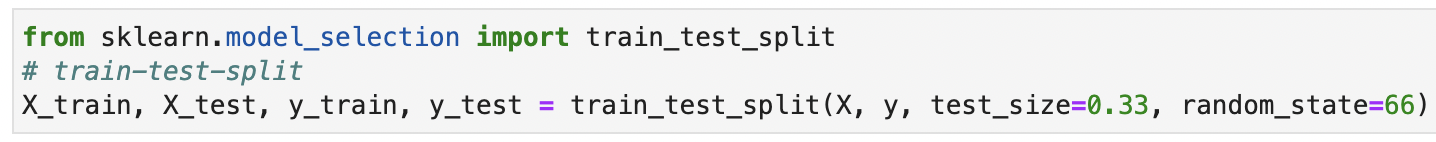
# Random Forest In SciKit Learn

# train-test-split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=66)



from sklearn.ensemble import RandomForestClassifier

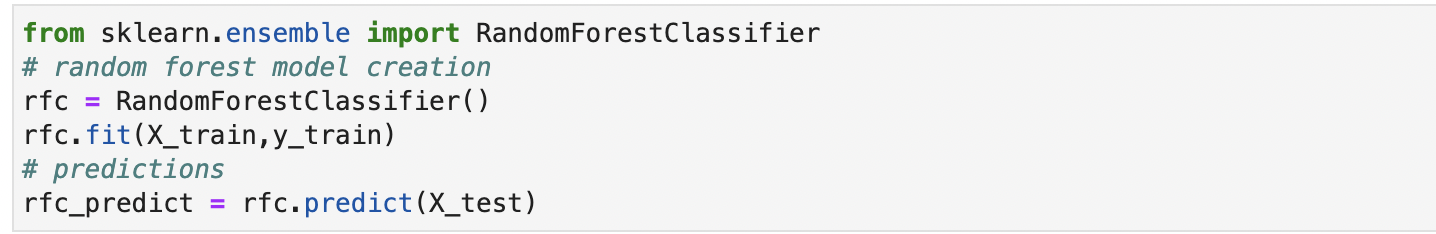
# random forest model creation

rfc = RandomForestClassifier()

rfc.fit(X\_train,y\_train)

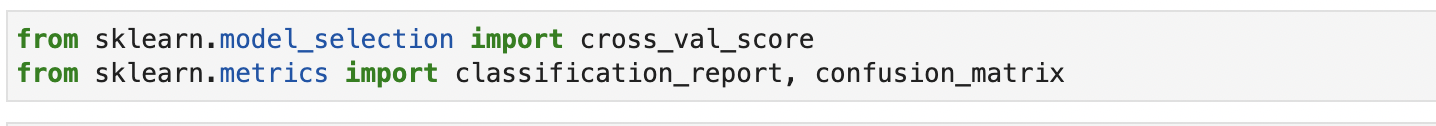
# predictions

rfc\_predict = rfc.predict(X\_test)



from sklearn.model\_selection import cross\_val\_score

from sklearn.metrics import classification\_report, confusion\_matrix



print("-- Confusion Matrix --")

print(confusion\_matrix(y\_test, rfc\_predict))

Before jumping to explain Random Forest algorithm, we must first grasp the concept of **bagging**. Bagging is a straightforward but effective ensemble strategy. It's a generic approach for lowering the variance in our model. Your model is overfitted if it has a greater variance. Certain algorithms like decision tree, are very well known to have a lot of volatility. In another aspect, they are highly dependent on the data used to train them. If the underlying data is modified even slightly, the resulting algorithm can be different, and our model's predictions will be substantially different as a result.

Bagging is a method of dealing with large volatility. By averaging numerous decision trees, it can reduce overfitting in a methodical way. Bagging employs bootstrap sampling before averaging the different models to obtain the final predictions. Bootstrap sampling is basically taking rows from the training dataset at random and replacing them.

As a result of bagging, it is possible to draw a single training example multiple times from the dataset. As a result, an altered version of the training dataset emerges. This also allows us to generate fresh data which is actually similar to the dataset that we began with. This allows us to fit a variety of models that are similar.

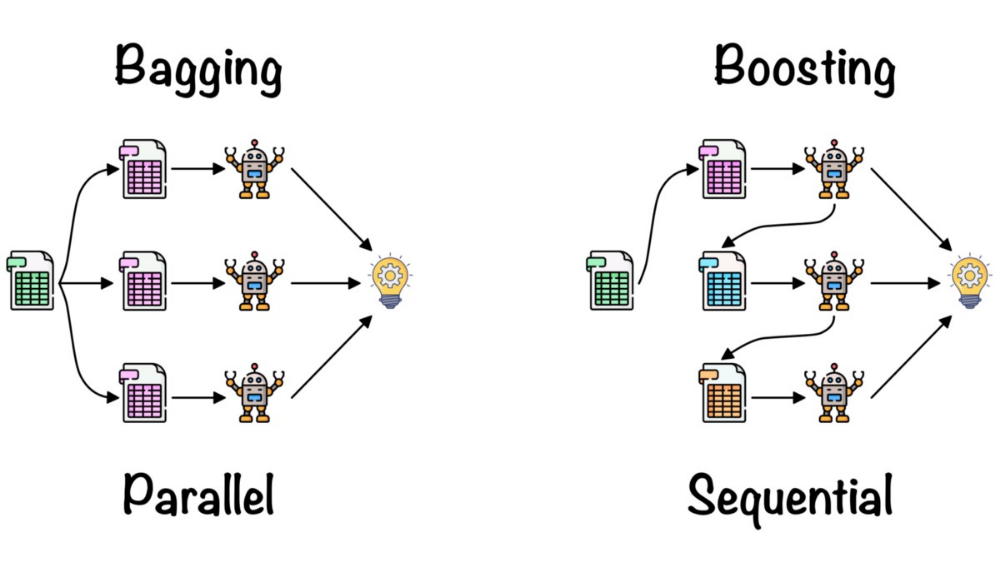
Let’s see how bagging works?

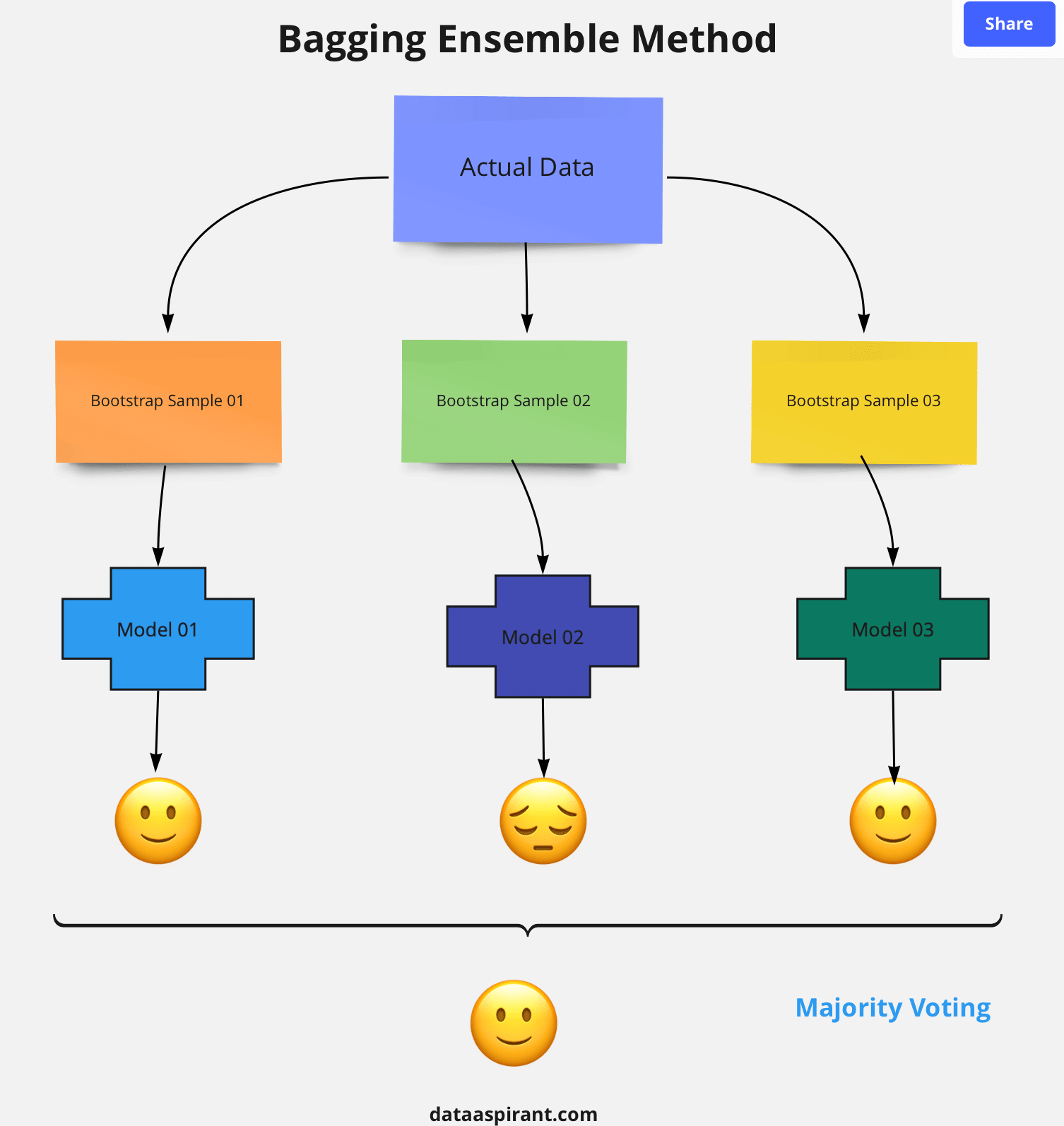
You take X replacement samples from the original data set, where X is a number of points less than or equal to the total number of samples in the training dataset.

On newly created bootstrapped samples, rain decision trees. Steps 1 and 2 can be repeated as many times as you like. In general, the more trees there are, the better the model. But keep in mind! An excessive number of decision trees can complicate a model and eventually lead to overfitting as your model begins to see relationships in the data that did not exist in the first place.

To use the bagged trees approach to make a prediction, you must first make a prediction from each of the decision trees, and then simply average the guesses together to get a final prediction. The average forecast over the sampled bootstrapped trees is known as a bagged or ensemble prediction. The council model is pretty similar to your bagged trees idea. When a council needs to make a decision, it usually takes a majority vote into account. The alternative with the most votes (for example, option A with 100 votes and option B with 90 votes) is the council's final choice.

Similarly, when you're trying to solve a classification problem with bagging, you're simply taking a majority vote of all your decision trees. We just take the average of all our decision tree forecasts in the case of regression. The combined knowledge of a diversified group of decision trees usually outperforms the knowledge of any single tree. As a result, bagged trees perform better in terms of prediction.





Now, Let’s talk about Random Forest

In one regard, random forest bagging differs from vanilla bagging. It employs a modified tree learning method that inspects a random subset of the features at each learning split. We do this to avoid a tree-to-tree connection. If we have a very strong predictor in the data set, along with a number of other moderately strong predictors, then most or all of our decision trees in the bagged tree collection will use the very strong predictor for the initial split! All of the bagged trees will have the same appearance. As a result, all of the bagged tree forecasts will be highly connected. Correlated predictors can't help you improve your prediction accuracy. Random Forests systematically avoids correlation and increases model performance by selecting a random subset of features. The Random Forest algorithm is demonstrated in the example below.

One of the most extensively used ensemble learning methods is random forest. Why is it so successful? The reason for this is that by combining numerous samples from the original dataset, the final model's variance is reduced. Keep in mind that low variance equates to low overfitting. Because our dataset is only a small sample of the population of all possible cases of the phenomenon we're trying to predict, overfitting occurs when our model tries to explain slight fluctuations in the dataset. If our training set was sampled incorrectly, it may contain undesired (but inevitable) artefacts such as noise, outliers, and over- or underrepresented samples.

Where to use Random Forest?

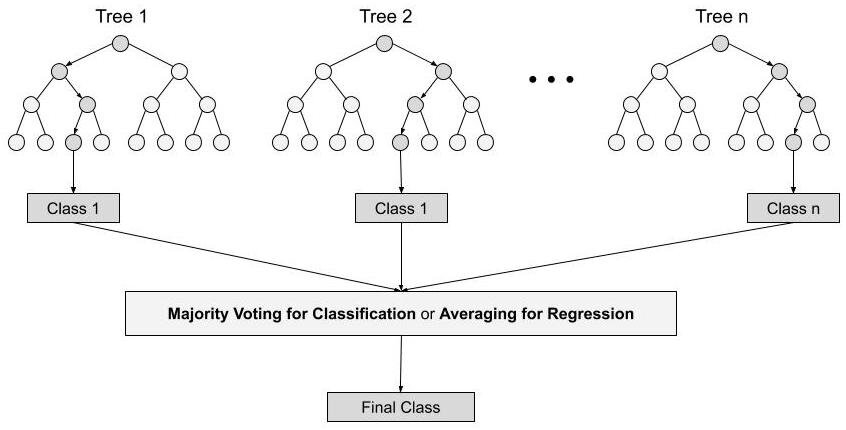
Regression Example –

Let's imagine you want to figure out what your town's average household income is. Using the Random Forest Algorithm, you may quickly find an approximation. You'd start by sending out surveys that asked folks to answer a variety of questions. Each person's estimated household income would be calculated based on how they answered these questions.

After you've located numerous people's decision trees, you can use the Random Forest Algorithm to analyse the data. You'd look at the outcomes of each decision tree and utilise random forest to get an average income for all of them. This algorithm would give you an accurate estimate of the average household income of the persons you polled if you used it.

Classification Example –

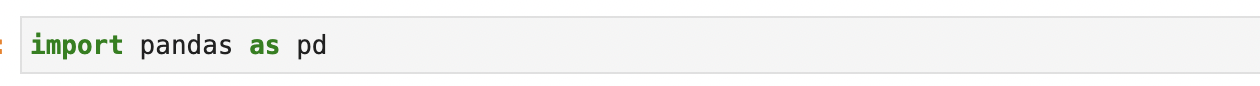
The next example involves classification data, sometimes known as non-numerical data. Assume you're conducting market research for a new business that wants to discover who is most likely to buy its items. You'll most likely begin by polling a sample of people in the same target market about their purchasing habits and the types of products they prefer. You'll be able to determine whether they're a potential customer or not based on their responses.



Implementation of Random Forest in Python using Sci-Kit Learn :

Let’s take a simple classification problem and solve it using Random Forest Classifier.

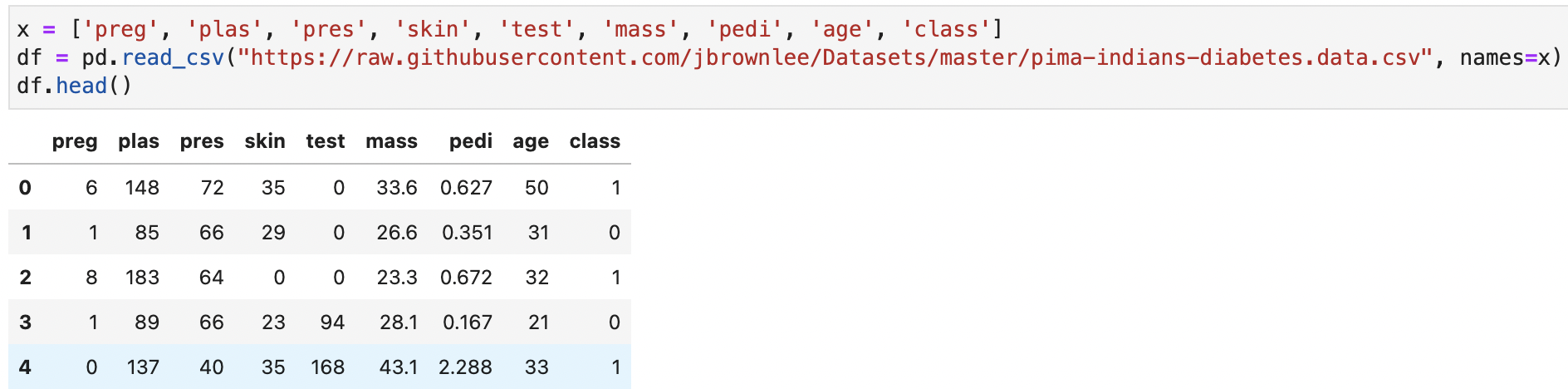
import pandas as pd



x = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

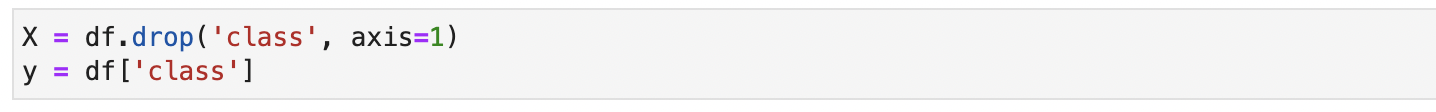
df = pd.read\_csv("https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv", names=x)

df.head()



X = df.drop('class', axis=1)

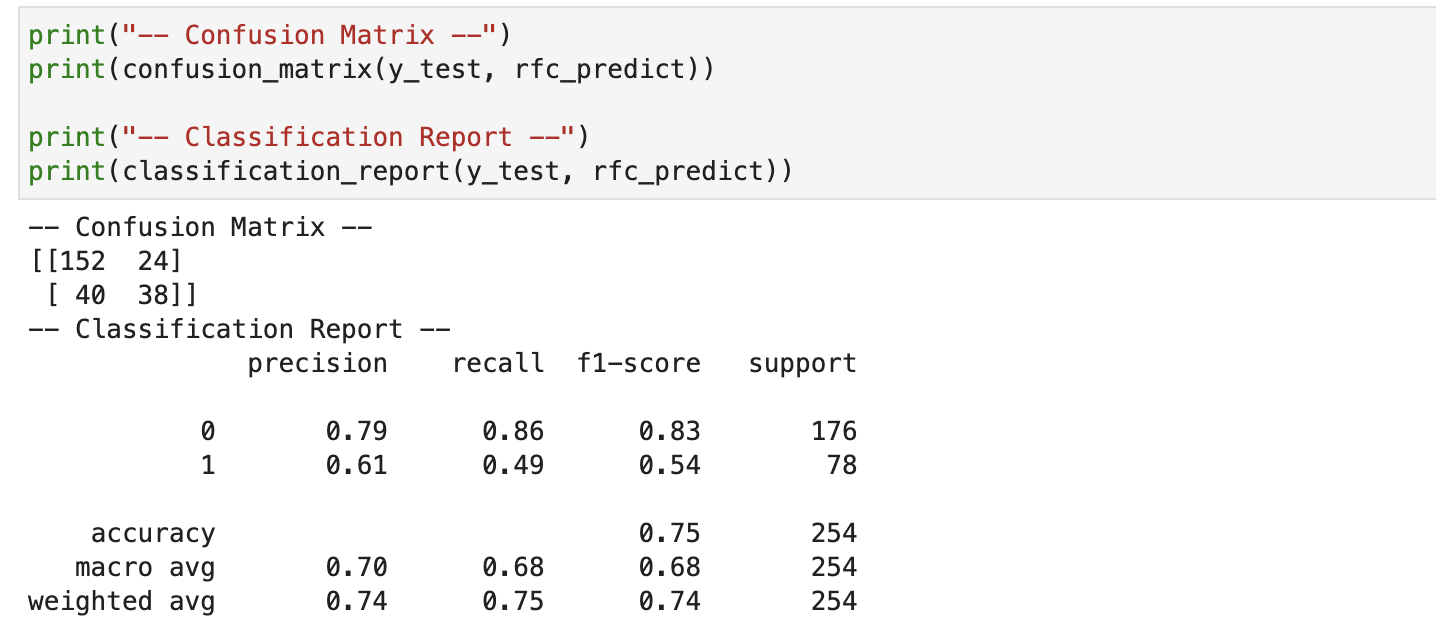
y = df['class']



from sklearn.model\_selection import train\_test\_split

print("-- Classification Report --")

print(classification\_report(y\_test, rfc\_predict))



Advantages of Random Forest Algorithm

* The Random Forest technique will prevent the overfitting problem in classification situations.
* The same random forest approach can be utilised for classification and regression tasks.
* The Random Forest technique can be used to feature engineer, or discover the most relevant features from a training dataset.

*Following hyperparameters increases the predictive power:*

1. **n\_estimators**– number of trees the algorithm builds before averaging the predictions.

2. ***max\_features****–*maximum number of features random forest considers splitting a node.

*3.****mini\_sample\_leaf****–*determines the minimum number of leaves required to split an internal node.

*Following hyperparameters increases the speed:*

*1.****n\_jobs****–*it tells the engine how many processors it is allowed to use. If the value is 1, it can use only one processor but if the value is -1 there is no limit.

*2.****random\_state****–*controls randomness of the sample. The model will always produce the same results if it has a definite value of random state and if it has been given the same hyperparameters and the same training data.

*3.****oob\_score****– OOB* means out of the bag. It is a random forest cross-validation method. In this one-third of the sample is not used to train the data instead used to evaluate its performance. These samples are called out of bag samples.

|  |  |
| --- | --- |
| **Decision trees** | **Random Forest** |
| 1. Decision trees normally suffer from the problem of overfitting if it’s allowed to grow without any control. | 1. Random forests are created from subsets of data and the final output is based on average or majority ranking and hence the problem of overfitting is taken care of. |
| 2. A single decision tree is faster in computation. | 2. It is comparatively slower. |
| 3. When a data set with features is taken as input by a decision tree it will formulate some set of rules to do prediction. | 3. Random forest randomly selects observations, builds a decision tree and the average result is taken. It doesn’t use any set of formulas. |

### Important Features of Random Forest

**1. Diversity-**Not all attributes/variables/features are considered while making an individual tree, each tree is different.

**2. Immune to the curse of dimensionality-** Since each tree does not consider all the features, the feature space is reduced.

**3. Parallelization-**Each tree is created independently out of different data and attributes. This means that we can make full use of the CPU to build random forests.

**4.  Train-Test split-**In a random forest we don’t have to segregate the data for train and test as there will always be 30% of the data which is not seen by the decision tree.

Summary

I hope this post provided you with a basic grasp of the Random Forest algorithm.

One of the best aspects about random forest is that it can accept missing variables, so it's a perfect choice for anyone who wants to create a model quickly and efficiently.

Random forest is a fast, simple, versatile, and robust model, but it does have certain drawbacks.

References:

* [Jupyter Book Online](https://jupyter.org/try)
* Official documentation of Random Forest [Classifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) and [Regressor](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html)