CS5063: Foundations of Machine Learning

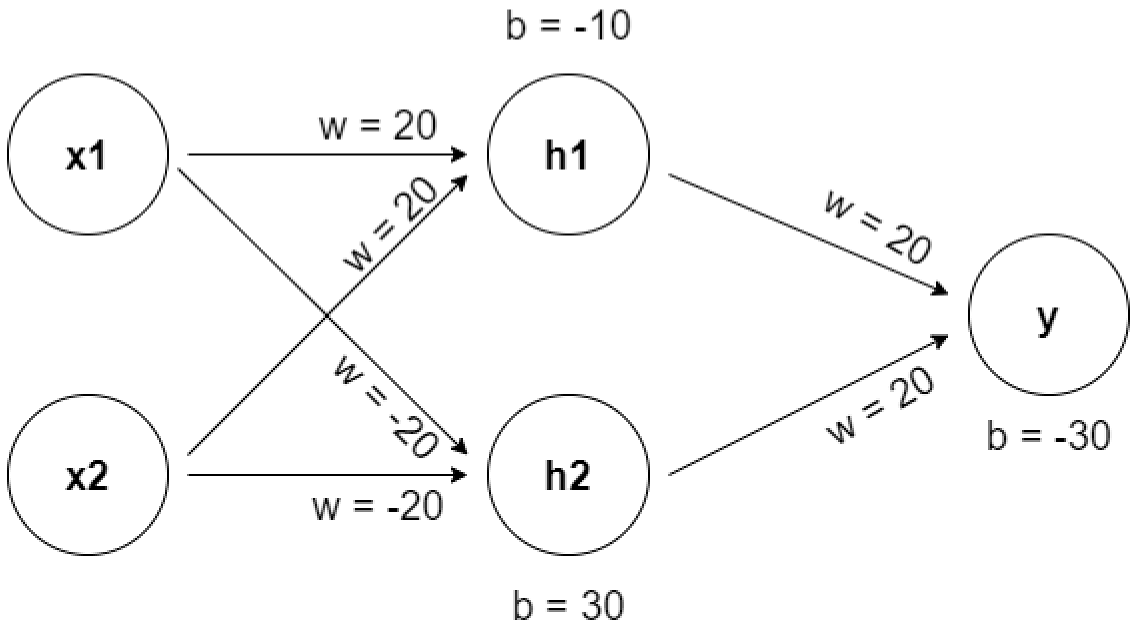
Assignment 3

Submitted By:
Vishal Singh Yadav – CS20MTECH01001

1. An XOR function has the following truth table:

X1	X2	O
0	0	0
0	1	1
1	0	1
1	1	0

We can create a network with one hidden layer using the following network:



Let *x1*, *x2* be the input layer. *h1*,*h2* are hidden layers, and *y* is the output layer. *w* denotes the weight of each connection in the network. *b* represents the bias of each node. *h1*, *h2* and *y* use the sigmoid function as their activation function.

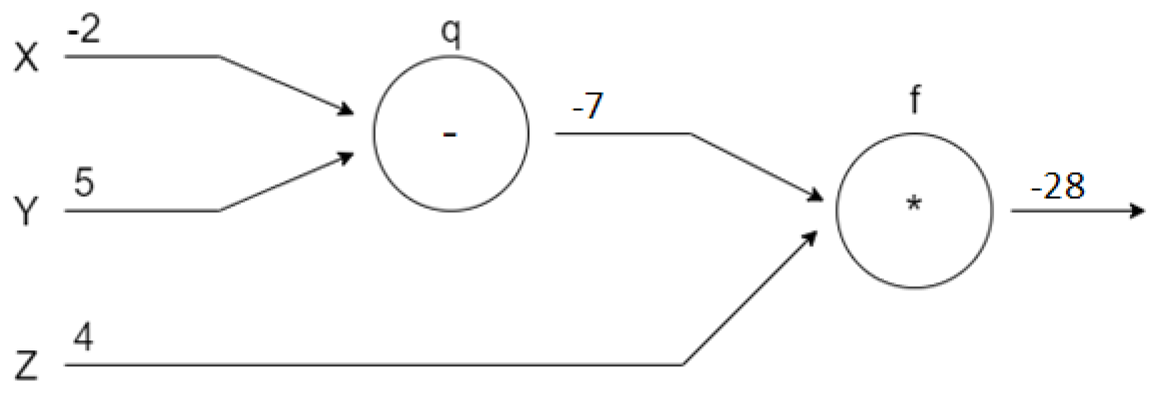
So, $h1 = \rho(20x1 + 20x2 - 10)$, $h2 = \rho(-20x1 - 20x2 + 30)$, and $y = \rho(20h1 + 20h2 - 30)$

Using the XOR value table above, put the values in the network,

X1	X2	h1	h2	y
0	0	ρ(20 * 0 + 20 * 0 - 10) = 0	ρ(-20 * 0 - 20 * 0 + 30) = 1	ρ(20 * 0 + 20 * 1 - 30) = 0
0	1	ρ(20 * 0 + 20 * 1 - 10) = 1	ρ(-20 * 0 - 20 * 1 + 30) = 1	ρ(20 * 1 + 20 * 1 - 30) = 1
1	0	ρ(20 * 1 + 20 * 0 - 10) = 1	ρ(-20 * 1 - 20 * 0 + 30) = 1	ρ(20 * 1 + 20 * 1 - 30) = 1
1	1	ρ(20 * 1 + 20 *1 - 10) = 1	ρ(-20 * 1 - 20 * 1 + 30) = 0	ρ(20 * 1 + 20 * 0 - 30) = 0

$$f = q * z, \qquad q = x - y$$

The graphical representation of f can be shown as:



The derivatives of *f* wrt *x*, *y*, and *z* can be calculated as

$$\frac{df}{dx} = \frac{\delta f}{\delta q}\frac{\delta q}{\delta x} \Rightarrow z.1 \Rightarrow z \Rightarrow 4$$

$$\frac{df}{dy} = \frac{\delta f}{\delta q}\frac{\delta q}{\delta y} \Rightarrow z.1 \Rightarrow z \Rightarrow 4$$

$$\frac{df}{dz} = q \Rightarrow x - y \Rightarrow -2 - 5 \Rightarrow -7$$

2.

$$E(w) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{kn} \ln y_k(x_n, w)$$

$$y_k(x, w) = \frac{\exp(a_k(x,w))}{\sum_j \exp(a_k(x,w))}$$

first we differentiate $y_k = h(a_k) = \text{softmax}(a_k)$ wrt a_k

$$\frac{\partial y_k}{\partial a_k} = \frac{\partial}{\partial a_k} \text{softmax}(a_k)$$

$$= \frac{\partial}{\partial a_k} \left(\frac{\exp a_k}{\sum_j \exp a_j} \right)$$

$$= y_k \frac{\sum_{j \neq k} \exp a_j}{\sum_j \exp a_j} \quad = y_k (1 - y_k)$$

Now differentiate $y_k = \text{softmax}(a_k)$ wrt a_i

$$\frac{\partial y_k}{\partial a_i} = \frac{\partial}{\partial a_i} \text{softmax}(a_k) = \frac{-\exp(a_k)\exp(a_i)}{(\sum_j \exp a_j)^2}$$

$$= -y_i y_k$$

Summarizing the results using kronecker delta function,

$$\frac{\partial y_k}{\partial a_i} = \frac{\partial}{\partial a_i} \text{softmax}(a_k) = y_k (\delta_{kj} - y_i)$$

lets define error $E(w)$ as function of independent parameters.

$$E(w) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_k(x_n, w)$$

$$= \sum_{n=1}^{N} t_{n1} \ln y_{n1} + t_{n2} \ln y_{n2} + \cdots + t_{nK} \ln y_{nK}$$

$$= -\sum_{n=1}^{N} \left(\sum_{k=1}^{K-1} t_{nk} \ln y_{nk} + t_K \ln \left(1 - \sum_{k=1}^{K-1} y_{nk}\right) \right)$$

since $\sum_{k=1}^{K} y_{nk} = 1$ for every n, so first $K-1$ parameters determine ~~last one~~ K^{th} parameter

Differentiating wrt a_j.

$$\partial_{a_j} E(w) = -\left[\sum_{k \neq j}^{K-1} t_k \frac{1}{y_k} \partial_{a_j} y_k + t_j \frac{1}{y_j} \partial_{a_j} y_j \right.$$

$$\left. - t_K \frac{1}{\left(1 - \sum_{k=1}^{K-1} y_k\right)} \partial_{a_j} \left(1 - \sum_{k=1}^{K-1} y_k\right) \right]$$

Now simplify

$$\partial a_j E(w) = -\left[\sum_{k \neq j}^{K-1} t_k \frac{1}{y_k} (-y_k y_j) + t_j \frac{1}{y_j} y_j (1-y_j) - t_K \frac{1}{\left(1-\sum_{k=1}^{K-1} y_k\right)} \partial a_j \left(1 - \sum_{k=1}^{K-1} y_k\right) \right]$$

$$= -\left[\sum_{k \neq j}^{K-1} t_k (-y_j) + t_j (1-y_j) - t_K \frac{1}{\left(1-\sum_{k=1}^{K-1} y_k\right)} \left(-y_j \sum_{k \neq j}^{K-1} y_k + y_j (1-y_j) \right) \right]$$

$$= -\left[-y_j \sum_{k=1}^{K-1} t_k + t_j - t_K \frac{1}{\left(1-\sum_{k=1}^{K-1} y_k\right)} \left(-y_j \sum_{k \neq j}^{K-1} y_k + y_j (1-y_j) \right) \right]$$

$$= -\left[-y_j \sum_{k=1}^{K-1} t_k + t_j - t_K y_j \right]$$

$$= -\left[-y_j \sum_{k=1}^{K} t_k + t_j \right]$$

$$= y_j - t_j .$$

$$\therefore \frac{\partial E}{\partial a_k} = y_k - t_k$$

3.

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

$$= \mathbb{E}_x \left[\sum_{m=1}^{M} \frac{1}{M} (y_m(x) - f(x))^2 \right] \quad -(i)$$

For a convex function,

$$f(\lambda a + (1-\lambda) b) \leq \lambda f(a) + (1-\lambda) f(b)$$

where $x = \lambda a + (1-x) b$.

So, $f(x)$ satisfies

$$\sum_{m=1}^{M} \lambda_m x_m \leq \sum_{m=1}^{M} \lambda_m f(x_m) \quad -(ii).$$

where $\lambda_m \geq 0$, $\Sigma_m \lambda_m = 1$ for all x_m in set $\{x_m\}$

Since $f(x) = x^2$ is convex, using (i) in (ii)

$$\left(\sum_{m=1}^{M} \frac{1}{M} (y_m(x) - f(x)) \right)^2 \leq \sum_{m=1}^{M} \frac{1}{M} (y_m(x) - f(x))^2$$

Since this holds for all expectation over x

$$\Rightarrow \mathbb{E}_x \left(\frac{1}{M} \sum_{m=1}^{M} (y_m(x) - f(x)) \right)^2 \leq \frac{1}{M} \sum_{m=1}^{M} \mathbb{E} (y_m(x) - f(x))^2$$

$$\Rightarrow \mathbb{E}_{ENS} \leq \mathbb{E}_{AV}$$

For $E(y)$,

$$E_{AV} = \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}_x (E(y(x)))$$

$$= \mathbb{E}_x \left[\sum_{m=1}^{M} \frac{1}{M} E(y(x)) \right]$$

from (ii)

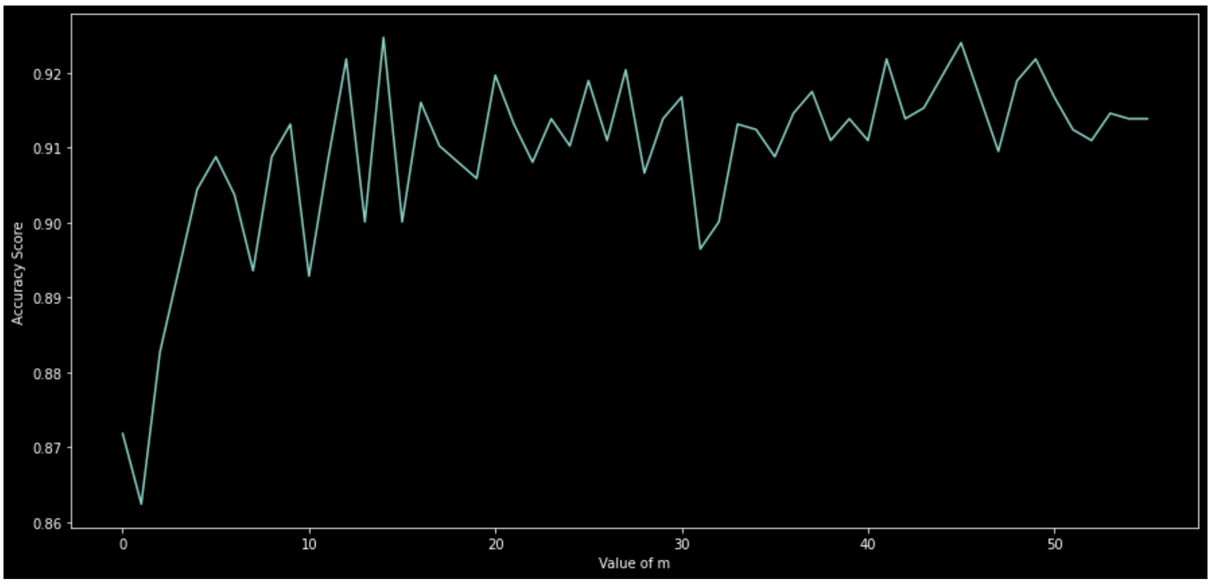
$$\mathbb{E}_x \left[\sum_{m=1}^{M} \frac{1}{M} E(y(x)) \right] \geq \mathbb{E}_x \left[E\left(\sum_{m=1}^{M} \frac{1}{M} y(x) \right) \right]$$

$\Rightarrow E_{AV} \geq E_{ENS}$.

4.

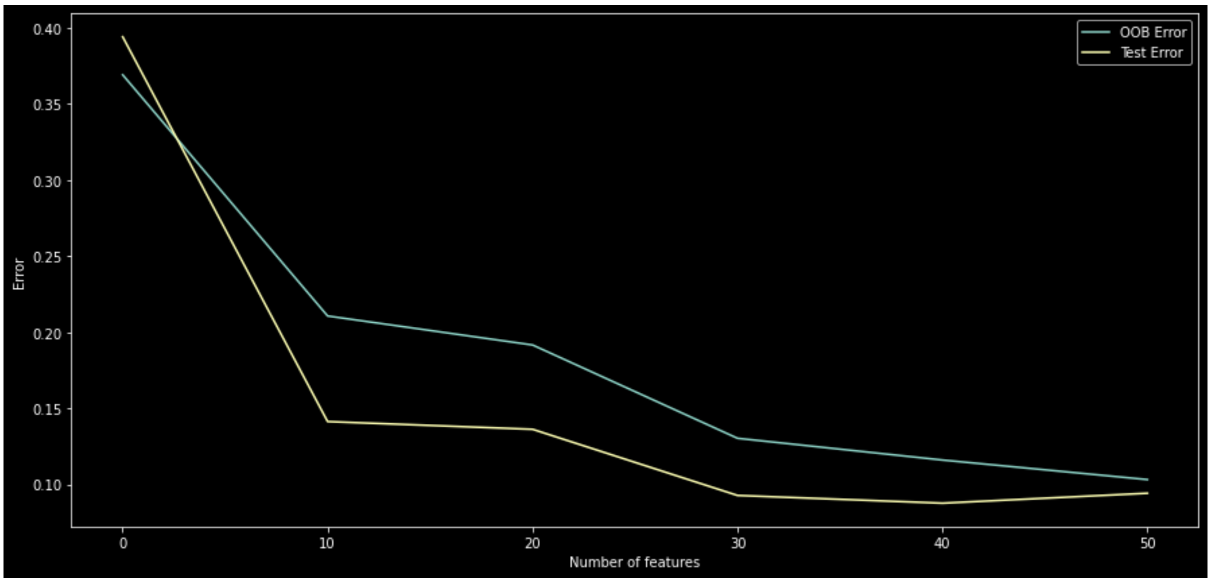
a. Comparing our random forest implementation with Sklearn’s method, we see that our model takes 679.194314 seconds to execute and gives an accuracy of 0.883418. In contrast, Sklearn’s implementation of the Random Forest model takes 0.019740 seconds and provides an accuracy of 0.910210. We see that the library implementation of SKlearn gives marginal higher accuracy than our implementation while also using only a fraction of the time to execute.

b. We can see the sensitivity of Random Forest wrt to the number of parameters in the below graph



We see the accuracy rise as the number of parameters increases and then oscillates with the number of features.

c. We plot the OOB error and test error on our implementation of Random Forest. The plot is as below:



We see that OOB error and test error follow a similar distribution, and for the most part, OOB error forms the upper bound for test error. Looking at this graph, we can conclude that our test error will never be higher than the OOB error.

5.

a. For pre-processing, we start by changing the *loan_status* values as described in the question. Then we analyse the dataset to find null values. We see that a lot of columns are empty and hence of no use to us. We remove all these empty columns. We also remove a few columns that are not entirely empty but have many values as NULL or NAN.

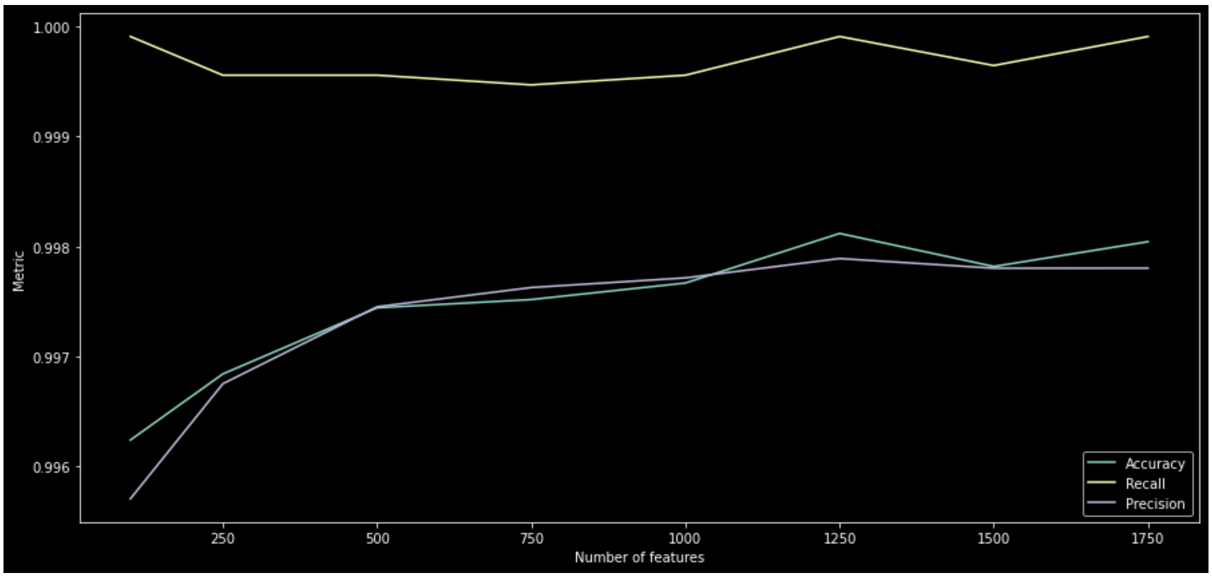
   We then check the remaining columns and find that many columns exist that may not contain valuable data such as *member_id, desc, grade, URL*, etc. We drop these columns as well. After removing all the useless columns, we convert all categorical columns. Some columns had int/float values, but some text was added along with it, such as *term* and *int_rate.* Since these columns are not technically categorical, we convert them to float. For other categorical values, we could either convert them to one-hot vectors or give each category a value. I decided to provide a value to each class as it should not affect the decision tree much and result in more accessible programming.

   Then I fill all nan values with median and remove all infinity values. Lastly, I scale all the values using MinMaxScaler to keep them in a limited range, though it would not have affected our model a lot.

b. We apply GradientBoostingClassifier to the processed data. Firstly, I applied GradientBoostingClassifier without any set parameters (i.e. default sklearn parameters). For that, I got test accuracy: 0.995483, precision: 0.994827, and recall: 0.999912. Then I tuned the learning rate and estimators for the model. We could have used GridSearch to find an even better model, but it took a long time, so I did it one by one. Finally, we select the learning rate as 0.15 and the number of estimators as 1250. With this model, I got test accuracy: 0.998118, precision: 0.997889, and recall: 0.999912.

   Since we were to select the model with the highest accuracy, we chose this model.

   The effect of the number of trees on the accuracy, precision, and recall is shown below:



We see that our accuracy and precision increase as we increase the number of trees.

c. For a final comparison, we use this model and compare it with DecisionTreeClassifier. For decision tree we get accuracy: 0.998043, precision: 0.997802, and recall: 0.999912. For our GraidentBoostingClassifier we get accuracy: 0.997892, precision: 0.997889, and recall: 0.999648. We observe that results from both models are similar.
This could be due to the small dataset. On a larger dataset, GradientBoostingClassifier