Q.1 (a) $D = \{(x_i, y_i)_{i=1}^n\}$ is a dataset for a 2-class classification problem, where $y_i \in \{-1, +1\}$. distribution over the dataset is given by w_t at iteration t, . Classifier G_t whose error rate

$$\epsilon_t = P_{x \sim w_t}[G_t(x) \neq y] \leq \frac{1}{2} - \gamma/2 \quad \text{for } \forall t.$$

$$0 < \gamma \leq 1$$

(a) $$g(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t G_t(x)\right).$$

For Adaboost, prove the Training error rate satisfies,

$$\frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(g(x_i) \neq y_i) \leq \exp\left(\frac{-\gamma^2 T}{2}\right)$$

As Given,

for adaboost error rate ϵ_t is given by, $= P_{x \sim w_t}[G_t(x) \neq y]$.

which is equal to,

$$\frac{\sum_{i=1}^{N} w_t(i)\, \mathbb{I}(y_i \neq G_t(x_i))}{\sum_{i=1}^{N} w_t(i)} \quad - (1)$$

Also, for updating the value of w at $(t+1)^{th}$ iteration, we have

$$w_{t+1}(i) = \frac{w_t(i) \cdot \exp(-\alpha_t y_i G_t(x_i))}{Z_t} \quad - (2)$$

where Z_t is a normalization factor.

Now, if we expand (2) w.r.t $w_t(i)$, we will get

$$w_{t+1}(i) = \left[\frac{w_{t-1}(i)\exp(-\alpha_t y_i G_{t-1}(x_i))}{Z_{t-1}}\right] \cdot \frac{\exp(-\alpha_t y_i G_t(x_i))}{Z_t}$$

Similarly, if we keep expanding $w_{t-2} \ldots w_{t-n}(i)$ we will keep adding the exponential terms and keep multiplying the Z_t terms.

Eventually,

$$w_{t+1}(i) = w_1(i) \cdot \frac{e^{-\sum_{t=1}^{T} \alpha_i y_i G_t(x_i)}}{\prod_{t=1}^{T} Z_t} \rightarrow (3)$$

Putting $\sum_{i=1}^{N}$ on both sides,

$$\sum_{i=1}^{N} w_{t+1}(i) = \sum_{i=1}^{N} w_1(i) \cdot \frac{e^{-\sum_{t=1}^{T} \alpha_i y_i G_t(x_i)}}{\prod_{t=1}^{T} Z_t} \rightarrow (4)$$

(set at starting) $\rightarrow \frac{1}{N}$

and $\sum_{i=1}^{N} w_{t+1}(i) = 1 \rightarrow (5)$

$$1 = \frac{1}{N} \sum_{i=1}^{N} \frac{e^{-\sum_{t=1}^{T} \alpha_i y_i G_t(x_i)}}{\prod_{t=1}^{T} Z_t} \rightarrow (6)$$

From (6), $\frac{1}{N}\sum_{i=1}^{n} e^{-\sum_{t=1}^{T}\alpha_t y_i G_t(x_i)} = \prod_{t=1}^{T} Z_t$ → (7)

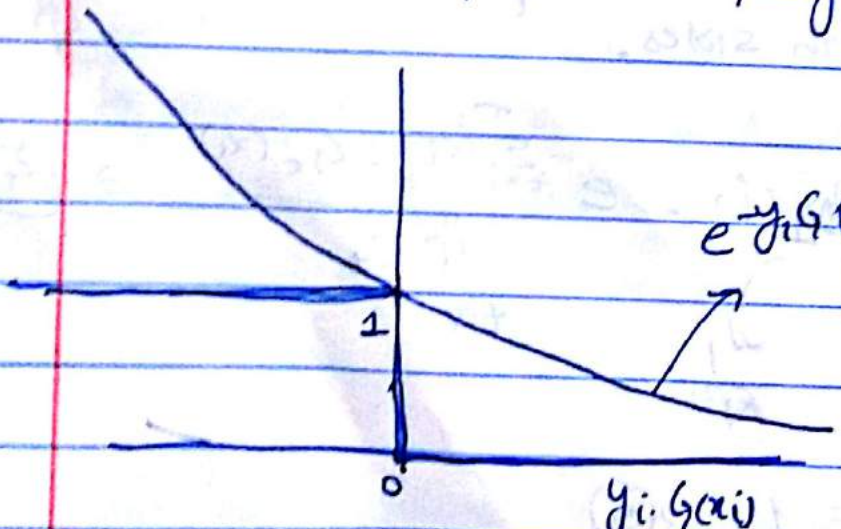
and $e^{-\sum_{t=1}^{T}\alpha_t y_i G_t(x_i)} = e^{-y_i G(x_i)}$ → (8)

From (8) & (9), we have

$$\frac{1}{N}\sum_{i=1}^{n} e^{-y_i G(x_i)} = \prod_{t=1}^{T} Z_t \rightarrow (9)$$

Now, to prove $\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(g(x_i) \neq y_i) \leq \exp\left(\frac{-r^2 T}{2}\right)$

the 0-1 loss function is given by.



With this, we can say that, 0-1 indicator function $\sum_{i=1}^{N} \mathbb{1}(g(x_i) \neq y_i)$ will always be upper bounded by $\sum_{i=1}^{N} \exp(-y_i G(x_i))$

$$\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(g(x_i) \neq y_i) \leq \frac{1}{N}\sum_{i=1}^{N} \exp(-y_i G(x_i))$$

↓ From (9)

$$\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(g(x_i) \neq y_i) \leq \prod_{t=1}^{T} Z_t \rightarrow (9.1)$$

Now, note that, $Z_t = \sum_{i=1}^{N} w_t(i) \cdot \exp(-\alpha_t y_i G_t(x_i))$

can also be written as,

$$= e^{-\alpha_t} \sum_{y_i = G_t(x_i)} w_t(i) + e^{\alpha_t} \sum_{y_i \neq G_t(x_i)} w_t(i) \quad — (10)$$

$$\Rightarrow (e^{\alpha_t} - e^{-\alpha_t}) \sum w_t(i) \mathbb{1}(y_i \neq G_t(x_i)) + e^{-\alpha_t} \sum_{i=1}^{n} w_t(i) \quad \hookrightarrow (11)$$

from (1) $\epsilon_t = \sum_{i=1}^{N} \frac{w_t(i) \mathbb{1}(y_i \neq G_t(x_i))}{\sum_{i=1}^{N} w_t(i)}$

we get $Z_t = (1-\epsilon_t) e^{-\alpha_t} + \epsilon_t e^{\alpha_t} \rightarrow (12)$.

minimizing this over α_t, we get

$$\frac{d(Z_t)}{\alpha_t} = -(1-\epsilon_t) e^{-\alpha_t} + \epsilon_t e^{\alpha_t}.$$

setting the gradient to zero, we get

$$e^{\alpha_t} \cdot \epsilon_t = (1-\epsilon_t) e^{-\alpha_t}$$

$$e^{2\alpha_t} = \frac{1-\epsilon_t}{\epsilon_t}$$

$$\ln(e^{2\alpha_t}) = \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

$$2\alpha_t = \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) \rightarrow (13)$$

plugging back α_t in (12).

$$Z_t = (1-\epsilon_t) e^{-\frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)} + \epsilon_t e^{\frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)}$$

$$= (1-\epsilon_t)\left(\frac{1-\epsilon_t}{\epsilon_t}\right)^{-1/2} + \epsilon_t\left(\frac{1-\epsilon_t}{\epsilon_t}\right)^{1/2}$$

$$\Rightarrow \sqrt{\epsilon_t(1-\epsilon_t)} + \sqrt{\epsilon_t(1-\epsilon_t)}$$

$$Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)} \rightarrow (14)$$

Given that $\epsilon_t \leq \frac{1}{2} - \frac{\gamma_t}{2}$ at each t^{th} iteration

$$Z_t = 2\sqrt{\left(\frac{1}{2}-\frac{r_t}{2}\right)\left(1-\frac{1}{2}+\frac{r_t}{2}\right)} = 2\sqrt{\left(\frac{1}{2}-\frac{r_t}{2}\right)\left(\frac{1}{2}+\frac{r_t}{2}\right)}$$

$$Z_t \Rightarrow \sqrt{(1-r_t)(1+r_t)} = \sqrt{1-r_t^2} \rightarrow (15)$$

which means each Z_t will always be less than 1.

From (9.1) So, we proved that $\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(g(x_i) \neq y_i) \leq \prod_{t=1}^{T} Z_t$

this can be written as,

$$\frac{1}{N}\sum_{i=1}^{N} (g(x_i) \neq y_i) \leq \prod_{t=1}^{T} \sqrt{1-r_t^2} \rightarrow (16)$$

↓

This can be bounded by

$$\exp\left(-\frac{1}{2}\sum_{t=1}^{T} r_t^2\right) \rightarrow (17)$$

And we assume that r_t is a constant for all t. (as given)

we can write (17) as. $\exp\left(-\frac{1}{2} T.r^2\right) \rightarrow (18)$

So, $\frac{1}{N}\sum_{i=1}^{N} (g(x_i) \neq y_i) \leq \exp\left(-\frac{1}{2} r^2 T\right)$.

Hence proved

(b) We assume that we can always get a weak classifier G_t whose error rate satisfies $\epsilon_t \leq \frac{1}{2} - \gamma_2$ for $0 < \gamma \leq \frac{1}{2}$,

the training error rate

$\frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(g(x_i) \neq y_i)$ will eventually become zero, as we keep on increasing t. (or when we keep adding more weak classifiers).

The reason is, the error rate is bounded by,

$\frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(g(x_i) \neq y_i) \leq \prod_{t=1}^{N} Z_t$ as shown in previous part.

We know that $Z_t = \sqrt{1 - \gamma_t^2}$ which will always be less than 1.,

So for each t, will keep on multiplying the Z_t's (which are less than 1), and will be getting our upper bound very close to zero at Tth iteration which is very large, (say 10000), our upper bound of the training error rate will be very close or almost zero as T is increased.

Q.3

(a) We have a two class classification problem with data $\{(x_i, y_i)\}_{i=1}^{n}$ where $y_i \in \{0,1\}$ & $x_i \in R^d$. We have a activation model $a_i = a(x_i, w) = w^T x_i$. and activation function of the form

$$F_{sigmoid}(a_i) = \frac{1}{1+\exp(-a_i)}$$

$$F_{relu}(a_i) = \max(0, a_i)$$

(a) We have square loss on the entire dataset in terms of activation function $F_{sigmoid}$ applied to all the activations $a = [a_1\ a_2 \ldots a_n]$.

$$Loss^{(sigmoid)}(a) = \sum_{i=1}^{n} (y_i - F_{sigmoid}(a_i))^2.$$

$$Loss^{sigmoid}(a) = \sum_{i=1}^{n} \left(y_i - \frac{1}{1+\exp(-a_i)}\right)^2$$

For the $L_{sigmoid}^{sq}$ (a), we will prove that it is not convex.

The definition of convexity used here is, we find the Hessian of the 'activation' vector (L_{sq}(sigmoid)).

The Hessian is to proved posi-semi definite. If it is not, we say that the function $L_{sq}^{sigmoid}$ is not convex.

Given, $L_{sq} = \sum_{i=1}^{n} (y_i - F(a_i))^2$ → (1).

derivative of this function is,

$$\nabla L_{sq} = [-2(y_1 - F(a_1))F'(a_1), -2(y_2 - F(a_2))F'(a_2), \ldots, -2(y_n - F(a_n))F'(a_n)]$$ → (2)

This is a vector, and we will calculate the double derivative of this.

Let's calculate the double derivative of ∇L_{sq} w.r.t a_i,

$$\nabla\nabla L_{sq}{}_{a_2} \Rightarrow -2(y_1 - F(a_1))F''(a_1) + 2F'(a_1)\cdot F'(a_1) \quad \hookrightarrow (3)$$

Computing the Hessian matrix, we get a matrix with diagonal elements only, as the other derivatives will be zero (derivative of a_{n-1} wrt. a_1 will zero.). Basically, all other elements will be zero except a_1, likewise for a_n.

Therefore we get,

$$\nabla\nabla L_{sq} = \begin{bmatrix} -2(y_1 - F(a_1))F''(a_1) + 2(F'(a_1))^2 & 0 & 0 & 0 & 0 \\ 0 & \ddots & & & 0 \\ 0 & & \ddots & & 0 \\ 0 & & & \ddots & 0 \\ 0 & & & & 0 \\ 0 & & & & -2(y_n - F(a_n))F''(a_n) + 2(F'(a_n))^2 \end{bmatrix} \quad \hookrightarrow (4)$$

The above matrix is a diagonal matrix. & now, to prove this as positive semi-definite, we will say that all the diagonal elements should be greater than zero. If it doesn't happen, we say that it is not positive-semi definite.

or, in simple terms. $v^T(\nabla \nabla L_{sq}) v > 0$, then it is positive semi definite.

For (a) we have $f_{sigmoid}(a_i) = \frac{1}{1 + exp(-a_i)} = a \rightarrow (5)$

by differentiating this, we get

$$f'_{sigmoid}(a_i) = \frac{exp(-a_i)}{(1 + exp(-a_i))^2} \rightarrow (6)$$

or $a(1-a) = f_{sigmoid}(a_i) * (1 - f_{sigmoid}(a_i))$

$\rightarrow (7)$

Taking double derivative of (7). we get.

$$F''_{sigmoid}(a_i) = \left(1 - F_{sigmoid}(a_i)\right)^2 F_{sigmoid}(a_i) - F^2_{sigmoid}(a_i)\left(1 - F_{sigmoid}(a_i)\right)$$

taking $f(a)$ & $(1 - F(a))$ common, we get.

$$F''_{sigmoid}(a_i) = F_{sigmoid}(a) \cdot \left(1 - F_{sigmoid}(a)\right)\left(1 - 2F_{sigmoid}(a)\right) \rightarrow (8)$$

Substituting (7) & (8) in (4), we have.

(ith element)

$$\Rightarrow -2\left(y_i - F(a_i)\right) F''(a_i) + 2\left(F'(a_i)\right)^2$$

$$\Rightarrow -2\left(y_i - F_s(a_i)\right) \cdot F_s(a_i) \cdot \left(1 - F_s(a_i)\right)\left(1 - 2F_s(a_i)\right) + 2\left[F_s(a_i) \cdot \left(1 - F_s(a_i)\right)\right]^2$$

$$\Rightarrow 2F_s(a_i)\left(1 - F_s(a_i)\right)\left[F_s(a_i)\left(1 - F_s(a_i)\right) - \left(y_i - F_s(a_i)\right) \cdot \left(1 - 2F_s(a_i)\right)\right]$$

$$\Rightarrow 2F_s(a_i)\left(1 - F_s(a_i)\right)\left[F_s(a_i) - F_s^2(a_i) - y_i + 2y_i F_s(a_i) + F_s(a_i) - 2F_s^2(a_i)\right]$$

$$\Rightarrow 2 f_s(a_i)(1-f_s(a_i))\left[2y_i f_s(a_i) - y_i - 3 f_s^2(a_i) + 2 f_s(a_i)\right]$$

$\rightarrow$ (9)

Case I

$y=1$ $\rightarrow$ substituting y in (9).

(9) $\rightarrow$ quadratic.

$$\Rightarrow 2 f_s(a_i)(1-f_s(a_i))\left[4 f_s(a_i) - 1 - 3 f_s^2(a_i)\right]$$

$$= 2 f_s(a_i)(1-f_s(a_i))\left[(f_s a_i - 1) + 3 f_s(a_i) - 3 f_s^2(a_i)\right]$$

$$= 2 f_s(a_i)(1-f_s(a_i))\left[(1-3 f_s(a_i))(f_s(a_i) - 1)\right]$$

switching the signs.

$$\Rightarrow 2 f_s(a_i)(1-f_s(a_i))^2 (3 f_s(a_i) - 1)$$

Since $0 < f_s(a_i) < 1$,

$\Rightarrow$ $2 f_s(a_i)(1-f_s(a_i))^2 \cdot (3 f_s(a_i) - 1)$

always positive. $\downarrow$ +ve or -ve

Case II $y = 0$. Substituting y in (9).

$$\Rightarrow 2 f_s(a_i)(1 - f_s(a_i))\left(2 y_i f_s(a_i) - y_i - 3 f_s^2(a_i) + 2 f_s(a_i)\right)$$

$$\Rightarrow 2 f_s(a_i)(1 - f_s(a_i))\left(2 f_s(a_i) - 3 f_s^2(a_i)\right)$$

$$\Rightarrow 2 f_s^2(a_i)(1 - f_s(a_i))\,(2 - 3 f_s(a_i)) \quad \because 0 < f_s(a_i) < 1$$

always positive — this can be positive or negative.

Therefore, the diagonal elements of the Hessian matrix can be greater or less than zero for some values of a_i.

With that we can say that there will be a vector v for which $v^T (\nabla \nabla L_{sq}^{sigmoid}) v < 0$;

Hessian

Hence the $L_{sq}^{sigmoid}$ is not convex.

Proved

(b) Given as

$$F_{relu}(a_i) = \max(0, a_i)$$

From eq (4), the diagonal matrix of the Hessian is.

$$\nabla\nabla L_{sq}^{relu} \text{ or } H(F_{relu}) = \begin{bmatrix} 2(y_1 - F(a_1))F''(a_1) + 2(F'(a_1))^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \ddots & & & & \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & \\ 0 & 0 & 0 & 0 & 0 & 2(y_n - F(a_n))F''(a_n) + 2(F'(a_n))^2 \end{bmatrix}$$

$$F_{relu}(a_i) = \max(0, a_i)$$

As F_{relu} is non-smooth function we calculate the derivatives piece-wise.

$$F'_{relu}(a_i) \Rightarrow \begin{cases} 1 & a_i > 0 \\ 0 & a_i \leq 0 \end{cases}$$

$$F''_{relu}(a_i) = \begin{cases} 0 & a_i > 0 \\ 0 & a_i \leq 0 \end{cases}$$

At 0, the F_{relu} is non-differentiable, so we calculate

the derivatives at values very close to zero, which are as follows,

↳ for $a_i > 0$, we get something like,

$$2(y_i - a_i) \cdot 0 + 2(1)^2 = 2$$

For $a_i < 0$, we get something like,

$$= 2(y_i - (a_i)) \cdot 0 + 2 \cdot (0)^2 = 0$$

In both cases, we are getting the values of Hessian elements matrix as greater than 0.

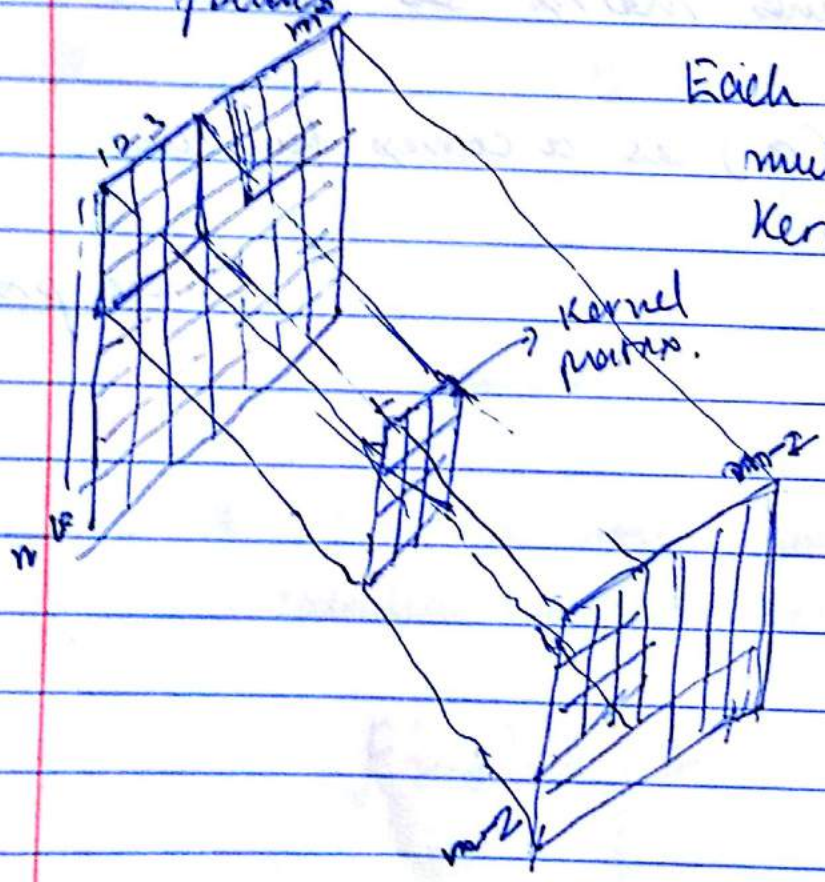
Hence, $L_{sq}^{selu}(a_i)$ is a convex function.

Hence proved

Q.4(a) The Given problem is about convolutionizing Kernels. The input given is $X \in R^{n \times m}$. and a Kernel matrix. (3×3), applied to input., without any zero padding or any other extensions.

We get the output matrix Z as $R^{(n-2)*(m-2)}$

The convolution operation which takes place is as follows



Each pixel is sequentially multiplied by values of Kernel matrix.. - For; example if we have a section of input matrix as 3×3 and is multiplied with Kernel matrix, we get the value as;

↓

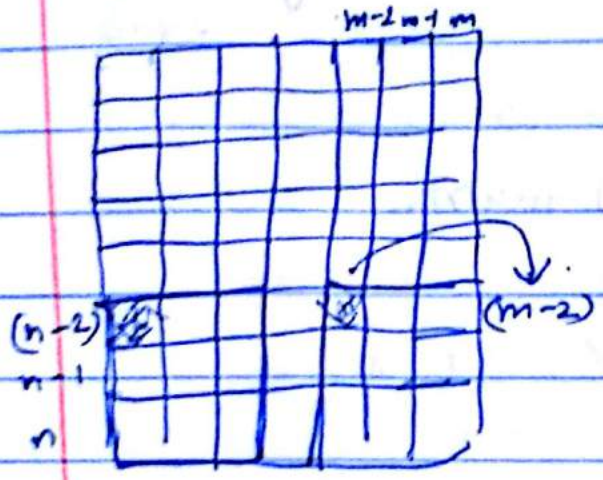
a	b	c
d	e	f
g	h	i

$*$

1	2	3
4	5	6
7	8	9

$= (1*a) + (2*b) + (3*c) + (4*d) + (5*e) + (6*f) + (7*g) + (8*h) + (9*i).$

The Z matrix is therefore computed using the previous method.



In the given problem, the Kernel matrix is multiplied to the input matrix till (n-2) rows and (m-2) columns, as there is no padding.

The value of convolution is not defined if we exceed the boundaries, Hence, we limit the operation to $\underbrace{n-2}_{\text{rows}}$ & $\underbrace{m-2}_{\text{columns}}$.

Pseudo code

Assumming we have index starting with 1.

Begin

Set Z as (n-2) and (m-2) size matrix.

[n, m] as the size of X.

For i=1 to n-2

For j=1 to m-2

set sum = 0

For K=1 to 3

For l=1 to 3

sum = sum + Kernel(K, l) * X(i+K-1, j+l-1)

end for

End for

Z(i, j) = sum;

end for

end for.

Output → $Z \in R^{(n-2)*(m-2)}$

(b) Let $vec(X)$ be R^{nm} → vectorized form of X.
constructed column wise.

$Vec(X)[1:n] = x[1:n, 1]$

$[n+1, 2n] = x[1:n, 2]$

⋮

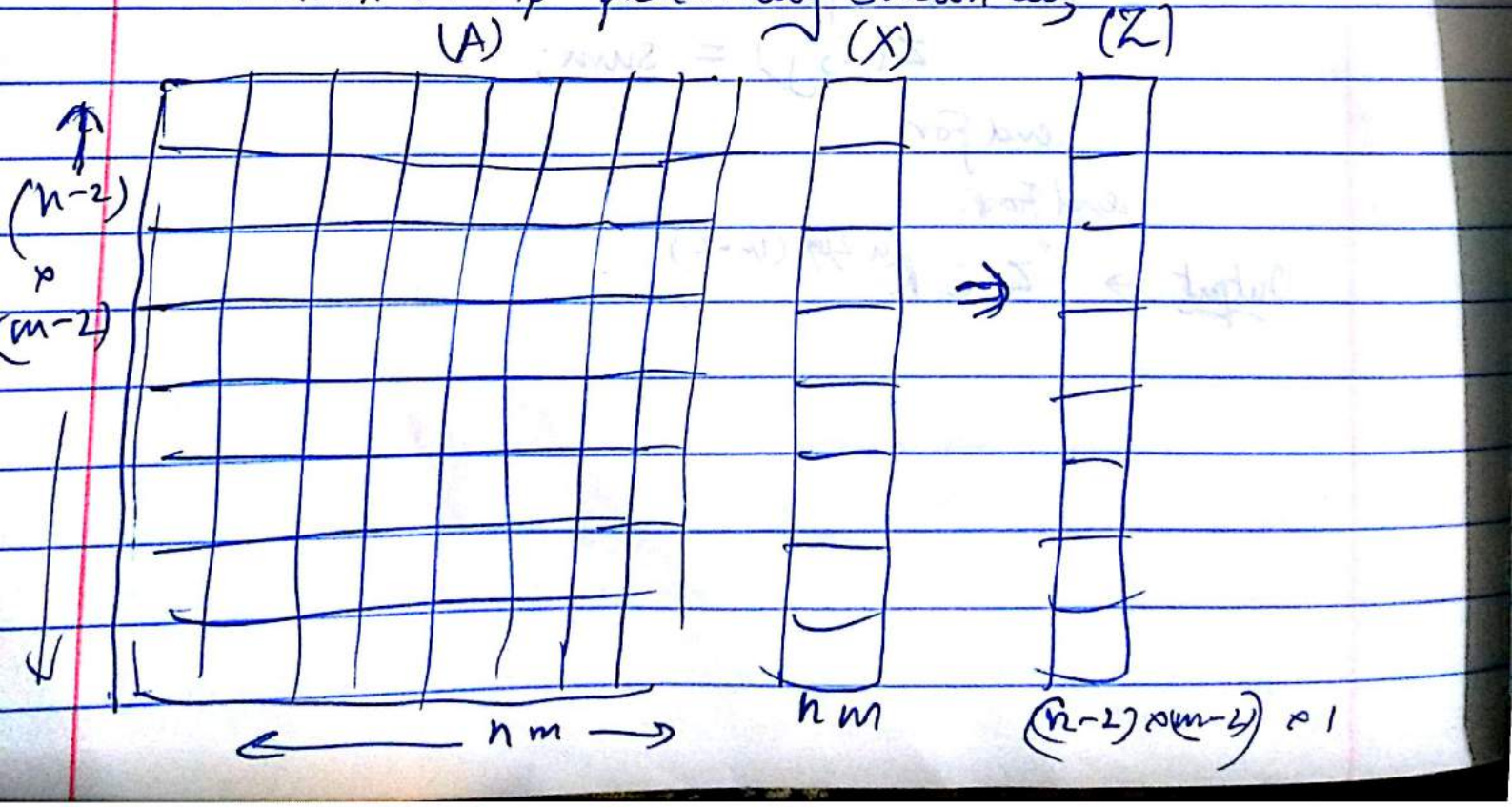
$Vec(X)[(m-1)n+1, mn] = x[1:n, m]$

$A \in R^{((n-2)\times(m-2))\times nm}$ is a matrix such that

$Vec(Z) = A.vec(X)$

This A is the Kernel matrix for this vectorized form.

Matrix A is pictorially shown as,



We need to map the elements of A with Kernel matrix. K.

K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}
K_{31}	K_{32}	K_{33}

Mathematically, the First row contains the Kernel elements as follows,

First, second, third element is set as K_{11} K_{21} K_{31} remaining $(n-3)$ are set to zero. Next $(n+1)$ $(n+2)$ $(n+3)$ elements are set as K_{12} K_{22} K_{32} and remaining $(n-3)$ are set to zero. $(2n+1)$, $(2n+2)$, $(2n+3)$ are set as K_{13}, K_{23}, K_{33} and remaining all the elements are set to zero.

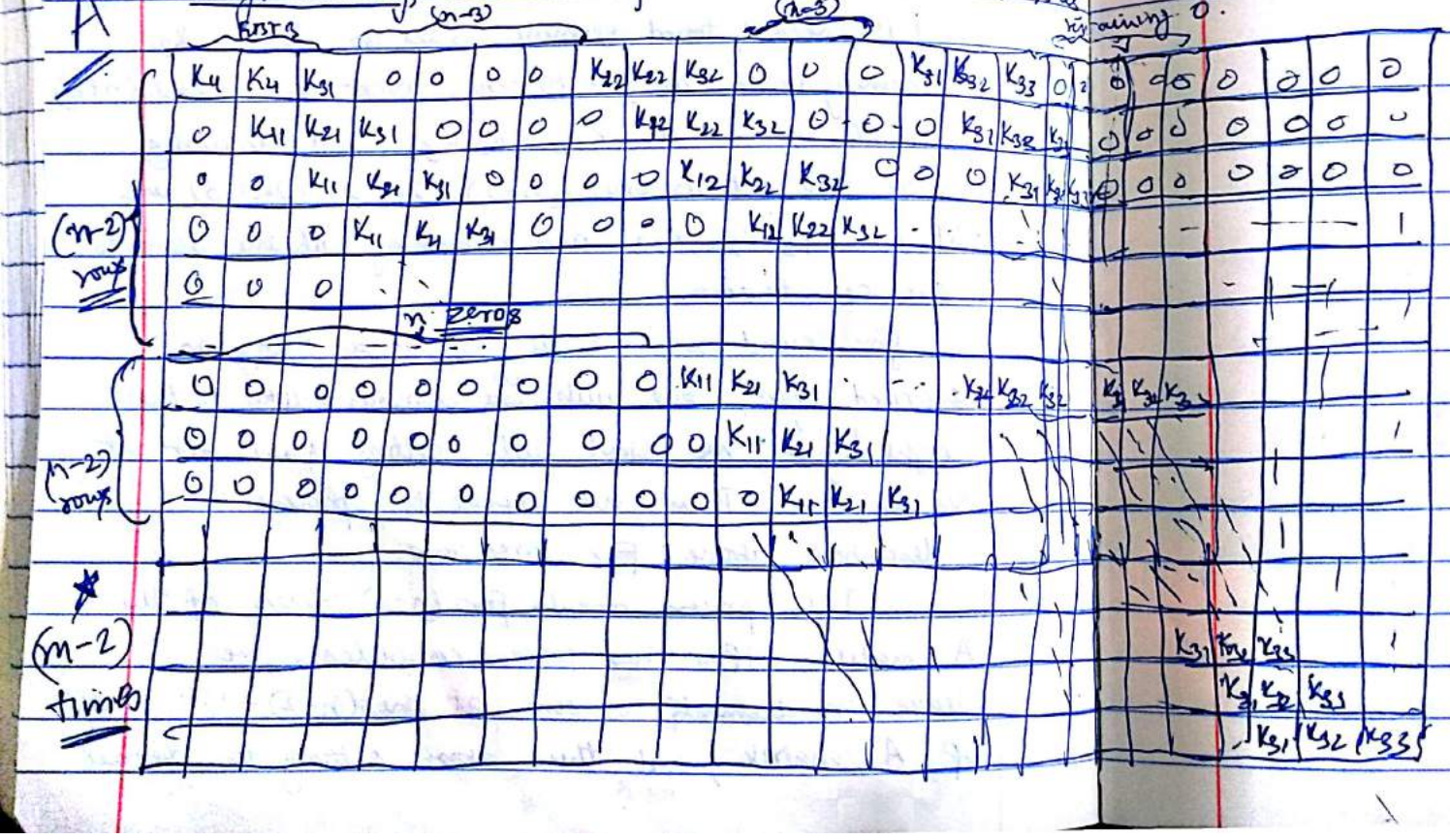
For second row, we do the similar thing as described above, but with all elements shifted to the right by 1; we start with setting first element set as zero. Then, we repeat the process as described above for first row.

This process repeats for $(n-2)$ rows of the A matrix. After this process completes, we have n elements as zero at the $((n-2)+1)$th row of A matrix, and then start setting the Kernel

matrix values K_{11}, K_{21}, K_{31} for next three elements. Then again leave $(n-3)$ elements as zero. & so on. and so forth... [as told for First row]

This procedure is repeated $(n-2)$ times. till we reach the $((n-1)n+1)$th row of the vector X.

Diagramatically we can represent the same matrix A as



Q.2(a) Summary of methods & Results

Bagging :→ Bagging or Boot-strap aggregating is one of the ensemble algorithms which is usually applied to decision trees.

In Bagging, we are given a training data set D of size n, where bagging generates m new training sets D_i, each of size n' by sampling from D uniformly with replacement. By sampling with replacement, some observations may be repeated in each D_i.

In this problem, we are keeping n' = n. A bootstrap sample may contain new unique training samples as well as few duplicates. We have the bag size of [5, 10, 15, 20, 25, 30, 35, 40, 45, 50] in the given problem which is the number of decision trees or the classifiers we will use to test the data.

In this problem, the decision tree is restricted to a 2-layer decision tree [mentioned] We create the decision trees using the best split value whenever Information Gain

is highest among the set of samples across the features. This way splitting is done. [More explained later] - The second level, we stop the decision tree forming by voting for the majority of class labels in that respective split.

The classifiers or the decision trees are used to test the data, and each sample is fitted with each classifier [depending upon the bag size], and the predicted label is classified based on the majority voting for that respective bag.

Algorithm:- Bagging:-

1) Consider training set D of size 'n' samples.
2) Create Di training set by drawing 'n' samples with replacement. In expectation, Di will leave few samples.
3) Create K bootstrap samples, $D^1, D^2, \ldots D^K$. Here K is the B array given in the problem.
4) Train a distinct classifier over each Di.
5) Test and classify a sample using majority voting.

Splitting Criteria:-

We split the decision tree based on the following set of statements.

(i) We iterate through the samples for a feature and select the unique values from the sample for that respective feature. We select two unique values at a time, and select the value as their mean.

(ii) We find the maximum Information Gain for the samples for each feature. Iteratively, we find this Information Gain for each feature too. Which ever the maximum, we assign the split value [value at which the Information Gain was highest for that ~~mean~~ mean of two sample feature values with respect to the feature]. This value is selected to divide the data into two parts.

(iii) The 'greater than' value goes on the left side of the tree and the 'less than' the split value goes on right side.

(iv) This procedure is repeated to find the split value and the feature from level 2 nodes too.

If we obtain pure class of labels at any split we assign the class labels to the leaves.

"Assumption → I have assumed that we should have labels on both side of the root node, otherwise the tree would grow in one direction only. This is taken care by the information gain criteria, the maximum Information gain splits the decision tree into two halves of population."

At any sub-level (level 1) node, if we have do not have pure class of labels, we stop forming further tree [as it is a 2 layer binary tree], and assign the labels based on the majority of the labels for that splitted data.

Another point to be taken care is, we do not Split on the same feature on the children node as we did on root node. This is decided as we would get pure pure class in second level everytime which is redundant.

Information Gain Calculation :-

Information Gain is calculated using the following formulas :→

Entropy $H(X) = \sum_{i=1}^{n} - p(x_i) \log_2 p(x_i)$.

Condition Entropy $H[X/Y] = -\sum_{j=1}^{m} p(y_j) H(X/y_j)$

Information Gain : $IG[X/Y] = H(X) - H[X/Y]$

What this means is shown below;
Suppose we have a data with 20 samples. where the split is 17 | 3. i.e 17 goes on leftside [satisfying the inequality], 3 goes on right side. The total labels are distributed as 14 positive 6 negative.

The Entropy $H(X)$ would be $= -\frac{6}{20} \log_2 \frac{6}{20} - \frac{14}{20} \log_2 \frac{14}{20}$

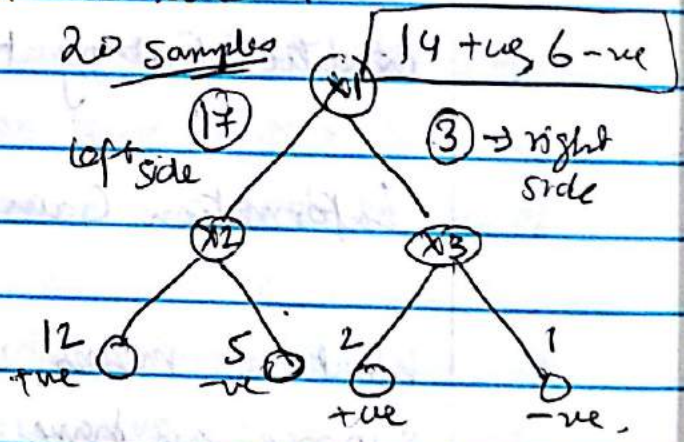
The conditional entropy would be :-
Out of 17, 12 positive labels were there, and 5 negative, Similarly, out of 3, 2 were positive and 1 was negative.

The conditional Entropy would be,

$$H[X/Y] \Rightarrow \frac{17}{20}\left[-\frac{12}{17}\log_2\frac{12}{17} - \frac{5}{17}\log_2\frac{5}{17}\right] + \frac{3}{20}\left[-\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3}\right]$$

Information Gain = $H[X] - H[X/Y]$.

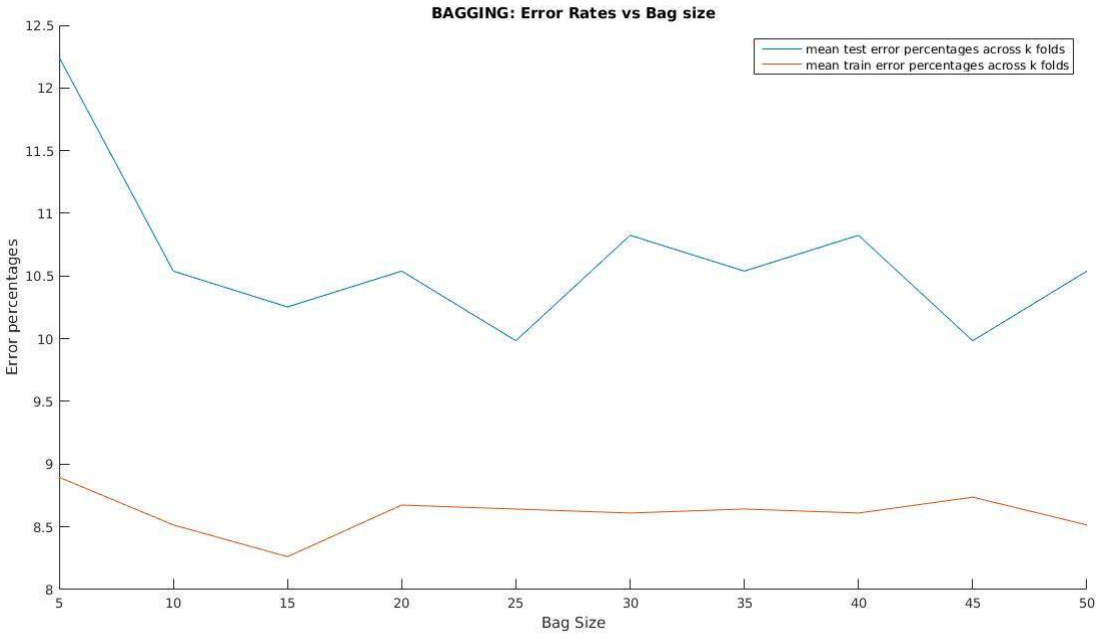
The tree for this would look like.



Results:-

Attached is the figure and the test & training error table, with mean and standard deviation of the errors for each Bag size.

Inference is:- As we increase the Bag size, the standard deviation should decrease, and the errors should also decrease, but because our bag size are small → 5 to 50, it is somewhat fluctuating with some parts decreasing overall. The test error rates are always less than the training error rates too.


BAGGING: Error Rates vs Bag size
mean test error percentages across k folds
mean train error percentages across k folds
Error percentages
Bag Size
12.5
12
11.5
11
10.5
10
9.5
9
8.5
8
5
10
15
20
25
30
35
40
45
50

BAGGING

TrainTest/BagSize	5	10	15	20	25	30	35	40	45	50
TrainFold1	0.085443038	0.0949367089	0.098101266	0.0886075949	0.088607595	0.091772152	0.088607595	0.088607595	0.098101266	0.088607595
TestFold1	0.0285714286	0.0285714286	0.028571429	0.0285714286	0.028571429	0.028571429	0.028571429	0.028571429	0.028571429	0.028571429
TrainFold2	0.1012658228	0.082278481	0.091772152	0.0917721519	0.091772152	0.091772152	0.094936709	0.091772152	0.094936709	0.091772152
TestFold2	0.0857142857	0.0857142857	0.085714286	0.0857142857	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286
TrainFold3	0.0949367089	0.0949367089	0.098101266	0.0981012658	0.094936709	0.098101266	0.098101266	0.094936709	0.098101266	0.098101266
TestFold3	0.0285714286	0.0285714286	0.057142857	0.0285714286	0.028571429	0.057142857	0.057142857	0.057142857	0.057142857	0.057142857
TrainFold4	0.0949367089	0.085443038	0.085443038	0.0949367089	0.088607595	0.088607595	0.088607595	0.088607595	0.091772152	0.091772152
TestFold4	0.2285714286	0.0857142857	0.085714286	0.0857142857	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286
TrainFold5	0.0696202532	0.0664556962	0.066455696	0.0696202532	0.069620253	0.066455696	0.066455696	0.066455696	0.066455696	0.066455696
TestFold5	0.2857142857	0.2857142857	0.285714286	0.2857142857	0.285714286	0.285714286	0.285714286	0.285714286	0.285714286	0.285714286
TrainFold6	0.0791139241	0.082278481	0.069620253	0.0759493671	0.085443038	0.069620253	0.079113924	0.082278481	0.085443038	0.069620253
TestFold6	0.1428571429	0.1428571429	0.142857143	0.1428571429	0.142857143	0.142857143	0.142857143	0.142857143	0.142857143	0.142857143
TrainFold7	0.0759493671	0.0886075949	0.056962025	0.0949367089	0.088607595	0.094936709	0.088607595	0.091772152	0.088607595	0.091772152
TestFold7	0.0857142857	0.0571428571	0.028571429	0.0571428571	0.057142857	0.057142857	0.057142857	0.057142857	0.057142857	0.057142857
TrainFold8	0.085443038	0.082278481	0.088607595	0.085443038	0.085443038	0.085443038	0.085443038	0.085443038	0.085443038	0.085443038
TestFold8	0.1142857143	0.1428571429	0.114285714	0.1428571429	0.142857143	0.142857143	0.114285714	0.142857143	0.114285714	0.114285714
TrainFold9	0.1111111111	0.0920634921	0.088888889	0.0857142857	0.085714286	0.088888889	0.092063492	0.088888889	0.079365079	0.088888889
TestFold9	0.1388888889	0.1111111111	0.111111111	0.1111111111	0.055555556	0.111111111	0.111111111	0.111111111	0.055555556	0.111111111
TrainFold10	0.0917721519	0.082278481	0.082278481	0.082278481	0.085443038	0.085443038	0.082278481	0.082278481	0.085443038	0.079113924
TestFold10	0.0857142857	0.0857142857	0.085714286	0.0857142857	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286
Mean(TrainTest)	0.1057097649	0.0946678198	0.091990133	0.0954529085	0.092535653	0.096558421	0.095296689	0.096558421	0.093006307	0.094667321
Std(TrainTest)	0.0566288246	0.0504042195	0.050228592	0.050301869	0.0507626	0.04900312	0.047957227	0.04881787	0.048556908	0.048192609

(b) Random Forests

Random Forests is another ensemble methods of machine learning, which builds a Forests of decision trees. Similar to bagging, in Random Forest also work on the bootstrap sampling technique for picking up training data. where the dataset D_i of size n_i is selected, and a new training set D_i is generated with size n' [$n' = n$ in this problem] by sampling from D uniformly with replacement, [some observations may be repeated for each D_i]. Apart from this Random ness, Random forest also has another Random ness component, which is based on randomly selected features [$m < d$]. In this problem, we are given a matrix of [1 .... 34] which is the feature size each time used to create decision trees uniquely.

Again, for this problem also, we restrict ourselves to a 2-layer simple binary decision tree, and we create the decision tree using the best split value

by calculating Information gain across the selected samples & randomly drawn features. [Explained Later again]. At the second level, post splitting, we stop and classify by choosing the majority of class labels accordingly for that respective split.

Similar to bagging, once the tree or the classifiers are formed. We test the data [test, train data] on each classifier [100 in this problem]. and classify the label by majority voting.

Algorithm: → Random Forest

(1) Consider training set D with size of 'n' samples
(2) Create a D_i training set by drawing 'n' samples with replacement. In Expectation, D_i will leave out few samples.
(3) Create K bootstrap samples $D^1, D^2 \ldots D^K$. Here K = 100
(4) Randomly select $m \leq d$ Features while creating the decision tree. [Here m is a matrix from 1:34].
(5) Train a distinct classifier for each D_i for a given m.
(6) Determine the best split using the selected features only.
(7) Test & classify the samples using majority vote.

Splitting criteria :-

We split the decision trees using the following set of statements:

(i) We iterate through the samples for the randomly selected feature, out of 34 features. For. eg → if M=3, we select 3 random features for the data samples and iterate sequentially over each sample and over each feature [which is randomly selected] to ~~gain~~ get the split value wherever the Information Gain is highest. ~~The~~ Similar, to Bagging, the split value is selected by identifying unique values in the samples for the selected feature [Iteratively].

(ii) This procedure is repeated even for child nodes, and if we get a split value which divides the data into two population, we stop further forming of the tree & do majority voting for the leaves.

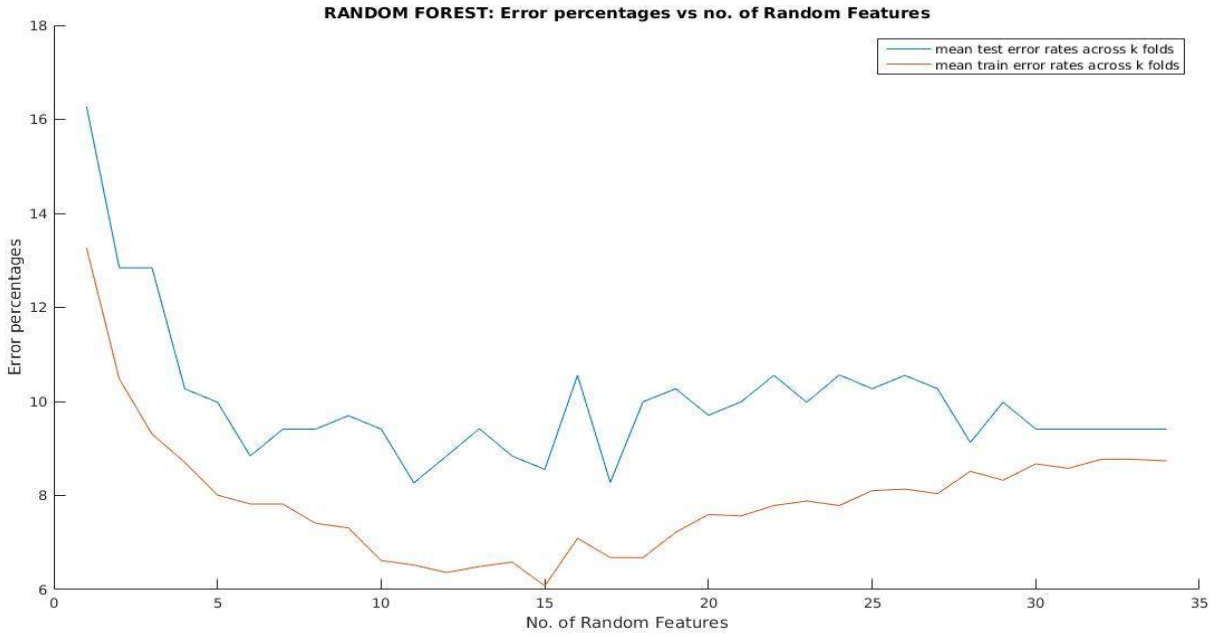
Assumption : Similar to Bagging, we do not split the root node, if we have pure class labels, otherwise the tree would grow in one direction.

Same as Bagging, if a randomly selected feature is selected on root node, we skip the feature to be split on the children nodes. i.e. While iterating through the features to calculate max. information gain, we skip the root node selected feature.

Random Forest as bigger Bag size = 100 in this problem, and we are expected to get lower error rates than Bagging. also due to random selection of features, we avoid over fitting in the Random forest algorithm.

Results

Attached is the figure and test error rates & training error rates for 10 folds. With increasing number of features, as we go from 1 to 34, we see the error rates go down, and from the figure we can see the error rates started stabilizing at around 20 features and above. We see some fluctuations in between like 'feature = 15', there is sudden drop in error rate [training] which can be due to the data or features that were selected. The total error rate percentages are between 6 to 16%. , with test error rates always higher than training error rates.


RANDOM FOREST: Error percentages vs no. of Random Features
mean test error rates across k folds
mean train error rates across k folds
Error percentages
No. of Random Features
18
16
14
12
10
8
6
0
5
10
15
20
25
30
35

RANDOM FOREST

TrainTest/No.OfFeatures	1	2	3	4	5	6	7	8	9	10
TrainFold1	0.113924051	0.101265823	0.091772152	0.075949367	0.075949367	0.0759493671	0.075949367	0.082278481	0.069620253	0.063291139
TestFold1	0.2	0.2	0.228571429	0.171428571	0.171428571	0.1714285714	0.171428571	0.2	0.228571429	0.2
TrainFold2	0.101265823	0.117088608	0.082278481	0.091772152	0.085443038	0.0759493671	0.082278481	0.075949367	0.07278481	0.066455696
TestFold2	0.142857143	0.085714286	0.085714286	0.057142857	0.057142857	0.0571428571	0.057142857	0.057142857	0.057142857	0.057142857
TrainFold3	0.14556962	0.107594937	0.079113924	0.094936709	0.075949367	0.0791139241	0.060126582	0.075949367	0.07278481	0.050632911
TestFold3	0.228571429	0.142857143	0.171428571	0.171428571	0.142857143	0.1428571429	0.142857143	0.142857143	0.142857143	0.142857143
TrainFold4	0.113924051	0.101265823	0.098101266	0.088607595	0.085443038	0.085443038	0.079113924	0.082278481	0.07278481	0.079113924
TestFold4	0.142857143	0.114285714	0.057142857	0.057142857	0.057142857	0.0571428571	0.057142857	0.085714286	0.057142857	0.085714286
TrainFold5	0.123417722	0.117088608	0.082278481	0.091772152	0.079113924	0.0727848101	0.069620253	0.069620253	0.069620253	0.069620253
TestFold5	0.142857143	0.142857143	0.085714286	0.114285714	0.114285714	0.0857142857	0.085714286	0.085714286	0.085714286	0.085714286
TrainFold6	0.155555556	0.092063492	0.098412698	0.092063492	0.079365079	0.0825396825	0.092063492	0.066666667	0.076190476	0.066666667
TestFold6	0.055555556	0.055555556	0.055555556	0.055555556	0.055555556	0.0555555556	0.055555556	0.055555556	0.055555556	0.055555556
TrainFold7	0.117088608	0.101265823	0.082278481	0.075949367	0.082278481	0.0727848101	0.079113924	0.063291139	0.066455696	0.069620253
TestFold7	0.257142857	0.228571429	0.228571429	0.114285714	0.2	0.1142857143	0.114285714	0.085714286	0.114285714	0.085714286
TrainFold8	0.164556962	0.088607595	0.117088608	0.094936709	0.082278481	0.085443038	0.085443038	0.079113924	0.085443038	0.069620253
TestFold8	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.0857142857	0.085714286	0.085714286	0.085714286	0.085714286
TrainFold9	0.132911392	0.123417722	0.088607595	0.082278481	0.07278481	0.0727848101	0.079113924	0.082278481	0.07278481	0.066455696
TestFold9	0.142857143	0.114285714	0.057142857	0.057142857	0.028571429	0.0285714286	0.028571429	0.028571429	0.028571429	0.028571429
TrainFold10	0.158227848	0.098101266	0.110759494	0.082278481	0.082278481	0.0791139241	0.079113924	0.063291139	0.07278481	0.060126582
TestFold10	0.228571429	0.114285714	0.228571429	0.142857143	0.085714286	0.0857142857	0.142857143	0.114285714	0.114285714	0.114285714
Mean(TrainTest)	0.147671288	0.105994849	0.100673553	0.086251301	0.081786217	0.0757288071	0.07832758	0.076453961	0.077322502	0.072857873
Std(TrainTest)	0.048406829	0.044750564	0.058691626	0.03802691	0.04261456	0.0341776805	0.037043306	0.038252014	0.043547048	0.039008844

11	12	13	14	15	16	17	18	19	20	21	22
0.066455696	0.079113924	0.085443038	0.069620253	0.060126582	0.075949367	0.069620253	0.069620253	0.060126582	0.056962025	0.060126582	0.056962025
0.171428571	0.228571429	0.2	0.171428571	0.171428571	0.2	0.142857143	0.228571429	0.2	0.142857143	0.142857143	0.171428571
0.066455696	0.066455696	0.069620253	0.085443038	0.056962025	0.056962025	0.063291139	0.056962025	0.060126582	0.082278481	0.082278481	0.085443038
0.057142857	0.057142857	0.057142857	0.085714286	0.057142857	0.057142857	0.057142857	0.057142857	0.057142857	0.085714286	0.085714286	0.085714286
0.044303797	0.047468354	0.047468354	0.053797468	0.047468354	0.053797468	0.075949367	0.079113924	0.075949367	0.07278481	0.075949367	0.079113924
0.142857143	0.142857143	0.142857143	0.142857143	0.171428571	0.142857143	0.142857143	0.142857143	0.142857143	0.142857143	0.142857143	0.142857143
0.060126582	0.056962025	0.053797468	0.066455696	0.082278481	0.085443038	0.053797468	0.056962025	0.085443038	0.085443038	0.085443038	0.085443038
0.057142857	0.057142857	0.085714286	0.085714286	0.085714286	0.085714286	0.057142857	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286
0.069620253	0.060126582	0.075949367	0.056962025	0.063291139	0.060126582	0.082278481	0.056962025	0.082278481	0.079113924	0.060126582	0.085443038
0.085714286	0.085714286	0.085714286	0.085714286	0.057142857	0.085714286	0.085714286	0.057142857	0.085714286	0.085714286	0.085714286	0.114285714
0.079365079	0.06984127	0.057142857	0.06984127	0.073015873	0.063492063	0.06031746	0.050793651	0.066666667	0.053968254	0.063492063	0.06031746
0.055555556	0.055555556	0.027777778	0.055555556	0.055555556	0.055555556	0.027777778	0.027777778	0.055555556	0.027777778	0.027777778	0.055555556
0.069620253	0.060126582	0.060126582	0.060126582	0.056962025	0.085443038	0.047468354	0.053797468	0.079113924	0.082278481	0.079113924	0.082278481
0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286
0.066455696	0.060126582	0.082278481	0.063291139	0.056962025	0.079113924	0.079113924	0.082278481	0.082278481	0.082278481	0.079113924	0.079113924
0.028571429	0.028571429	0.114285714	0.057142857	0.028571429	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714
0.066455696	0.063291139	0.066455696	0.075949367	0.060126582	0.085443038	0.060126582	0.085443038	0.082278481	0.085443038	0.088607595	0.085443038
0.028571429	0.028571429	0	0.028571429	0	0.114285714	0	0.085714286	0.085714286	0.085714286	0.114285714	0.085714286
0.063291139	0.07278481	0.050632911	0.056962025	0.050632911	0.063291139	0.075949367	0.075949367	0.047468354	0.079113924	0.082278481	0.079113924
0.114285714	0.114285714	0.142857143	0.085714286	0.142857143	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714
0.067233364	0.06911018	0.072317205	0.070117084	0.066517343	0.080209875	0.067985917	0.075776755	0.079487004	0.078649972	0.079806199	0.083373975
0.036429821	0.045138868	0.045316973	0.034234704	0.043770605	0.03869201	0.036668242	0.044690255	0.037940966	0.031451592	0.032087294	0.033016198

23	24	25	26	27	28	29	30	31	32	33	34
0.056962025	0.060126582	0.056962025	0.060126582	0.056962025	0.088607595	0.053797468	0.091772152	0.088607595	0.088607595	0.085443038	0.094936709
0.085714286	0.171428571	0.142857143	0.171428571	0.114285714	0.028571429	0.085714286	0.028571429	0.028571429	0.028571429	0.028571429	0.028571429
0.082278481	0.082278481	0.085443038	0.082278481	0.082278481	0.088607595	0.085443038	0.088607595	0.082278481	0.091772152	0.088607595	0.088607595
0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286
0.07278481	0.075949367	0.079113924	0.079113924	0.079113924	0.082278481	0.082278481	0.085443038	0.082278481	0.085443038	0.082278481	0.085443038
0.142857143	0.142857143	0.142857143	0.142857143	0.142857143	0.142857143	0.142857143	0.142857143	0.142857143	0.142857143	0.142857143	0.142857143
0.082278481	0.085443038	0.088607595	0.085443038	0.085443038	0.088607595	0.088607595	0.082278481	0.085443038	0.088607595	0.091772152	0.085443038
0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286
0.088607595	0.088607595	0.088607595	0.085443038	0.088607595	0.085443038	0.088607595	0.088607595	0.088607595	0.088607595	0.088607595	0.088607595
0.114285714	0.114285714	0.085714286	0.085714286	0.114285714	0.085714286	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714
0.085714286	0.06031746	0.082539683	0.088888889	0.082539683	0.088888889	0.088888889	0.088888889	0.095238095	0.095238095	0.095238095	0.088888889
0.055555556	0.027777778	0.055555556	0.055555556	0.055555556	0.055555556	0.055555556	0.055555556	0.055555556	0.055555556	0.055555556	0.055555556
0.079113924	0.082278481	0.079113924	0.082278481	0.079113924	0.082278481	0.088607595	0.082278481	0.079113924	0.082278481	0.085443038	0.085443038
0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286	0.085714286
0.079113924	0.082278481	0.085443038	0.079113924	0.082278481	0.082278481	0.085443038	0.088607595	0.085443038	0.085443038	0.085443038	0.085443038
0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714
0.085443038	0.082278481	0.082278481	0.088607595	0.085443038	0.085443038	0.085443038	0.085443038	0.088607595	0.085443038	0.085443038	0.085443038
0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714
0.075949367	0.079113924	0.082278481	0.082278481	0.082278481	0.079113924	0.085443038	0.085443038	0.082278481	0.085443038	0.088607595	0.085443038
0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714	0.114285714
0.081211756	0.08341005	0.083516905	0.084960363	0.083229218	0.080192979	0.083226022	0.082210897	0.08178028	0.082643341	0.082643341	0.082498584
0.027603745	0.035208289	0.029397982	0.032294516	0.028064861	0.028704937	0.027448988	0.029406187	0.029489136	0.029452229	0.029452229	0.029413945