Prediction of Diabetes using Random Forest Algorithm
Report for Final Project

(Data Structures and Algorithms)

Ashwin K Channakeshava(ak1645)
Prince Bose(pkb44)
Sarthak Vengurlekar(sv555)

DATE: 04/23/19

DATASET

The native Indian community of the Pima Heritage of Arizona and Mexico faced a prolonged problem of diabetes over the years. These women were surveyed twice, in 1965 and 1991, for symptoms of Type 2 diabetes and Obesity. It was found that this problem was more prominent in women above 21 years. The research of the health conditions of these women was considered crucial for scientific advances regarding the cure of Type 2 Diabetes. Here, we consider the roles of genes of the women and that of the environment they are present in are independent of each other. We obtained a dataset from Kaggle, consisting of details of their medical conditions which are the following:

1. Number of Pregnancies
2. Glucose concentration
3. Diastolic Blood pressure
4. Skin thickness
5. Insulin
6. Body to Mass Index(BMI)
7. Age
8. Outcome (Result = Diabetic(1 or True) / Not Diabetic(0 or False))

From this data, considering the conditions of women who we know suffer from diabetes, we aim to train a model that is capable of predicting whether a Pima Heritage Woman is diabetic or not.

There are quite a few inconsistencies in the data as there would be in any dataset obtained from the real world. Some of these consistencies are as follows:

1. 5 missing glucose concentration entries.
2. 35 missing diastolic blood pressure entries.
3. 227 missing skin thickness entries.
4. 374 missing insulin entries.
5. 11 missing BMI entries.

To tackle these inconsistencies we had the following options:

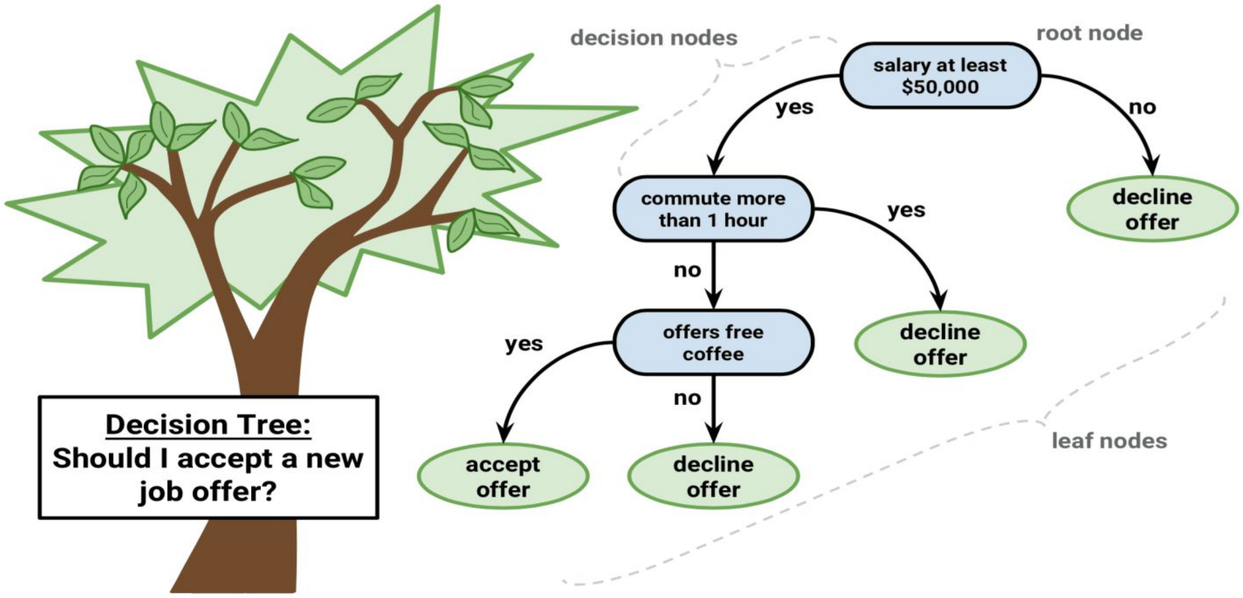
1. Ignore them
2. Delete them
3. Impute values:
    a. Impute with mean value
    b. Replace with computed value

To ignore so many values would decrease the size of our dataset considerably. Deleting so many values also has the same problem and would result in wastage of data. To impute values with computed values would require an expert opinion to decide such values and the data would then depend on the expertise of the individual. The most reasonable option would then be to impute the mean values at all missing values as this option does not have any of the problems mentioned above.

Here, we consider 20% of the data to train the model which we are implementing and the remainder of the dataset for testing of the model.

In order to understand the working of the Random Forest Classifier, we need to know about the basic building block of the Random Forest, Decision Trees.

Decision Trees



Decision trees are the basic building block of the random forest algorithm which we are using to predict diabetes and hence it is important to understand decision trees as the underlying concept before moving on to the algorithm. They are one of the most basic and commonly used machine learning algorithms. Decision trees can be broadly described as multiple branched if structures that lead to the outcome. In layman terms, there are multiple features describes as yes or no questions. Depending on the answer, it depends on which part of the structure the control flow of the algorithm will move to next. This can be understood easily seeing the example above.

Decision trees are Supervised algorithms. This means that the decision algorithms are trained based on predefined datasets before being used on the actual data. They can be expressed as algorithms which contain only conditional control statements. A decision tree is basically made of three components:

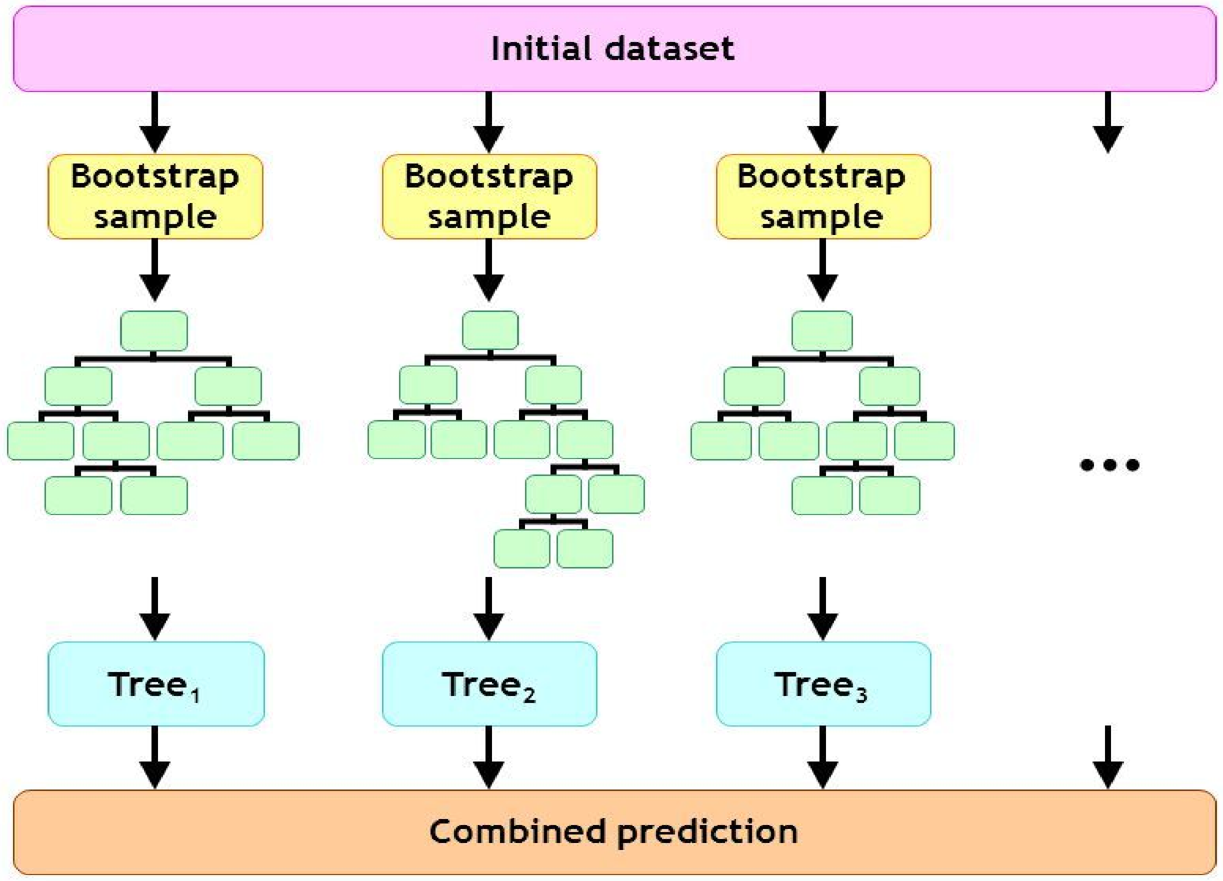
1. The internal node which represents the test on an attribute.
2. The branch represents the outcome of the test.
3. The leaf node represents the class label.

However, the decision trees have certain disadvantages:

1. They are unstable. Small changes are difficult to make once the trees are done designing.
2. They are often inaccurate when compared to other indicators.
3. Calculations can get very complex in cases where the outcomes may be linked.

Since we are considering eight different features for prediction, A single decision tree may get very complex and difficult to understand and the accuracy of the prediction may not be up to our expectations. Hence we use an algorithm which uses multiple decision trees and combines the results and gives a single prediction. Namely, random forest algorithm.

Random Forest Algorithm



The Random Forest Algorithm is a sophisticated and advanced algorithm. It is a supervised algorithm and is flexible. The dataset is randomly bootstrapped into small subsets. Each subset passes through the decision trees independently and n number of results are obtained where n is the number of subsets. Once the results are obtained, the majority of all the results is decided as the combined prediction.

Some advantages of using Random Forest Algorithm are:

1. This algorithm can be used for both classification and regression.
2. It adds additional randomness to the model while growing the trees.

Some advantages of using Random Forest algorithm over Decision trees are:

1. Reduction in overfitting: By averaging multiple decision trees, there is a significantly lower risk of overfitting.
2. Less variance: By using multiple trees , you reduce the chance of stumbling across a classifier that doesn’t perform well because of the relationship between the training and testing data.

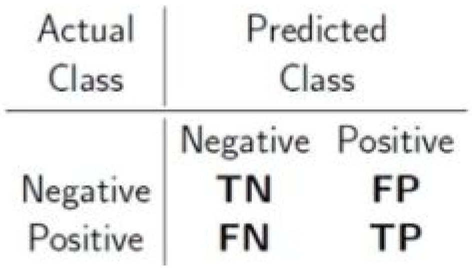
One disadvantage of random forest algorithms is the complexity of formation. They are much more harder and time consuming to construct than decision trees.

Precision and Recall

In a Binary Classification or a Predictive model that classifies the outcome in two possible classifications, positive or negative, there could be multiple cases. There could be cases which are actually positive but are predicted to be negative. Similarly, cases that are negative but are predicted to be positive. There are four possible outcomes in such a situation and they are listed below.

- TN True Negatives
- TP True Positives
- FN False Negatives
- FP False Positives

True Negatives are cases that are actually negative and predicted to be negative, likewise True Positives are cases that are positive and predicted to be positive. False negatives are cases that are positive but predicted to be negative and similarly False Positives are cases that are negative but predicted to be positive.



Actual Class	Predicted Class	
	Negative	Positive
Negative	**TN**	**FP**
Positive	**FN**	**TP**

The data is tabulated in the format given above. This table shows data that is actually positive and actually negative and it also shows data that is predicted to be negative and predicted to be positive. For example, predicted positive results are the FP(False Positive) and TP(True Positive) results collectively. Whereas, the actual positive outcomes are the FN(False Negative) and the TP(True Positive) outcomes collectively.

Accuracy is given by the total number of correct classifications divided by the total number of cases.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

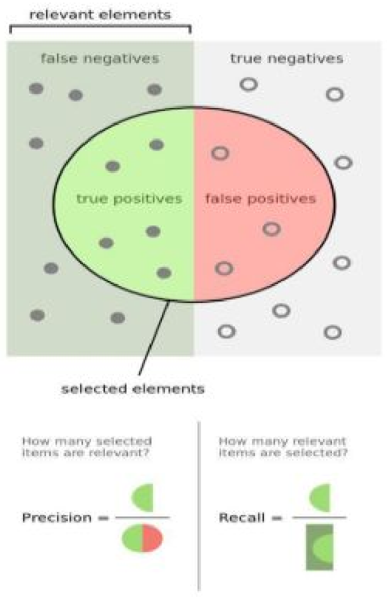
Recall is given by the proportion of correct positive classifications (true positives) from cases that are actually positive.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Precision is given by the proportion of correct positive classifications (true positives) from cases that are predicted as positive.

$$\text{Precision} = \frac{TP}{TP + FP}$$

The diagram below gives a comprehensive understanding of the terms discussed above.



Complexity

Random Forest algorithm basically builds multiple decision trees and merges them to get a better and a stable prediction outcome.

The time it takes for a classification algorithm to traverse through a single tree is log(N), where N is the number of elements in the tree. The time complexity is:

$O(N) = \log(N)$

In Random Forest, we have multiple trees. The time complexity also depends on the number of features that the algorithm is tested upon. Hence, for Random Forest Algorithm to traverse n number of trees for one feature, the time taken is n*log(N). The time complexity is:

$O(N) = n * \log(N)$

In our implementation of the Random Forest, the algorithm is worked on 7 distinct features. There can be k number of features in any implementation of the Random Forest. Hence, for k features, the time taken is k*n*log(N). The time complexity is given as:

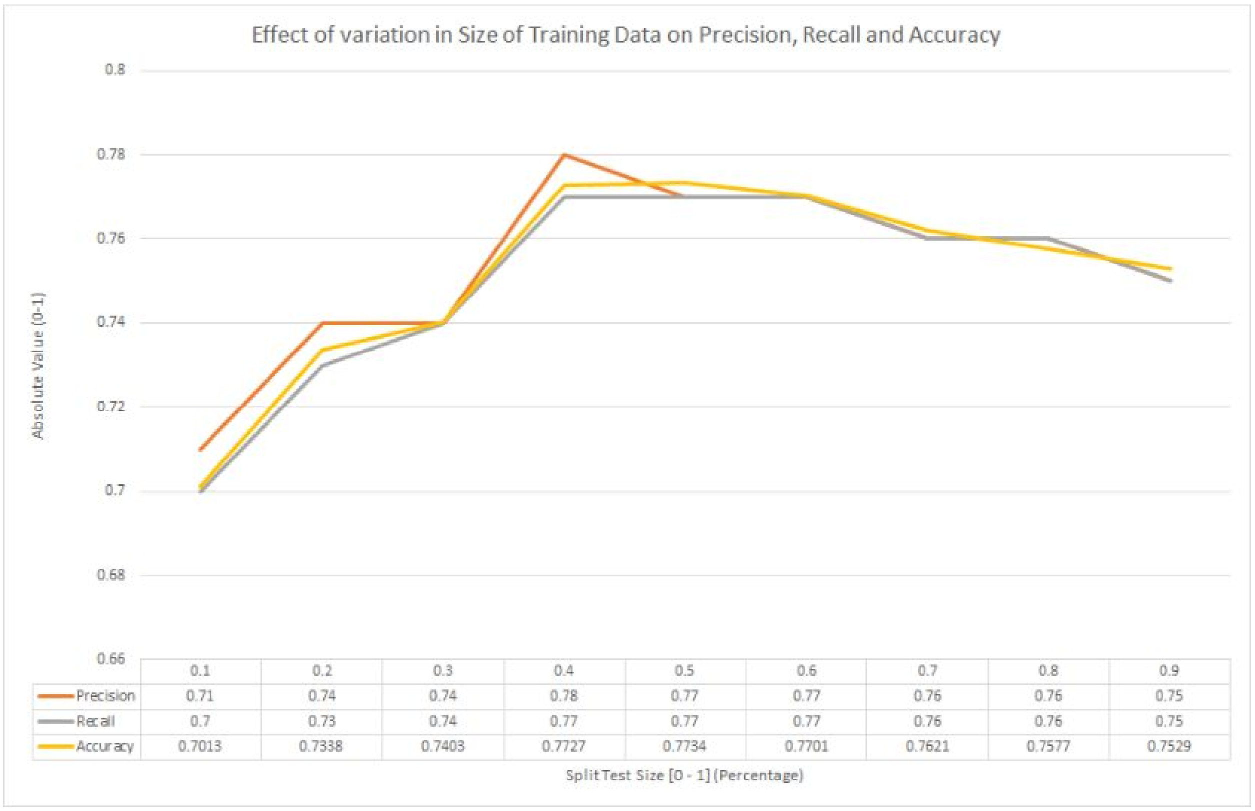
$O(N) = k * n * \log(N)$

RESULTS & ANALYSIS

SELECTION AND OPTIMIZATION OF HYPERPARAMETERS

1. SPLIT SIZE

In order to train our model and perform testing on the data, we need two different datasets of training and testing data. Since we have a limited database of 768 entries, we split the dataset into two parts. The partitioning of data is done using a random subset generator. The size of the subset depends on the split percentage (0 to 1). For example, we if the value of split_test_size is 0.1, that would generate a testing dataset of length equal to 10 percent of the total dataset.
If we select a very low value, we might not have accurate results as we will not have enough data to test our model on, thereby resulting in low accuracy. If we choose a very high value, we will not have enough data to train our model on, resulting in low accuracy. Hence, we ran tests by gradually increasing the split percentage from 0.1 to 0.9 with increments of 0.1, and obtained the following results.



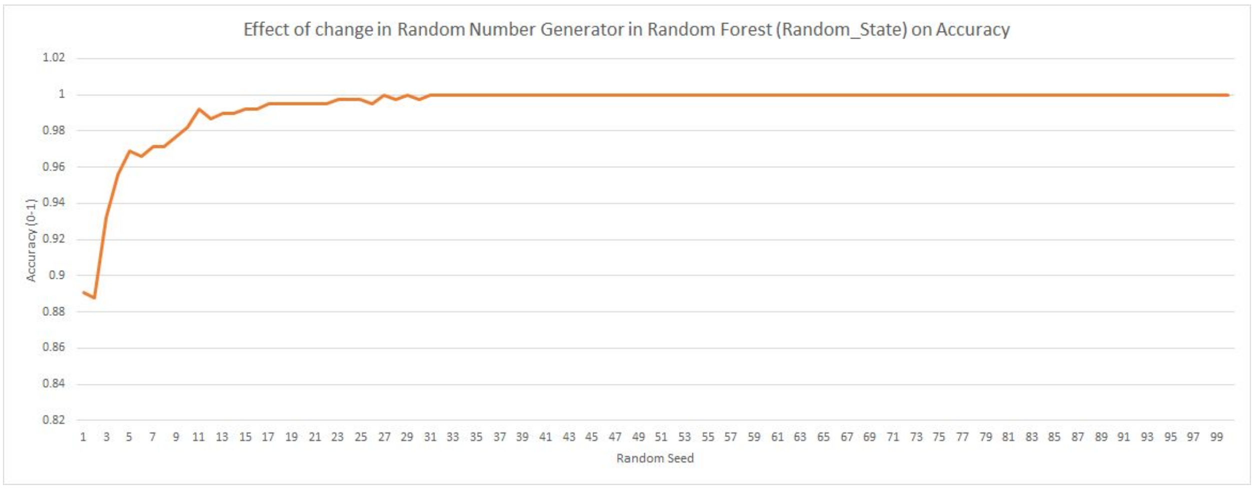
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Precision	0.71	0.74	0.74	0.78	0.77	0.77	0.76	0.76	0.75
Recall	0.7	0.73	0.74	0.77	0.77	0.77	0.76	0.76	0.75
Accuracy	0.7013	0.7338	0.7403	0.7727	0.7734	0.7701	0.7621	0.7577	0.7529

As expected, we achieved low accuracies for very low and very high values. We achieved maximum accuracy for split size = 0.4 and 0.5. But, since 0.4 gave us a better precision for “1” (Positive Outcome), we chose to go ahead with 0.4.

2. RANDOM STATE

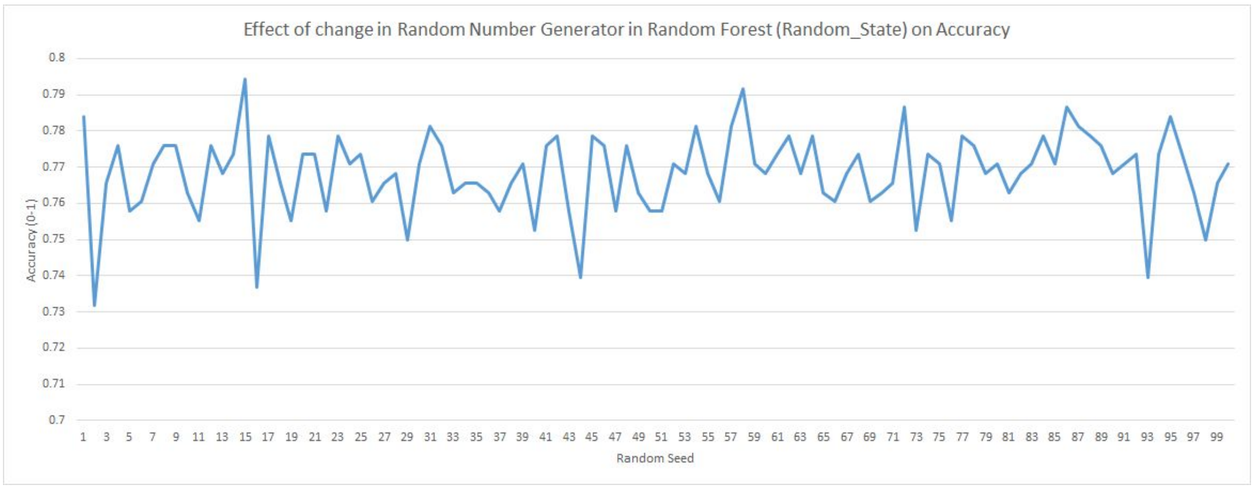
The Random Forest classifier uses a random number generator to generate bootstrap samples as explained in Section >>>>. This random number generator needs a seed value to initialize the pseudo random number generator. This is very crucial for the model, as the randomness and the uniformity of selection is dependent on the seed.

By default, the random_state (name of the variable containing random seed) value is set as 10. We tried to increment the value of the seed gradually 1 through 100 and measured the accuracy for each value. The results are shown in the graph below.



The accuracy of the model on the training dataset itself kept improving till the value was 32, after which it became constant at 1.00, i.e. 100% accuracy. Hence we could choose any value above 32. We collectively decided to choose 42 as the random seed.

The effect of change in random seed on the accuracy of testing data was random. The graph can be seen below.

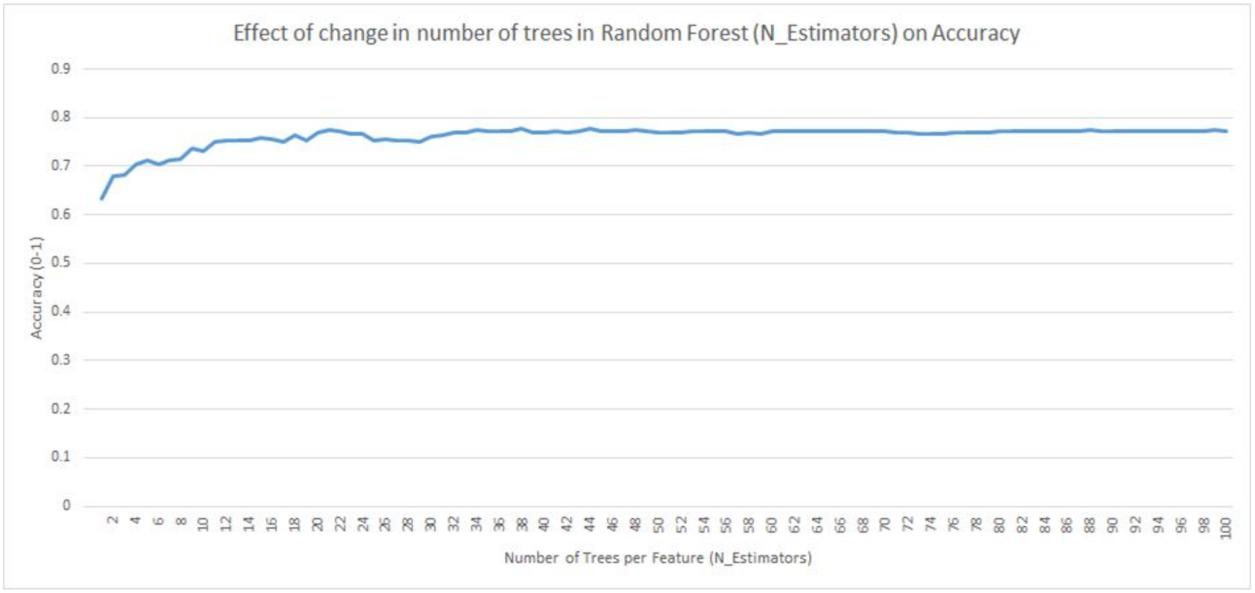


Hence we decided to make a choice of seed based on the accuracy of the model on the training data.

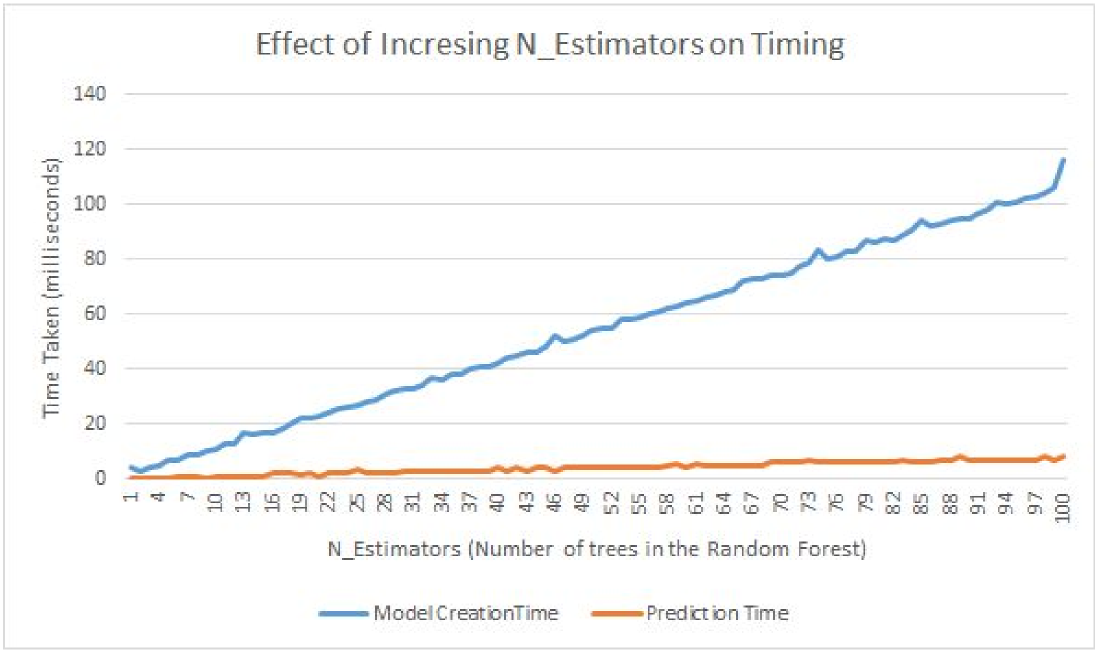
3. NUMBER OF TREES IN THE RANDOM FOREST (N_ESTIMATORS)

The accuracy of the Random Forest classification algorithm is dependent on the number of trees generated per feature.This is because, in order to get an outcome, the majority of the outcome of all trees is taken into account. Hence in order to determine the number of trees per feature, RandomForestClassifier has a hyperparameter called N_Estimators. The default value is set at 10 for version 2.0.0 and at 100 for version 2.0.2.

We wanted to optimize the model for better accuracy and timing. Hence we iterated N_Estimators from 1 through 100. The effect on accuracy was as seen in the graph below.



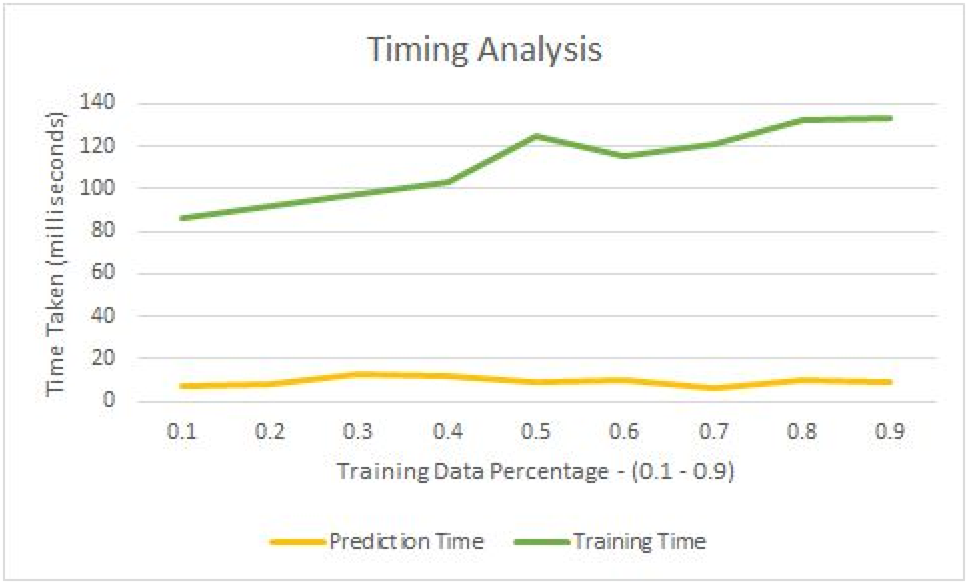
The Accuracy increases gradually till the value hits 13. After 13, the Accuracy has very slight variations till 100, showing no major effect or change. Hence we choose the value of 100 as more the number of trees, more accurate the prediction. But, there is a tradeoff, in the form of training time. The time required to train and predict rises. This effect is shown in the graph below.



We choose to accept this rise in the Prediction Time, as we need to choose accuracy over time for our model.

4. Effect of Increasing Training Data Size on the Timing

The effect of increasing the training data percentage on the run time for model creation and prediction can be seen as below. As we had a limited data set, we cannot justify the $O(N) = k*N*Log_2N$ timing claim. But we can see that as the dataset increases, the model creation and training timing also increases.



DISCUSSION

This implementation can further be improved by implementing Logistic Regression with Cross Validation.

If we had more data, we could experiment more on the accuracy and improving the timing of creation of model.

REFERENCES

- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- https://www.kaggle.com/uciml/pima-indians-diabetes-database
- https://towardsdatascience.com/decision-trees-and-random-forests-for-classification-and-regression-pt-1-dbb65a458df
- http://www.math.usu.edu/adele/randomforests/ovronnaz.pdf
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6050737/