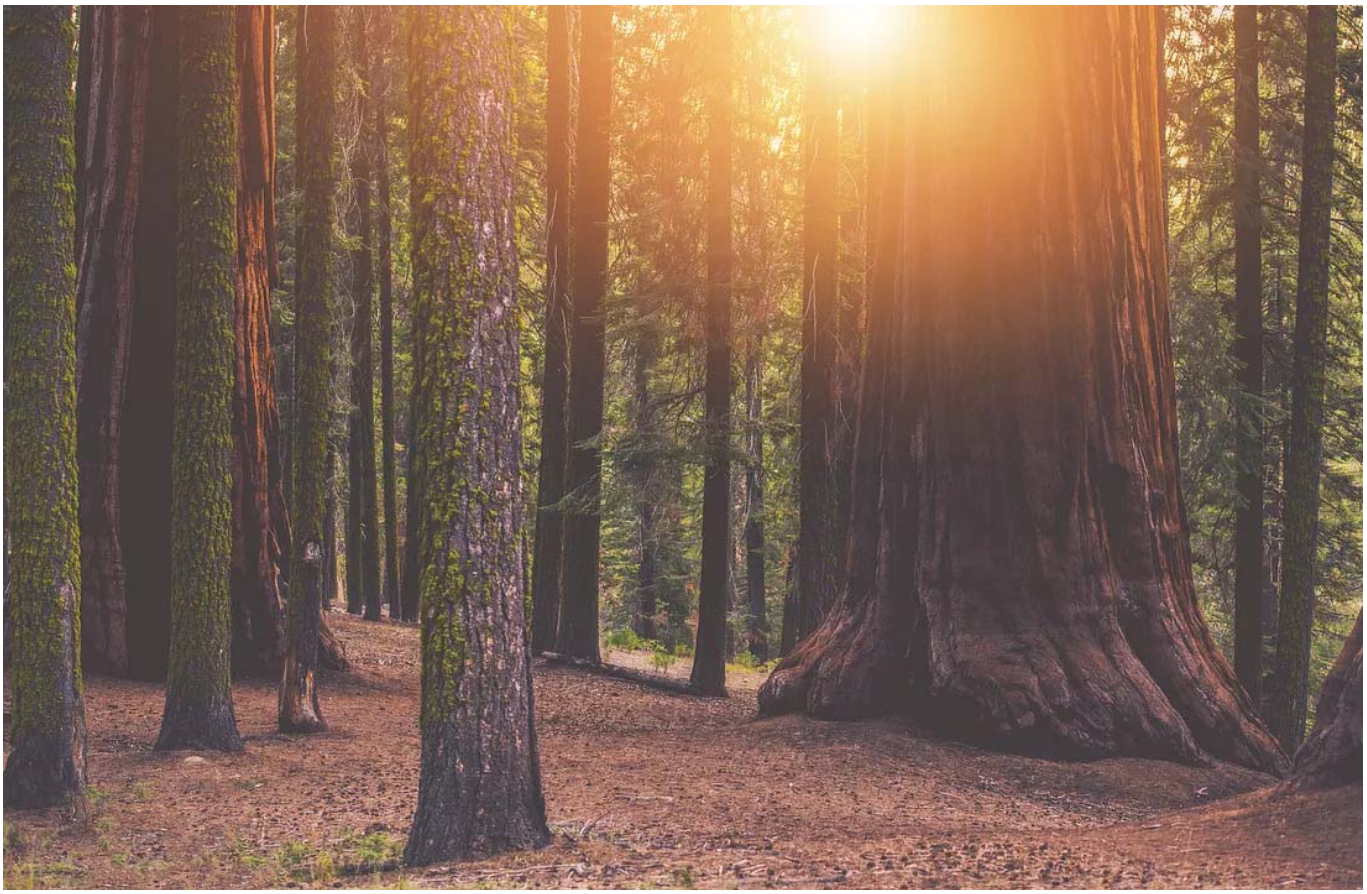
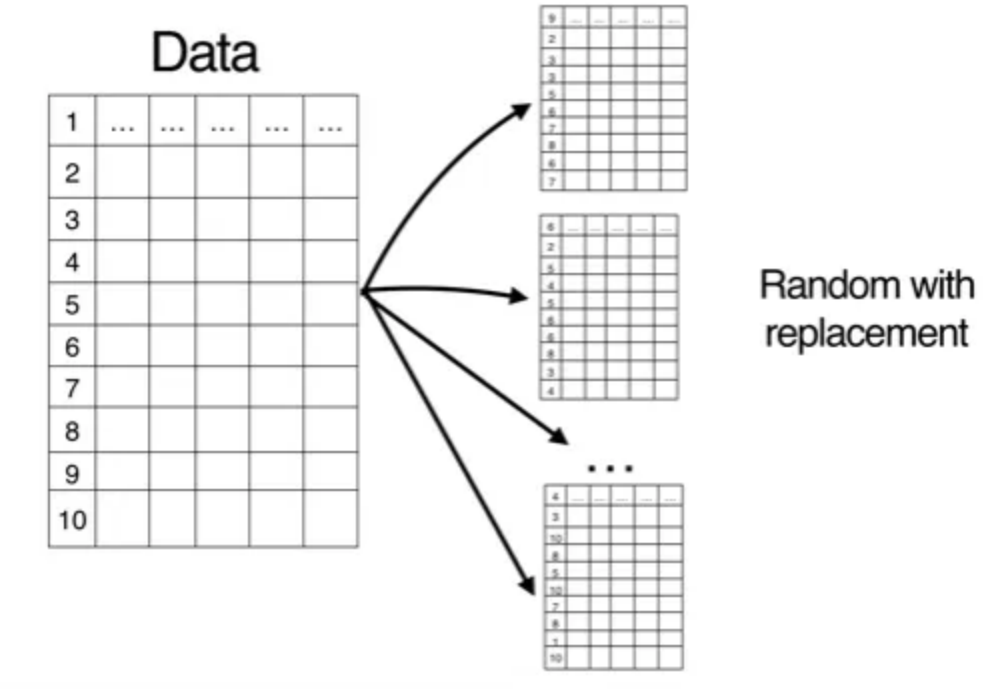
Understanding Random Forests



In this article, we will explore the famous supervised machine learning algorithm “Random Forests”. The aim of this article is to give you a holistic and intuitive understanding about how this algorithm works. Also, this article is a part of the series “One Algorithm at a Time”, an initiative that I recently started on Medium to demystify complex machine learning algorithms and help readers understand data science intuitively.

Before we start our discussion on random forests, we first need to understand Bagging. Bagging is a simple and a very powerful ensemble method. It is a general procedure that can be used to reduce our model’s variance. A higher variance means that your model is overfitted. Certain algorithms such as decision trees usually suffer from high variance. In another way, decision trees are extremely sensitive to the data on which they have been trained. If the underlying data is changed even a little bit, then the resulting decision tree can be very different and as result our model’s predictions will change drastically. Bagging offers a solution to the problem of high variance. It can systematically reduce overfitting by taking an average of several decision trees. Bagging uses bootstrap sampling and finally aggregates the individual models by averaging to get the ultimate predictions. **Bootstrap sampling simply means sampling rows at random from the training dataset with replacement.**

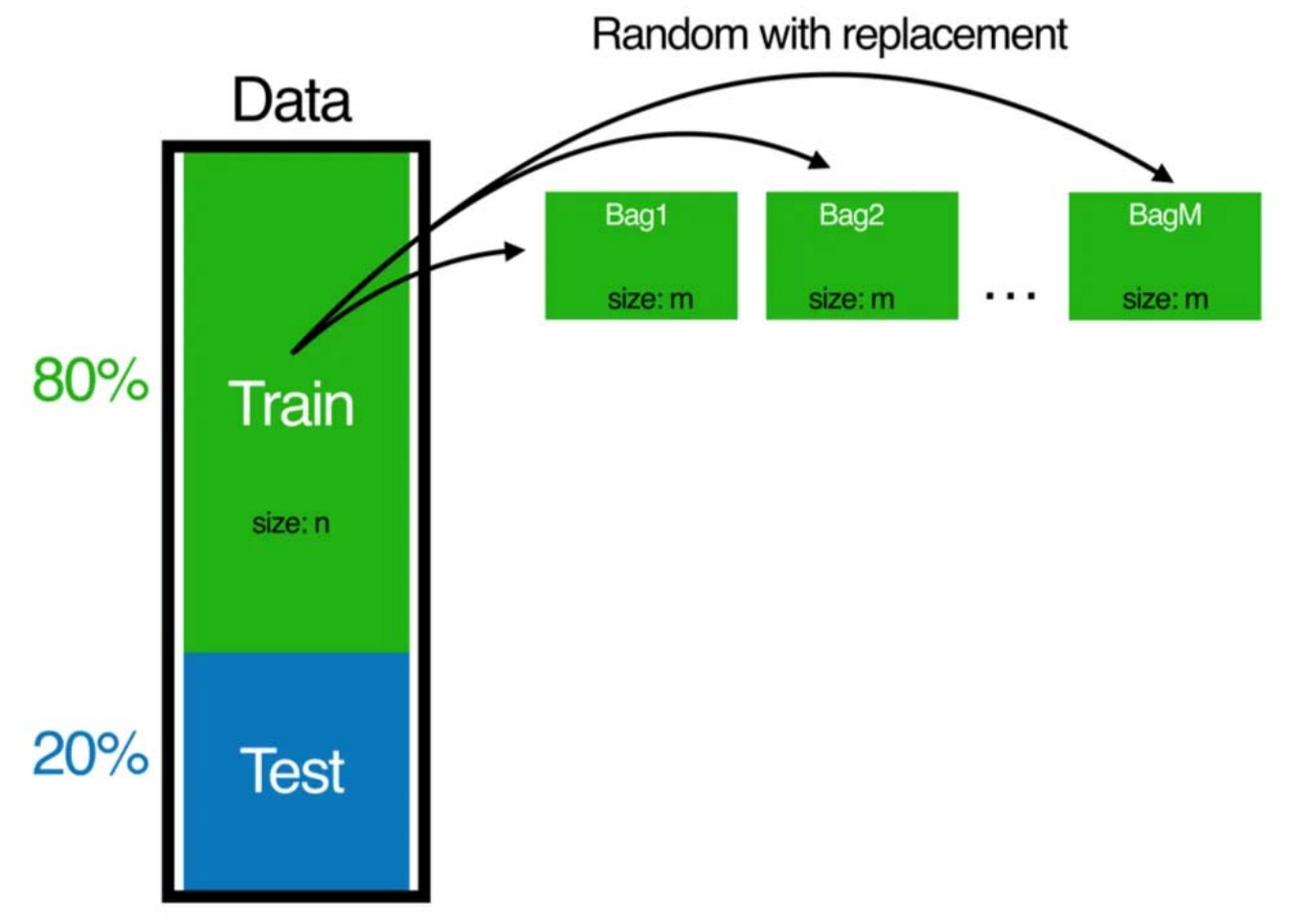


Bootstrap Sampling

With bagging, it is therefore possible that you draw a single training example more than once. This results in a modified version of the training set where some rows are represented multiple times and some are absent. This also lets you create new data, which is similar to the data you started with. By doing this, you can fit many different but similar models.

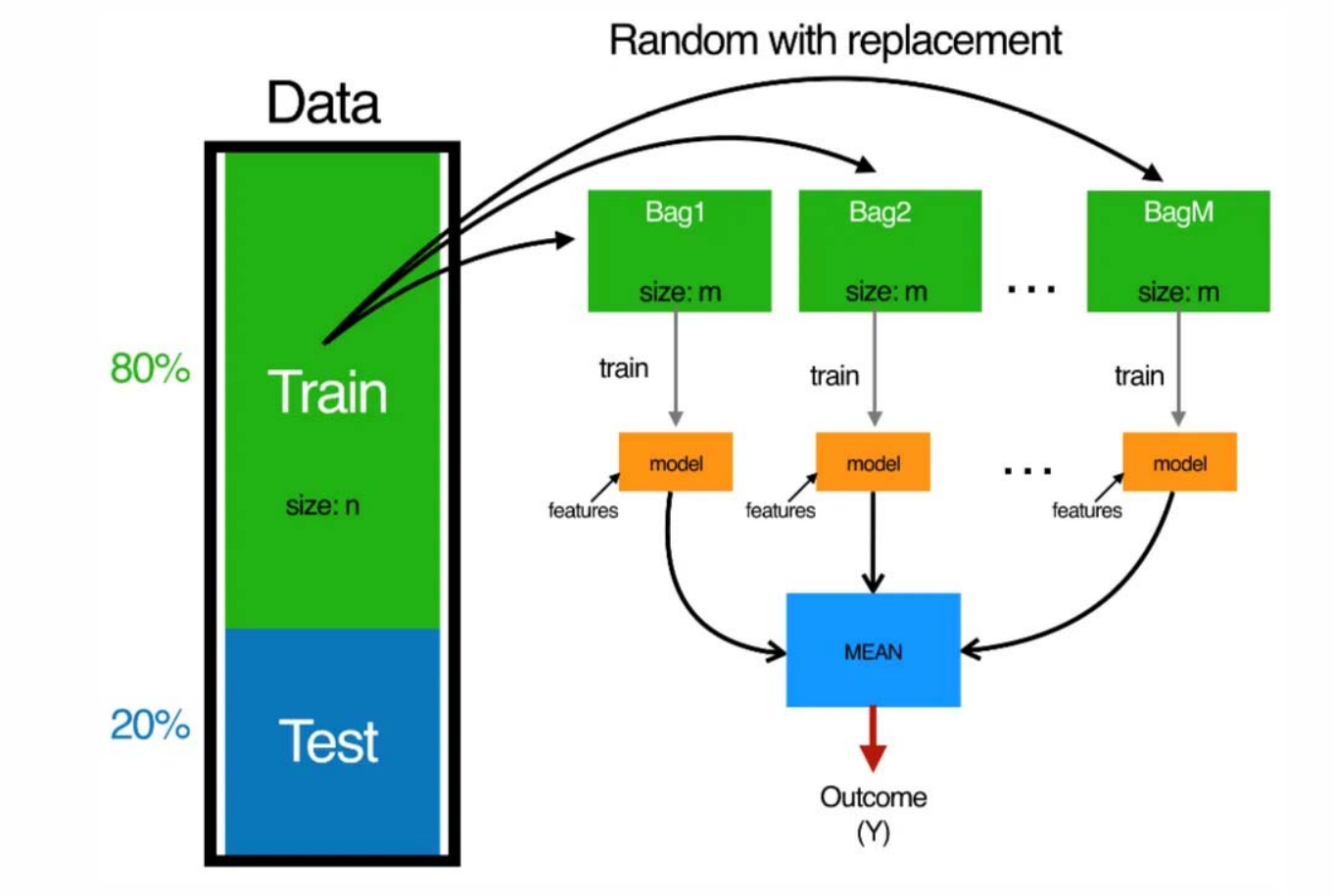
Specifically, the way bagging works is as follows:

Step 1: You draw B samples with replacement from the original data set where B is a number less than or equal to n, the total number of samples in the training set.



Step 1 : Understanding Bagging

Step 2: Train a decision trees on newly created bootstrapped samples. Repeat the Step1 and Step2 any number of times that you like. Generally, higher the number of trees, the better the model. But remember! Excess number of trees can make a model complicated and ultimately lead to overfitting as your model starts seeing relationships in the data that do not exist in the first place.

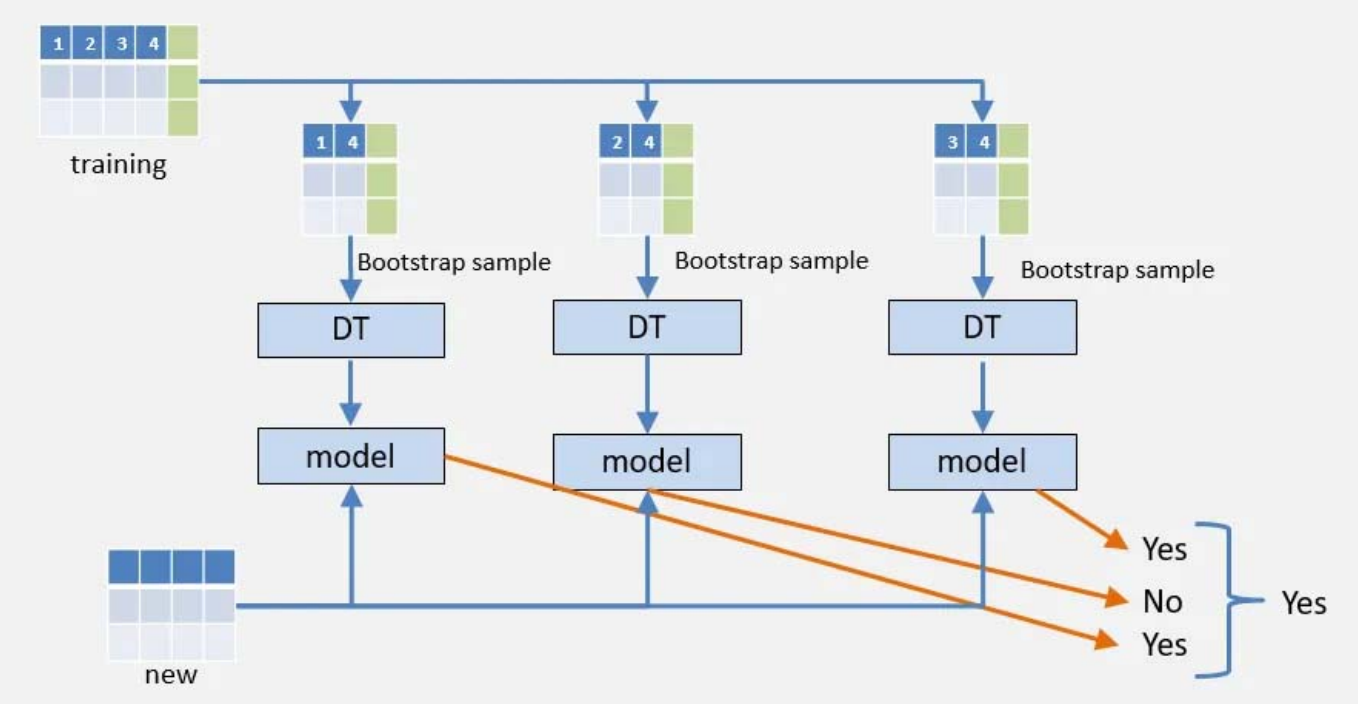


Step 2: Understanding Bagging

To generate a prediction using the bagged trees approach, you have to generate a prediction from each of the decision trees, and then simply average the predictions together to get a final prediction. Bagged or ensemble prediction is the average prediction across the sampled bootstrapped trees. Your bagged trees model works very similar to the council. Usually, when a council needs to take a decision, it simply considers a majority vote. The option that gets more votes (say- option A got 100 votes and option B got 90 votes), is the ultimately the council's final decision. Similarly, in bagging, when you are trying to solve a problem of classification, you are basically taking a majority vote of all your decision trees. And, in case of regression we simply take an average of all our decision tree predictions. The collective knowledge of a diverse set of decision trees typically beats the knowledge of any individual tree. Bagged trees therefore offer better predictive performance.

Random Forests

Random forest is different from the vanilla bagging in just one way. It uses a modified tree learning algorithm that inspects, at each split in the learning process, **a random subset of the features**. We do so to avoid the correlation between the trees. Suppose that we have a very strong predictor in the data set along with a number of other moderately strong predictors, then in the collection of bagged trees, most or all of our decision trees will use the very strong predictor for the first split! All bagged trees will look similar. Hence all the predictions from the bagged trees will be highly correlated.
Correlated
predictors cannot help in improving the accuracy of prediction. By taking a random subset of features, Random Forests systematically avoids correlation and improves model's performance. The example below illustrates how Random Forest algorithm works.



Understanding Random Forests

Let's look at a case when we are trying to solve a classification problem. As evident from the image above, our training data has four features- Feature1, Feature 2, Feature 3 and Feature 4. Now, each of our bootstrapped sample will be trained on a particular subset of features. For example, Decision Tree 1 will be trained on features 1 and 4 . DT2 will be trained on features 2 and 4, and finally DT3 will be trained on features 3 and 4. We will therefore have 3 different models, each trained on a different subset of features. We will finally feed in our new test data into each of these models, and get a unique prediction. The prediction that gets the maximum number of votes will be the ultimate decision of the random forest algorithm. For example, DT1 and DT3 predicted a positive class for a particular instance of our test data, while DT2 predicted a negative class. Since, the positive class got the majority number of votes(2), our random forest will ultimately classify this instance as positive. Again, I would like to stress on how the Random Forest algorithm uses a random subset of features to train several models, each model seeing only specific subset of the dataset.

Random forest is one of the most widely used ensemble learning algorithms. Why is it so effective? The reason is that by using multiple samples of the original dataset, we reduce the variance of the final model. Remember that the low variance means low overfitting. Overfitting happens when our model tries to explain small variations in the dataset because our dataset is just a small sample of the population of all possible examples of the phenomenon we try to model. If we were unlucky with how our training set was sampled, then it could contain some undesirable (but unavoidable) artifacts: noise, outliers and over- or underrepresented examples. By creating multiple random samples with replacement of our training set, we reduce the effect of these artifacts.

Random Forests in Python

In this section, we will see how to implement the Random Forest algorithm in Python. I have detailed all the necessary steps for anyone who is following along (including a couple of data pre-processing tasks). Here is a link to access my Jupyter Notebook.

HarshSingh16/Machine_Learning

Machine Learning Examples. Contribute to HarshSingh16/Machine_Learning development by creating an...

github.com

Data Pre-Processing

We will be using the *bank-full-additional dataset.csv* to perform this implementation. The data set can be accessed at my GitHub at the following link. The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Using features such as age, job, marital, education etc., I have tried to predict whether a given customer has subscribed to a term deposit or not. We will begin by importing the dataset and checking its information.

```
df=pd.read_csv("bank-additional-full.csv")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 20 columns):
age               41188 non-null int64
job               41188 non-null object
marital           41188 non-null object
education         41188 non-null object
default           41188 non-null object
housing           41188 non-null object
loan              41188 non-null object
contact           41188 non-null object
month             41188 non-null object
day_of_week       41188 non-null object
duration          41188 non-null int64
campaign          41188 non-null int64
pdays             41188 non-null int64
previous          41188 non-null int64
poutcome          41188 non-null object
emp.var.rate      41188 non-null float64
cons.price.idx    41188 non-null float64
cons.conf.idx     41188 non-null float64
euribor3m         41188 non-null float64
y                 41188 non-null object
dtypes: float64(4), int64(5), object(11)
memory usage: 6.3+ MB
```

Also, looking at our dataset, we see that some of our features such as *job, education, default, housing, loan, contact, month, day_of_week, poutcome,* are categorical in nature. That means we need to transform them using dummy variables so **sklearn** will be able to understand them. Let's do this in one clean step using **pd.get_dummies.**

```
df.head()
```

job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome
housemaid	married	basic.4y	no	no	no	telephone	may	mon	261	1	999	0	nonexistent
services	married	high.school	unknown	no	no	telephone	may	mon	149	1	999	0	nonexistent
services	married	high.school	no	yes	no	telephone	may	mon	226	1	999	0	nonexistent
admin.	married	basic.6y	no	no	no	telephone	may	mon	151	1	999	0	nonexistent
services	married	high.school	no	no	yes	telephone	may	mon	307	1	999	0	nonexistent

```
#Converting string values to dummies
df_final=pd.get_dummies(df,columns=["day_of_week","job",
                                    "marital","education",
                                    "default","housing",
                                    "loan","contact","month",
                                    "poutcome"],drop_first=True)
```

Converting Categorical Columns to Dummy Variables

We are now ready to split our data into training and test set for ultimately running the Random Forest algorithm. The following lines of code will separate the y-labels from the data frame, and perform the required train-test split. We have kept 30% of the data for testing purposes.

```
import sklearn
from sklearn.model_selection import train_test_split

#Separating the y-column from the rest of the data
df2=df_final.drop("y",axis=1)
X=df2
y=df_final["y"]

#Performing the Split
X_train, X_test, y_train, y_test = train_test_split(X,y , test_size=0.30)
```

Training and Testing our Random Forest Classifier

It is now time to fit our Random Forest Classifier. I have specified the *n_estimators* as 1000, which implies that our classifier will have a total of 1000 trees. For all the other parameters, I have used the default parameter settings. I highly recommend checking out the Random Forest's sklearn documentation if you want to learn more about the parameters RFC provides.

```
#Fitting a Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier

#Sepcifying the random forest-n_estimator specifies the number of trees
rfc=RandomForestClassifier(n_estimators=1000)
rfc.fit(X_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

We will now predict our test data. I have also imported a few more modules to access the confusion matrix, accuracy score and the classification report.

```
#Getting our Predictions
y_hat=rfc.predict(X_test)


#Printing the Accuarcy, Confusion Matrix and the Classification Report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score


y_actu = pd.Series(y_test, name='Actual')
df_confusion = pd.crosstab(y_actu, y_hat)
acc = accuracy_score(y_test, y_hat, normalize=True)
print('Model Accuracy:%.2f'%acc)
print(df_confusion)
print(classification_report(y_test,y_hat))
```

Here are the results: Our model has an overall accuracy of 0.92. I have also printed out the Classification Report and the Confusion Matrix. The overall precision of our model is 0.66, and the recall is 0.50.

```
Model Accuracy:0.92
col_0        no  yes
Actual
no        10637  348
yes         690  682
              precision    recall  f1-score   support

          no       0.94      0.97      0.95     10985
         yes       0.66      0.50      0.57      1372

   micro avg       0.92      0.92      0.92     12357
   macro avg       0.80      0.73      0.76     12357
weighted avg       0.91      0.92      0.91     12357
```

Final Remarks:

I hope that this article helped you to get a basic understanding of how the gradient boosting algorithm works. Feel free to add me on LinkedIn and I look forward to hearing your comments and feedback.

Citations:

As mentioned above, I have used *bank-full-additional dataset.csv* for the purpose of the article. This dataset is publicly available for research. The details are described in [Moro et al., 2014].

[Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22–31, June 2014