# FoML assignment III : Neural Networks, Boosting, XGBoost, Random Forests

Tamal Mondal

October 29, 2021

## 1 Neural Networks Theory Question 1

### 1.1 Solving XOR problem using a two-layer perceptron

The XOR problem have a non-linear decision boundary(unlike OR, AND and NAND) and so it can't be solved using a single layer perceptron as it can only create linear decision boundary. Using a hidden layer with two neurons the problem can be solved and Figure 1 shows the decision boundary for that. The intuition here is that the two neurons in the hidden layer are used to create the two separate decision boundary $L_1$ and $L_2$, the third(output) neuron is used to combine the two results.

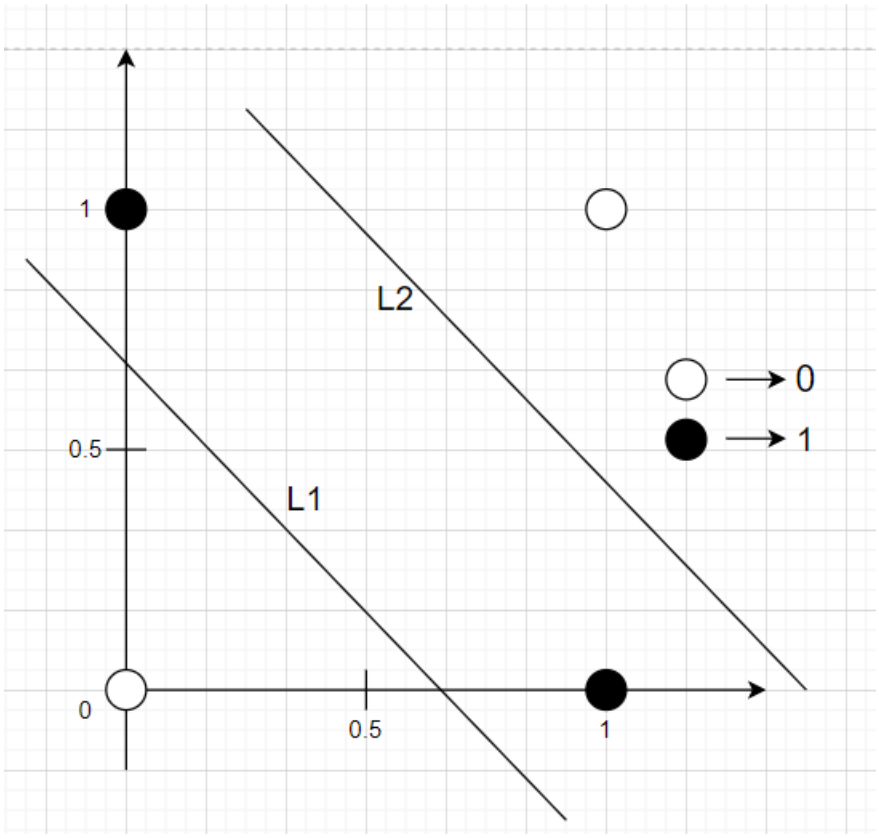


Figure 1: XOR decision boundary

As we can write, $XOR(a, b) = (a + b).(\overline{a.b})$, intuitively the three neurons actually does the OR, NAND and finally the AND operation.

Figure 2 shows the architecture of the neural network with the weights and biases to solve the XOR problem.

Here $N_{11}, N_{12}$ *and* $N_{21}$ are the three neurons and $S_{11}, S_{12}$ *and* $S_{21}$ represents sigmoid activation. Additionally, $P_1$ and $P_2$ are the inputs and $b_{11}, b_{12}$ *and* $b_{21}$ are the biases.

Here $a_{11} = \sigma(2P_1 + 2P_2 - 1)$, $a_{12} = \sigma(-P_1 - P_2 + 1.5)$ and final output $= \sigma(a_{11} + a_{12} - 1.5)$.

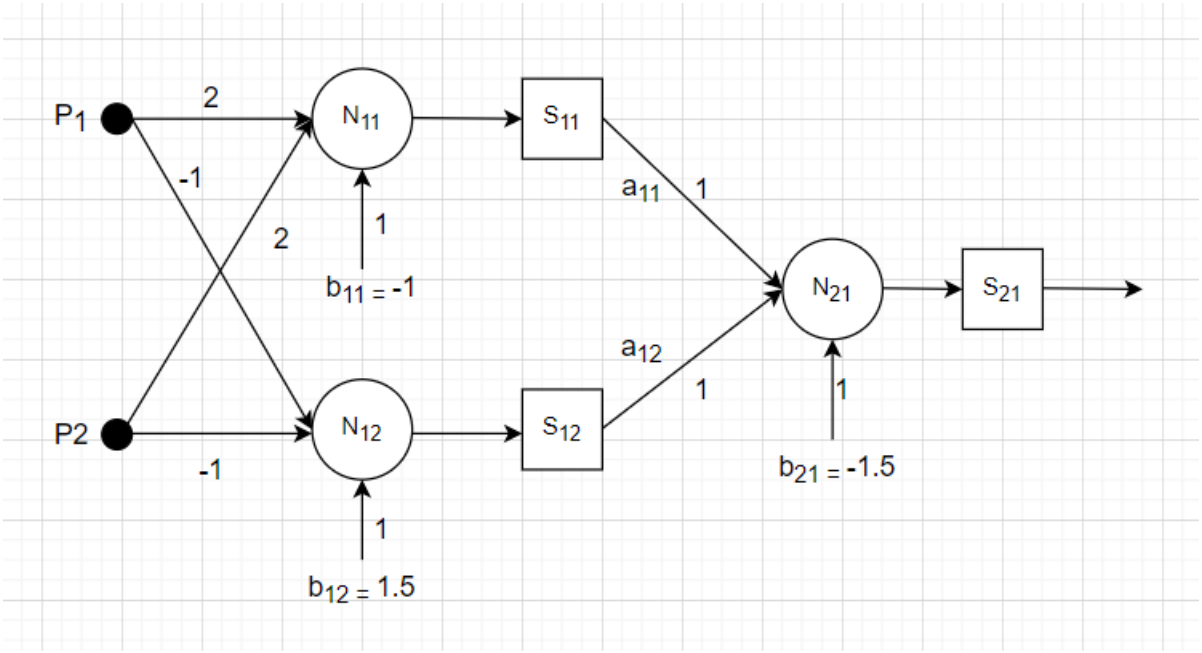


Figure 2: 2 layer perceptron to solve XOR problem

### 1.2 Find the gradient

Given that x = -2, y = 5 and z = -4. Also given that for the two neurons q and f, q = x - y and f = q * z. Figure 3 shows the graphical representation of the network and the gradients of f with respect to x, y and z are as follows:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}.\frac{\partial q}{\partial x} = z.(1) = z = -4$$
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}.\frac{\partial q}{\partial y} = z.(-1) = -z = 4$$
$$\frac{\partial f}{\partial z} = q = x - y = -7$$

## 2 Derivative of cross-entropy error function

If $y_i$ is the soft-max probability for the ith neuron and it's defined as follows where $a_j$'s are the pre-softmax activations of the output layer neurons (also called logits).

$$y_i = \frac{e^{a_i}}{\sum_j e^{a_j}}$$

Now we need to find $\frac{\partial y_i}{\partial a_k}$ and for that we will take help of "logarithmic derivative":

$$\frac{\partial \log y_i}{\partial a_k} = \frac{1}{y_i}.\frac{\partial y_i}{\partial a_k}$$

Now we can find $\frac{\partial y_i}{\partial a_k}$ as follows:

$$
\begin{aligned}
\frac{\partial y_i}{\partial a_k} &= y_i.\frac{\partial \log y_i}{\partial a_k} \\
&= y_i.\frac{\partial(\log(e^{a_i}) - \log(\sum_j e^{a_j}))}{\partial a_k} \text{ (As we know } y_i = \frac{e^{a_i}}{\sum_j e^{a_j}}) \\
&= y_i.\frac{\partial(a_i - \log(\sum_j e^{a_j}))}{\partial a_k} \\
&= y_i.\left(\frac{\partial a_i}{\partial a_k} - \frac{\partial(\log \sum_j e^{a_j})}{\partial a_k}\right) \\
&= y_i.\left(1\{i=k\} - \frac{e^{a_k}}{\sum_j e^{a_j}}\right) \text{ (Here, } 1\{.\} \text{ is an indicator function, returns 1 when i=k(so, } a_i = a_k) \text{ otherwise 0)} \\
&= y_i.\left(1\{i=k\} - y_k\right)
\end{aligned}
$$

Now we know the cross-entropy error function for multi-class classification can be written as follows:

$$E = -\sum_{i=1}^{K} t_i \ln y_i$$

where K is number of class, $t_i$ is the value at ith position of ground truth class(in form of one-hot vector) and $y_i$ is ith softmax output.

Finally we are ready to find the derivative of the above error function with respect to the activation $a_k$ for an output unit:

$$
\begin{aligned}
\frac{\partial E}{\partial a_k} &= -\frac{\partial\left(\sum_{i=1}^{K} t_i \ln y_i\right)}{\partial a_k} \\
&= -\sum_{i=1}^{K} t_i.\frac{\partial(\ln y_i)}{\partial a_k} \\
&= -\sum_{i=1}^{K} \frac{t_i}{y_i}.\frac{\partial y_i}{\partial a_k} \\
&= -\sum_{i=1}^{K} \frac{t_i}{y_i}.y_i.\left(1\{i=k\} - y_k\right) \quad \text{(using previous derivation of } \frac{\partial y_i}{\partial a_k}) \\
&= -\sum_{i=1,i\neq k}^{K} t_i.\left(1\{i=k\} - y_k\right) - t_k.(1\{k=k\} - y_k) \quad \text{(simply divided into cases where } i \neq k \text{ and } i = k) \\
&= \sum_{i=1,i\neq k}^{K} t_i.y_k - t_k.(1 - y_k) \quad \text{(As, } 1\{\text{i=k}\} = 1 \text{ only when k=i and otherwise 0)} \\
&= \left(\sum_{i=1,i\neq k}^{K} t_i.y_k + t_k.y_k\right) - t_k \\
&= \sum_{i=1}^{K} t_i.y_k - t_k \\
&= y_k.\sum_{i=1}^{K} t_i - t_k \\
&= y_k - t_k \quad \text{(As, t is one-hot representation of a class, sum of } t_i \text{ would be 1)}
\end{aligned}
$$

# 3 Ensemble Methods

When $f(x) = x^2$, we need to prove that for ensemble methods, The expected error($E_{ENS}$) $\leq$ the average expected sum-of-squares error($E_{AV}$). Given in the question:

$$E_{AV} = \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_x[(y_m(x) - f(x))^2] \tag{1}$$

and

$$E_{ENS} = \mathbb{E}_x[(\frac{1}{M}\sum_{m=1}^{M} y_m(x) - f(x))^2] \tag{2}$$

Now when $f(x) = x^2$, we can write $\mathbb{E}[f(x)] = Var(x) + (\mathbb{E}[x])^2 \geq (\mathbb{E}[x])^2 \geq f(\mathbb{E}[x])$ as Var(x) $\geq 0$. This inequality for expectation of f(x) would help us later.

Now we will start the proof from the 2nd equation of $E_{ENS}$ and show that it's no larger than $E_{EV}$:

$$\begin{aligned}
E_{ENS} &= \mathbb{E}_x[(\frac{1}{M}\sum_{m=1}^{M} y_m(x) - f(x))^2] \\
&= \frac{1}{N}\sum_{j=1}^{N}\left(\frac{1}{M}\sum_{m=1}^{M}\hat{y}_{mj} - y_j\right)^2 \text{ (just simplified the notations with proper subscripts)} \\
&= \frac{1}{N}\sum_{j=1}^{N}\left(\frac{1}{M}\sum_{m=1}^{M}(\hat{y}_{mj} - y_j)\right)^2 \\
&\leq \frac{1}{N}\sum_{j=1}^{N}\frac{1}{M}\sum_{m=1}^{M}(\hat{y}_{mj} - y_j)^2 \text{ (As we have already showed } \mathbb{E}[f(x)] \geq f(\mathbb{E}[x]) \text{ when } f(x) = x^2) \\
&\leq \frac{1}{M}\sum_{m=1}^{M}\frac{1}{N}\sum_{j=1}^{N}(\hat{y}_{mj} - y_j)^2 \\
&\leq \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_x[(y_m(x) - f(x))^2] \text{ (changed back to the notation given in question)} \\
&\leq E_{EV} \text{ (proved)}
\end{aligned}$$

Now for the **2nd part of the question**, we need to show that above inequality holds for any error function E(y), not just sum-of-squares, as long as it is convex in y.

Now, If f(y) is a convex function on $R_y$, and E[f(y)] and f(E[y]) are finite, then $E[f(y)] \geq f(E[y])$. This is well-known **Jensen's Inequality**. If we closely observe, we can see that we actually showed that $f(x) = x^2$ satisfies Jensen's Inequality and it's a convex function.

So, for any convex function(like $x^4$, $e^x$ etc.) to prove the inequality $E_{ENS} \leq E_{AV}$, we can use the Jensen's Inequality $\mathbb{E}[f(x)] \geq f(\mathbb{E}[x])$ like we have used for $f(x) = x^2$. So the inequality $E_{ENS} \leq E_{EV}$ holds for any error function as long as that is convex.

# 4 Random Forest programming question

## 4.1 Model and Accuracy comparison

I have tested the Random forest classifier(made from scratch) with the following configuration:

number of decision trees = **10**
criterion = **"entropy"**

number of features used to find best split = **30**(total 57 features are there)
number of samples drawn from training input(bootstrapping) = **2000**(total 3220 training samples are there)

For this model, time taken during training and testing was **499 secs** and **94 msecs**. Out-of-bag(OOB) and testing accuracy was **92.7%** and **93.7%** respectively.

Using Random forest classifier from Sklearn library with the same configuration, time taken during training and testing was **170 msec** and **7.7 msecs**. Out-of-bag(OOB) and testing accuracy was **92.7%** and **94%** respectively.

### 4.2 Change in Sensitivity for change in number of features(m)

For analysis, I have used the same configuration mentioned previously and only altered number of features(m). I have observed that Sensitivity was always around **0.91** for different values of m and the best was **0.925** for **m = 40**.

As we have total **57** features, I have checked and plotted Sensitivity for **m = 5,10,15,..,55**. Figure 4 is the graph for Sensitivity vs Number of features(m).

### 4.3 Change in OOB and test error for change in number of features(m)

For analysis, I have used the same configuration mentioned previously and only altered number of features(m). I have checked and plotted both OOB and Test error for **m = 5,10,15,..,55**. Figure 5 is the graph for OOB and Test error vs number of features(m).

As we can see, the best OOB and test accuracy was **92%** and **93%** respectively. We can see that on average OOB and test accuracy was around **91%** and **92%** which is pretty much maintained across the different values of m.

## 5 Gradient boosting programming question

### 5.1 Data preprocessing

Initial shape of the training and test dataset are **(24999, 111)** and **(14718, 111)** respectively. As instructed in the question, I have removed the samples having "loan_status" as "Current". Additionally for "loan_status" column, I have changed "Fully Paid" to 1 and "Charged Off" to -1. I have applied following preprocessing steps on this data.

#### 5.1.1 Handling missing values

After analysis I have found there are **57** columns(like "verification_status_joint", "all_util" etc.) having **more than 15000** NULL values in training dataset(out of 24301 samples so more than half), so I have removed these columns from both training and test dataset. Columns "emp_title", "desc" had some missing values and also seems not important to use for prediction, so I removed these 2 columns additionally.

Finally there are few more columns(like "pub_rec_bankruptcies", "tax_liens" etc.) which had very few NULL values**(less than 2%)**, so I have removed those rows from both training and test dataset as we have sufficient number of samples. After handling missing values, final shape of the training and test dataset are **(23763, 52)** and **(12962, 52)**.

#### 5.1.2 Remove unnecessary columns

After analysing further, I found **18** columns which are not useful features to make prediction. There are columns like "id", "member_id", "url" etc. which **doesn't have useful information**. Then there are columns(like "initial_list_status" and "application_type" etc.) which **have same value for all samples**.

Also there are columns(like "last_pymnt_d", "issue_d" etc.) which **have date in form of dd-MM format** and doesn't seem relevant to decide on loan lending. These 18 irrelevant columns I have removed from training and test data. After that, final shape of the training and test dataset are **(23763, 34)** and **(12962, 34)**.

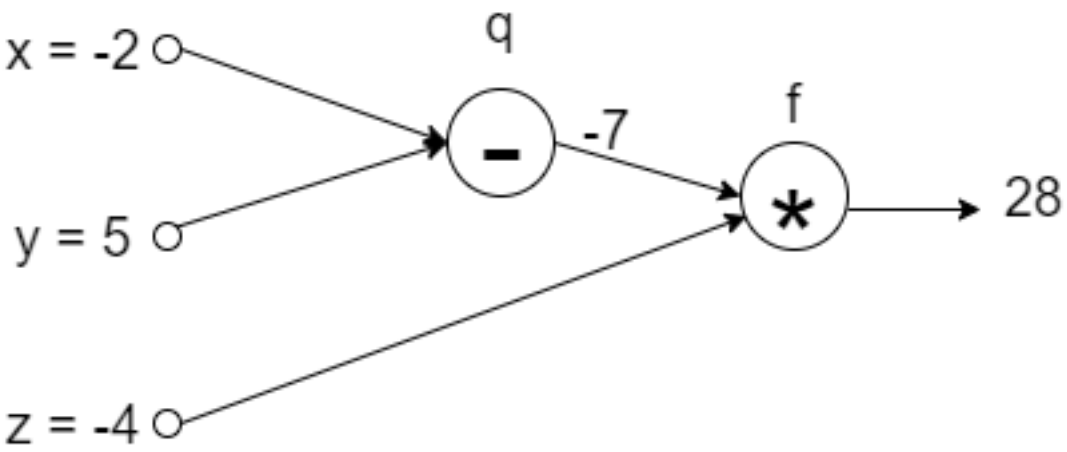


Figure 3: Graphical representation for given inputs

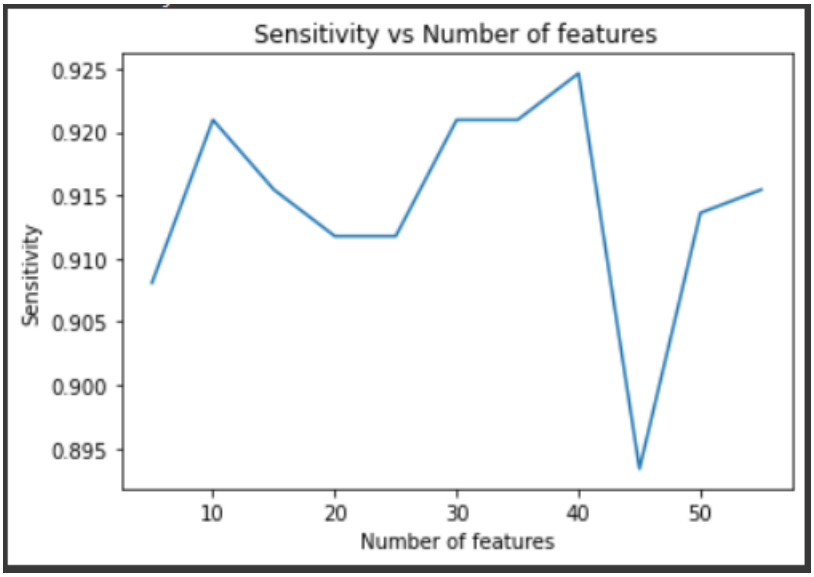


Figure 4: Sensitivity vs Number of features(m)

#### 5.1.3 Encode categorical and ordinal features

I have used sklearn **OrdinalEncoder** for ordinal features like "grade", "sub_grade", "home_ownership" etc. Columns "title"(more than 12k unique values) and "zip_code"(772 unique values) which may not be ordinal in nature but have too many unique values to use one-hot encoding, so I used ordinal encoding.

For the columns "purpose" and "addr_state", I have used one-hot encoding(**OneHotEncoder** from sklearn). After encoding, final shape of the training and test dataset are **(23763, 92)** and **(12962, 92)**.

#### 5.1.4 Any additional processing

Columns "int_rate" and "revol_util" were in string format and had "%". I have removed "%" and converted these 2 columns to float.

### 5.2 Results and comparison

#### 5.2.1 Best results

The best accuracy, precision and recall that I have got are **0.9981**, **0.9979** and **0.9999** respectively. For the skleran GradientBoostingClassifier, I have used the following hyper-parameters that gave best results:

loss = **"exponential"**
learning_rate = **0.8**
n_estimators = **200**
random_state = **42**
max_features = **"auto"**

For rest of the hyper-parameters I have used the default values.

#### 5.2.2 Effect of increasing number of trees

To analyze the effect of increasing number of trees(estimators), I have used the same configuration as mentioned previously and only altered "n_estimators" parameter which indicates number of trees. I have used **n_estimators = 5,15,25,...,245** and plotted the graph for accuracy, precision and recall. Figure 6 is the accuracy/precision/recall vs number of trees graph.

We can see when **n_estimators = 5**, then accuracy, precision and recall are **0.98**, **0.98** and **1** respectively and after that the values are always around **0.99** and there are very slight improvement with increase in number of trees.

#### 5.2.3 Performance comparison with simple decision tree

Using sklearn DecisionTreeClassifier with default configurartion, I have got accuracy, precision and recall as **0.987**, **0.995** and **0.989** respectively. So comparing with the best results for GradientBoostingClassifier as reported earlier, we can see **gradient decent results were little better than simple decision tree but the difference is very small** and both the classifiers performed pretty well on the preprocessed dataset.

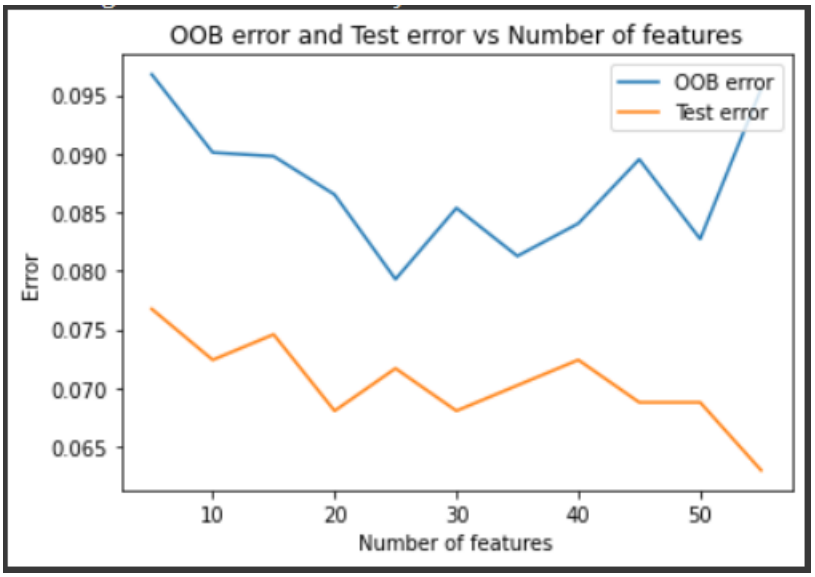


Figure 5: OOB and Test error vs Number of features(m)

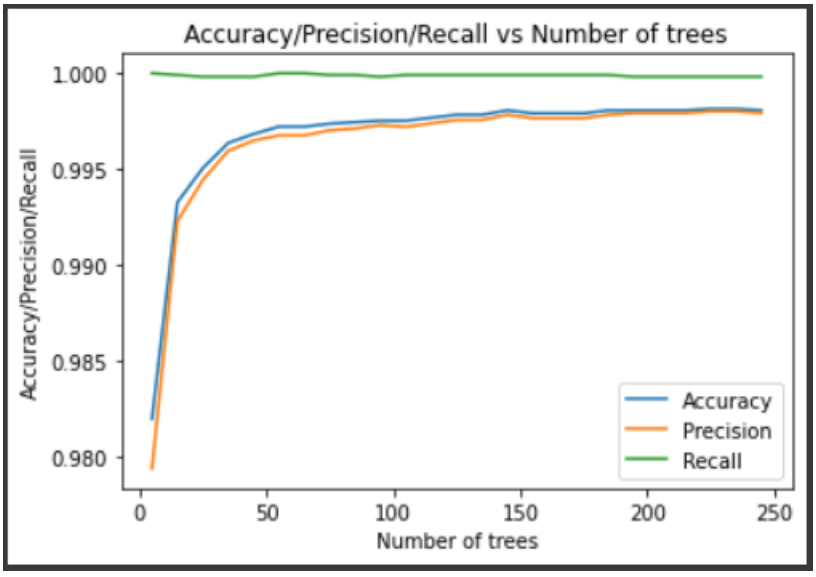


Figure 6: Accuracy/Precision/Recall vs Number of trees