are several algorithms that use boosting. A few are discussed here.

**Ada Boost (Adaptive Boosting)**

The steps to implement the Ada Boost algorithm using the decision trees are as follows:

**Algorithm:**

Assume that the number of training samples is denoted by N, and the number of iterations (created trees) is M. Notice that possible class outputs are Y = {-1, 1}

1. Initialize the observation weights *w*i = 1/N where i = 1, 2, ….., N for all the samples.
2. For m = 1 to M:
   * Fit a classifier Gm(x) to training data using weights wi
   * Compute

Text

Description automatically generated

* + Compute



This is the contribution of the tree to the result.

* + Calculate the new weights using the formula:



* + Normalize the new sample weights so that their sum is 1.
  + Construct the next tree using the new weights.

1. In the end, compare the summation of results from all the trees and the final result is either the one with the highest sum (for regression) or it is the class that has the most weighted voted average (for classification).

Text

Description automatically generated

**Example:**

For understanding this algorithm, we’ll use the following simple dataset for heart

Table

Description automatically generated

* There are a total of 8 rows in our dataset. Hence, we’ll initialize the sample weights (w = 1/N) as 1/8 in the beginning. And, in the beginning, all the samples are equally important.

Table, Excel

Description automatically generated

* We’ll consider the individual columns to create weak decision-makers as shown below and then try to figure out what are the correct and incorrect predictions based on that column.

Diagram

Description automatically generated

Diagram

Description automatically generated

A picture containing text, businesscard

Description automatically generated

* We’ll now calculate the Gini index of the individual stumps using the formula

Text, letter

Description automatically generated

And, we select the tree with the lowest Gini index. This will be the first decision-maker for our model.

* Now, we’ll calculate the contribution of this tree (stump) to our final decision using the formula:



As this stump classified only one data incorrectly out of the 8, hence the total error is 1/8

Putting this into the formula we get contribution = 0.97

* We’ll now calculate the weights using the formula:
  + Increase the sample weight for incorrectly classified datapoints New weight = old weight \* e^ contribution

= 1/8 \* e0.97 = 0.33

* + Decrease the sample weight for incorrectly classified datapoints New weight = old weight \* e^ contribution

= 1/8 \* e-0.97 = 0.05

* + Populate the new weights as shown below:

Table

Description automatically generated

* Normalize the sample weights: if we add all the new sample weights, we get 0.68. hence, for normalization we divide all the sample weights by 0.68 and then create normalized sample weights as shown below:

Graphical user interface

Description automatically generated with medium confidence

These new normalized weights will act as the sample weights for the next iteration.

* Then we create new trees which consider the dataset which was prepared using new sample weights.
* Suppose, m trees (stumps) are classifying a person as a heart patient and n trees (stumps) are classifying a person as a healthy one, then the contribution of m and n trees are added separately and whichever has the higher value, the person is classified as that.

For example, if the contribution of m trees is 1.2 and the contribution of n trees is 0.5 then the result will go in the favor of m trees and the person will be classified as a heart patient.

**Gradient Boosted Trees**

Gradient Boosted Trees use decision trees as estimators. It can work with different loss functions (regression, classification, risk modeling, etc), evaluate its gradient, and approximate it with a simple tree (stage-wisely, that minimizes the overall error).

AdaBoost is a special case of Gradient Boosted Tree that uses exponential loss function.

**The Algorithm:**

* Calculate the average of the label column as initially, this average shall minimize the total error.
* Calculate the Pseudo residuals.

“Pseudo residual = actual label – the predicted result (Which is average in the first iteration)

Derivative of the Pseudo residual

Text

Description automatically generated

Where L is the loss function.

Here, the gradient of the error term is getting calculated as the goal is to minimize the error. Hence the name gradient boosted trees.

* Create a tree to predict the pseudo residuals instead of a tree to predict the actual column values.
* New result = previous result + learning rate \* residual

Mathematically,



Where v is the learning rate and 𝛾 is the residual

Repeat these steps until the residual stops decreasing.

**Example**

For understanding this algorithm we’ll use the following simple dataset for weight prediction.

Table

Description automatically generated

* For the first iteration, calculate the average of the target column (weight here) as it minimizes the residual initially.



* We consider this as the first prediction and then we’ll calculate the residual which is the difference between the predicted and the actual value as shown below:

Diagram

Description automatically generated

Now we build a tree to predict the residuals as shown below:

Diagram

Description automatically generated

We are only building here to a limited depth just for simplicity. As you can see, some leaves have more than one residual. For those, we’ll calculate the average, and the final tree will look like this:

Diagram, text

Description automatically generated

* Now for prediction, we use the formula



If we consider the learning rate as 0.1, the result becomes.



Similarly, the new predictions for all the rows are calculated.

* The above steps are repeated until there is no significant improvement in residuals.
* The final result is given by

Final Value = First Prediction + learning rate \* 1st residual + learning rate \* 2nd residual + and so on.

**XGBoost**

XGBoost improves the gradient boosting method even further.

*“XGBoost (extreme gradient boosting) regularises data better than normal gradient boosted trees.”*

XGBoost’s objective function is the sum of loss function evaluated over all the predictions and a regularization function for all predictors (*j* trees). In the formula ***fj*** means a prediction coming from the *jth* tree.

Text

Description automatically generated

Loss function depends on the tasks being performed (classification, regression, etc.) and a regularization term is described by the formula equation:

A picture containing text, clock, watch

Description automatically generated

The first part (𝛾𝑇) is responsible for controlling the overall number of created leaves, and the second term



Watches over the scores.

**Mathematics Involved** Unlike the other tree-building algorithms, XGBoost doesn’t use entropy or Gini indices. Instead, it utilizes gradient (the error term) and hessian for creating the trees. Hessian for a Regression problem is the *number of residuals* and for a classification problem. Mathematically, Hessian is a second-order derivative of the loss at the current estimate given as:

Text

Description automatically generated

Where **L** is the loss function.

* Initialize the tree with only one leaf.
* compute the similarity using the formula

Text

Description automatically generated

Where 𝜆 is the regularisation term.

* Now for splitting data into a tree form, calculate



* For tree pruning, the parameter 𝛾 is used. This algorithm starts from the lowest level of the tree and then starts pruning based on the value of 𝛾.

Shape

Description automatically generated with medium confidence

* Learning is done using equation



where 𝜂 is the learning rate

**Case Study**

**Credit Card Fraud Detection**

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 is the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable, and it takes value 1 in case of fraud and 0 otherwise.

**Importing all the required libraries**

Text

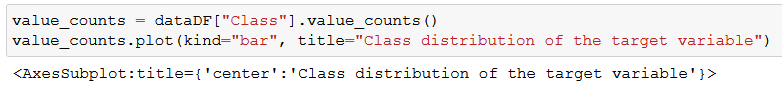
Description automatically generated

**Importing data**

Graphical user interface, text, application

Description automatically generated

If we look at the dimensions of the data frame we will notice that the data set contains 284,807 samples in total. It also appears that there are no missing values, however, as card transaction datasets usually contain mostly normal transactions and just a small fraction of fraudulent ones, our expectation is that the dataset will be highly unbalanced. We confirm this by checking the distribution of the target classes.



Chart, bar chart

Description automatically generated

Fraudulent transactions are 0.17% of the complete dataset. As expected, the majority of the samples are legitimate transactions. Only 0.17% of the transactions are flagged as fraudulent. Let’s also look at the basic descriptive statistics for all attributes.

Table

Description automatically generated

Inspecting the statistics above reveals that V1-V28 attributes are zero-cantered, but this is not the case for the other two input attributes Time and Amount.

Table

Description automatically generated

We can also plot the histogram for all input attributes and make sure nothing unusual stands out.

Text

Description automatically generated

A picture containing shoji, window, building, crossword puzzle

Description automatically generated

There isn’t much we can do about the principal components, but it looks like the Amount and Time attributes deserve a more detailed inspection.

First of all, the basic statistics for Time suggest that this attribute is given as a timestamp(seconds). We can convert it to local date and extract the hours in an attempt to identify if the hour of the transaction is correlated with the possibility of a transaction being fraudulent.

Text

Description automatically generated

Chart, histogram

Description automatically generated

It appears that the legitimate transactions plunge during the night and their rate increases with the start of the working day. In contrast, there is a peak of fraudulent transactions at around 2 am that looks unusual. In addition, the data for the fraudulent transactions looks more evenly spread. Let's write a simple function that will allow us to compare the basic statistics for a single attribute across the fraud/no-fraud classes.

A picture containing Word

Description automatically generated

Table

Description automatically generated

The fraudulent transactions definitely have a higher standard deviation, however, given their relatively low number we can't be certain if this is a genuine pattern or just a statistical coincidence.

We now shift our attention to the transaction amount. Let's visualize the distribution of fraudulent vs legitimate transactions via histograms.

Text

Description automatically generated

Graphical user interface, application, table, Excel

Description automatically generated

Let's also look at the basic statistics for the Amount attribute.

Table

Description automatically generated

It looks like the mean transaction amount is higher for fraudulent transactions, although the overall amount is significantly lower in absolute terms. We can also look at the top 5 most frequent legitimate transactions.



Text

Description automatically generated

versus the top 5 fraudulent transactions



Text, letter

Description automatically generated

We see that nearly 23% of all fraudulent transactions amount to 1.0, where the percentage for the same transaction amount in the legitimate set is closer to 5%. We also see a large number of 0-value transactions. which is probably the result of cardholder details verification transactions.

**Feature Engineering**

The first thing we'll do before we attempt any feature engineering is to create a holdout set. This is to prevent any information leakage into our test set.

Text

Description automatically generated with medium confidence

We noticed during the exploratory analysis that the Amount column is not zero mean-centered. Let's fix this, and also center the Hour attribute, which we'll be using instead of Time.

Graphical user interface, text

Description automatically generated

Next, we drop the Time attribute as we will be using Hour instead.

A screenshot of a computer

Description automatically generated with medium confidence

Now let's split the independent and the class variables into separate data frames.

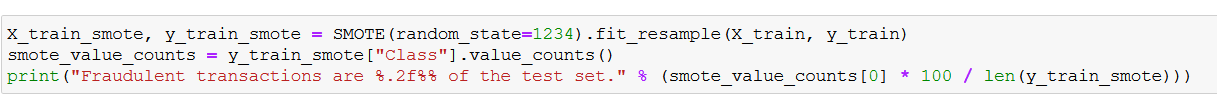
Text

Description automatically generated

Graphical user interface, text

Description automatically generated with medium confidence

**Oversampling**





Now that the class imbalance has been resolved, we can move forward with the actual model training.

## **Model training**

We will now train an XGBoost classifier, using the oversampled training set. First, we define a function that will perform a grid search for the optimal hyperparameters of the classifier. The highlights of the function are as follows:

We do a parameter search over the hyperparameters given in params

* The cross-validation strategy for each model uses 3 folds in a stratified KFold
* The metric the models in the search are evaluated on is the Area Under the Receiver Operating Characteristic Curve (ROC AUC)
* The function prints the parameters that yield the highest AUC score and returns the parameters of the best estimator as its output

Text

Description automatically generated

As the training set is quite large, we limit the grid search to a sample of 5000 observations. After xgboost\_search goes through all possible estimators it will print the parameters used for the best performing one and will return an XGBClassifier object that we can use to check how well the model generalizes.

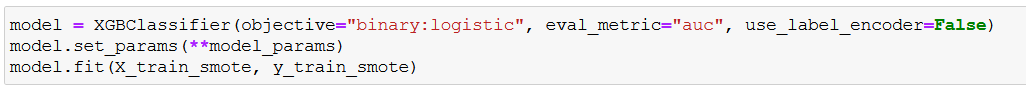
Note, that although we are using a relatively small subset of the training data, the search still needs to train over 1,600 models, and this will take a while



Text

Description automatically generated

Now that we have the set of parameters that produces the highest score, we can use them to train a model on the complete oversampled training set.



Text, letter

Description automatically generated

**Model Evaluation**

We will use a ROC AUC curve for evaluating how well our model performs on the holdout set (X\_test, y\_test). To generate the ROC curve we calculate the true positive (TP) and false positive (FP) rates on the holdout set at various threshold levels.

We also show the Area Under the Curve (AUC) on the plot, as it is equal to the probability the model will rank a uniformly drawn random positive higher than a uniformly drawn random negative.

Text

Description automatically generated

Chart, line chart

Description automatically generated

A standard approach for binary classification problems is to look at the probability produced by the model and classify the observation as class 0 if the probability is under 0.5, and class 1 if it is equal or over 0.5. In highly unbalanced datasets this interpretation could lead to poor predictions. Offsetting the threshold (threshold moving) is a standard technique for improving the predictions by finding an optimal interpretation of the probabilities produced by the classifier.

There is an extra factor that comes into play in fraud detection, which is that the cost of one type of misclassification is substantially higher than the other. In other words, classifying a legitimate transaction as fraud is inconvenient at best, but letting a fraudulent transaction slip through has more dire consequences. In this context, offsetting the threshold in a way that reduces the false negatives at the expense of false positives becomes a viable strategy.

Selecting the optimal threshold value can be performed in a number of ways. Looking at the ROC curve, we can intuitively see that the best performance (misclassification costs aside) would be yielded by the threshold that puts us in the top left section of the curve (i.e. TP rate is high, FP rate is low). With this criterion in mind, we can define a distance metric to the top left corner of the curve and find a threshold that minimizes it.

Text

Description automatically generated

We can also manually inspect the confusion matrices at different threshold values.

Text

Description automatically generated

Graphical user interface, diagram, application, Teams

Description automatically generated

Looking at the plot above, we can indeed confirm that T=0.9 gives the best interpretation. The false negatives and false positives are both low. Increasing the threshold further leads to missing more fraudulent transactions and reducing it lower almost doubles the number of false positives.

**SUMMARY**

In this chapter, we looked at Decision Tree, ensemble learning techniques like bagging and boosting. We looked at a real-world credit card transaction dataset and demonstrated how machine learning can be used to automate the detection of fraudulent transactions.

The imbalance in the dataset can be addressed by using under/oversampling techniques, and the interpretation of the probabilities can be fine-tuned to produce a better balance between false alarms and missed fraudulent transactions.

Finally, we have evaluated the performance of the model using the evaluation metrics like confusion matrix and ROC/AUC curves.

**Program Assignment**

Build an algorithm that can predict a fraudulent transaction based on transaction attributes, using the Fraud Detection business case.

Follow the below step by step approach:

* Data importing
* Data Exploring
* Data Cleaning
* Feature Engineering
* Feature selection
* Handling class imbalance issue
* Model building
* Hyperparameter tuning
* Model evaluation

**Assessment**

**Choose the appropriate option**

1. Decision tree is the most powerful for \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   1. Classification
   2. Regression
   3. Both A and B
   4. None of these
2. Decision tree is a flowchart like \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   1. Leaf structure
   2. Tree structure
   3. Steam
   4. None of the above
   5. All of the above
3. In Decision tree algorithm At the beginning, we consider the whole training set as \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   1. Leaf
   2. Root
   3. Steam
   4. None of these
4. Which of the following is/are true about bagging trees?
5. In bagging trees, individual trees are independent of each other
6. Bagging is the method for improving performance by aggregating the results of weak learners
   1. 1
   2. 2
   3. 1 and 2
   4. None of these
7. In Random Forest you can generate hundreds of trees (say T1, T2, …. Tn) and then aggregate the results of these trees. Which of the following is true about the individual (Tk) tree in Random Forest?
8. Individual tree is built on a subset of the features
9. Individual tree is built on all the features
10. Individual tree is built on a subset of observations
11. Individual tree is built on a full set of observations
    1. 1 and 3
    2. 1 and 4
    3. 2 and 3
    4. 2 and 4

**Fill in the spaces with appropriate answers**

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ algorithm doesn’t uses learning Rate as of one of its hyperparameter.
2. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is the measure of uncertainty of a random variable, it characterizes the impurity of an arbitrary collection of examples.
3. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is a metric to measure how often a randomly chosen element would be incorrectly identified.
4. Bootstrap and Aggregation, commonly known as \_\_\_\_\_\_\_\_\_\_\_\_\_
5. Random Forest has \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ as base learning models

**True or False**

1. Decision Tree (ID3) uses the Greedy approach to decide on the first split.
   1. True
   2. False
2. Decision Tree is prone to overfit and accuracy doesn’t help to generalize on the test data.
   1. True
   2. False
3. Boosted algorithms minimize errors in previously predicted values by the last estimator. So it always increases training error.
   1. Ture
   2. False
4. The first tree in boosted trees works on the original data, whereas all the rest work on a modified version of the data.
   1. True
   2. False
5. Boosted decision trees don’t perform better than logistic regression on anomaly detection problems (Imbalanced Class Problems).
   1. True
   2. False

**Assessment Solutions**

**Choose the appropriate option**

1. C
2. B
3. B
4. C
5. E

**Fill in the spaces with appropriate answers**

1. Random Forest
2. Entropy
3. Gini Index
4. Bagging
5. Multiple decision trees

**True or False**

1. True
2. True
3. False
4. True
5. False