# Deep Learning Project Report

## Data Visualization:

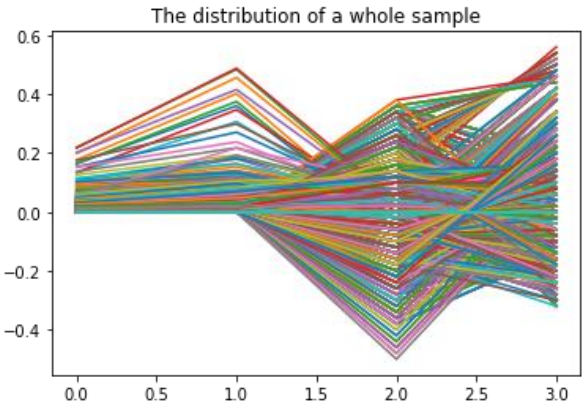


Fig. 1. The Line plot of an entire sample of dependent variables

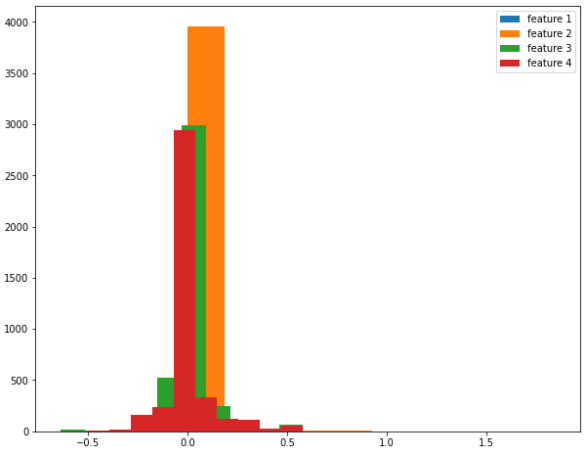


Fig. 2. Histogram of a X sample 1 from a set 4

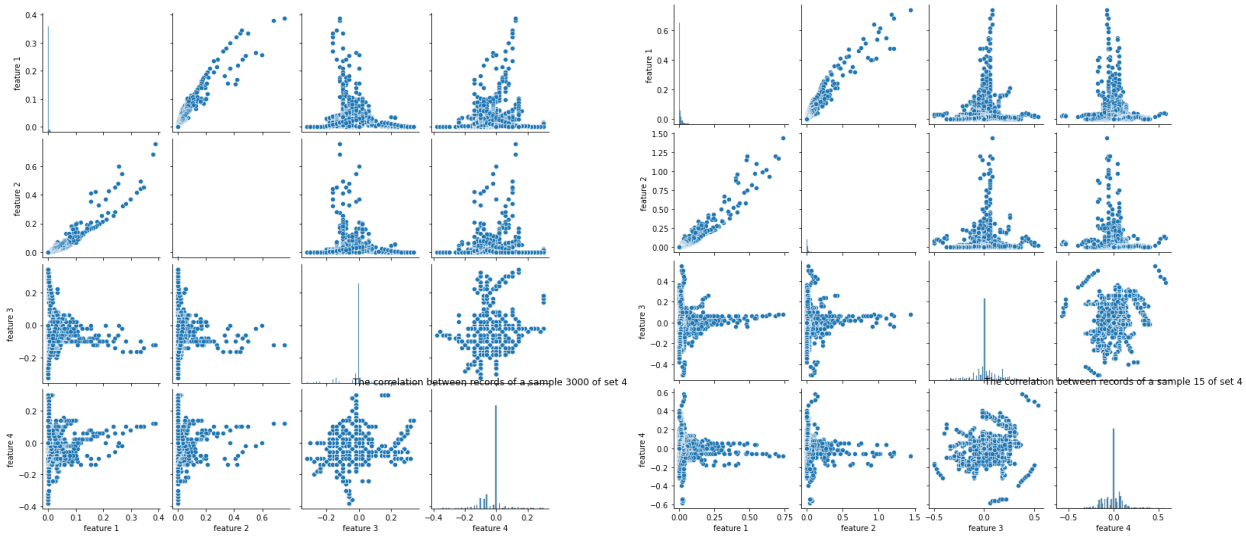


Fig. 3. Correlation between records of a sample 3000(left) and sample 15(right) of the set 4.

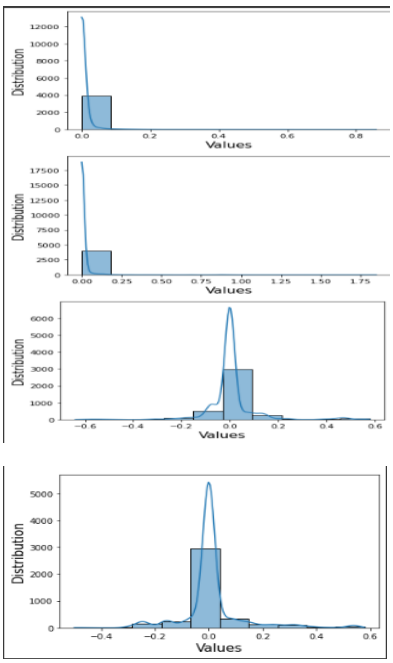


Fig. 4. Distplot of a sample 0 from set 4

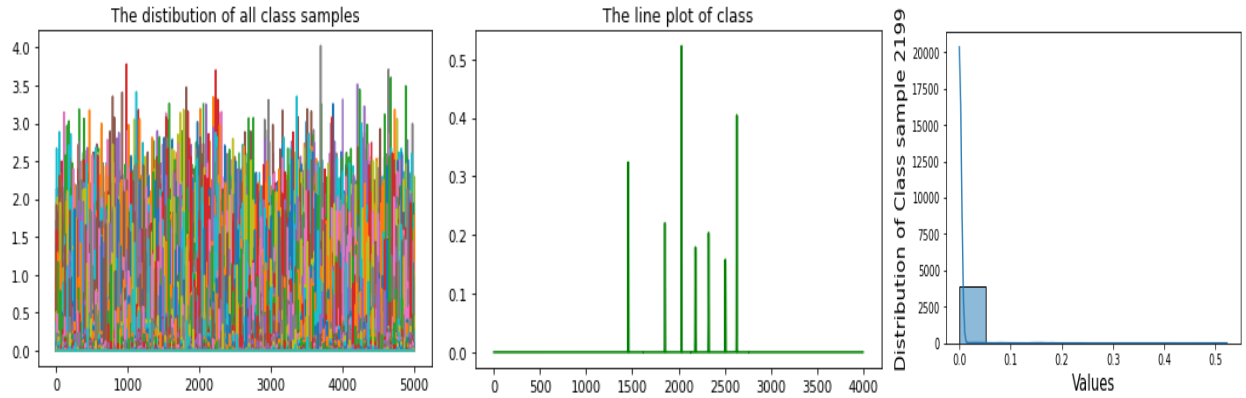


Fig. 5. The line plot of all class samples of set 4(left), The line plot of a sample class(middle), and distplot of a sample 2199 of set 4(right).

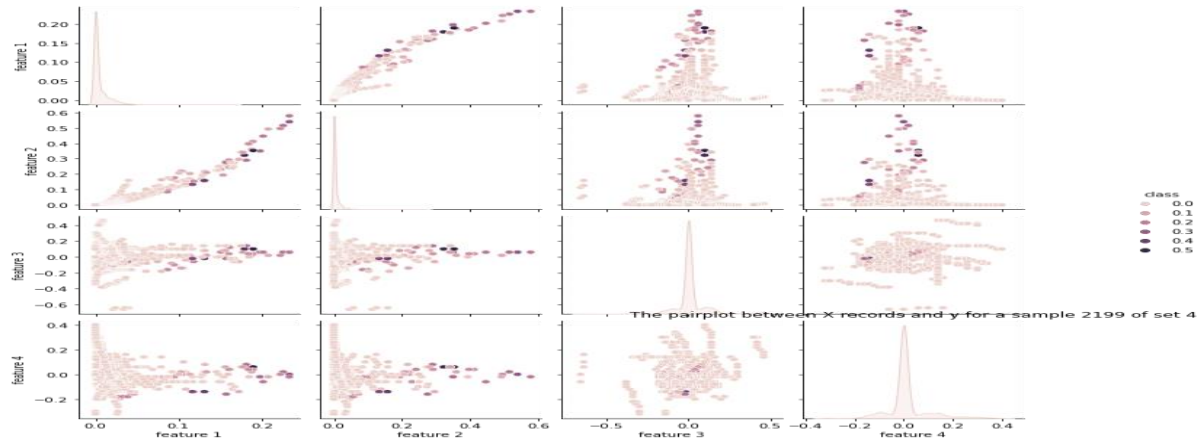


Fig. 6. The pair plot of the features and class of sample 2199 of set 4.

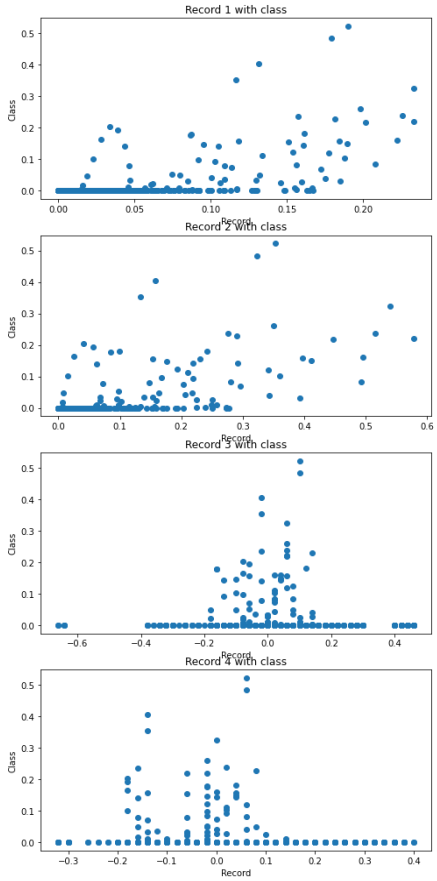


Fig. 7. Correlation between records and class of sample 2199 of set 4.

## Analysis:

From the Fig. 1 we can see that for the non-label data from the sample 1 of set 4, more data aggregated between the values 1 and 2.5 with distribution from -0.4 to 0.4. So, the model can learn useful information from that distribution, and after 2.5 the distribution is deviating from one other. By looking at Fig. 3. we can see that the record 1 and record 2 are always correlated in each sample data as they both give the same information which is more like a redundant data in the dataset. From the distplot of the records of sample 0 we can see that values in record 1 and record 2 are in between 0 and 0.25 whereas the record 3 and 4 are having gaussian distribution between 0 and 0.6. From the Fig. 5 (left) we can see that all the class samples of a set 4 are distributed between 0 and 4 and from the Fig. 7 we can illustrate that for every sample there is a correlation between the feature 1, feature 2 and class label.

From this whole data visualization, we can understand that there is a pattern in the class label and the pattern identification and useful feature extraction can be done greatly with CNN and done well with MLP. But as MLP is simple and easy to implement, MLP was taken as the initial model.

## Models:

➔ Model1:

```
MLP10(
  (my_network): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=16000, out_features=8000, bias=True)
    (2): Dropout(p=0.3, inplace=False)
    (3): ReLU()
    (4): Linear(in_features=8000, out_features=6000, bias=True)
    (5): Dropout(p=0.4, inplace=False)
    (6): ReLU()
    (7): Linear(in_features=6000, out_features=4000, bias=True)
    (8): Dropout(p=0.6, inplace=False)
    (9): ReLU()
    (10): Linear(in_features=4000, out_features=4000, bias=True)
  )
)
```

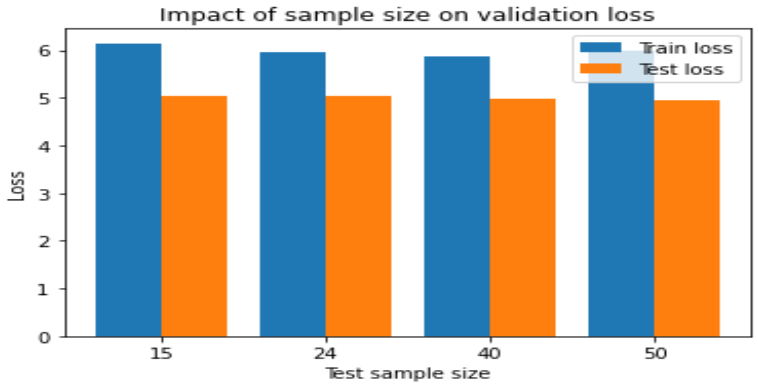


Fig. 8. Impact of sample size on the model 1 performance

In this model1 MLP was used as the dataset is a continuous dataset with 4000 classes to predict for each sample. As MLP works well for large datasets with many classes, the MLP was chosen for this model. Three hidden layers were used with sizes 8000, 6000, and 4000 respectively. The reason for choosing very minimal sizes for hidden layers is that when the sizes were increased the loss was also increasing. And the model is trained for 2 epochs as the loss was there no use of running for more epochs. The ReLU activation is used after each hidden layer as it was better than other activations like (TanH, Sigmoid, and LeakyRELU). And Dropout with combination of (0.3,0.4,0.6) was used as it was better than lower combinations like (0.2,0.2,0.4). Even tried using He weight initialization in the model, but it worsened the loss of the final model. And even Batch norm didn't improve the performance of the model as it was giving higher loss like 12, 13 etc. The RMSprop with learning rate of 0.00001 gave slightly better convergence than other optimizers like SGD, Adam.

From the Fig. 8 we can see that the model's performance (validation loss) is almost same (6 for training and 5 for validation) for all test sample sizes. So, we can choose any number of samples for the test set for our model to perform better.

➔ Model2:

```
CNN1(
  (features): Sequential(
    (0): Conv2d(1, 2, kernel_size=(3, 3), stride=(1, 1), padding=same)
    (1): Dropout2d(p=0.3, inplace=False)
    (2): ReLU(inplace=True)
    (3): BatchNorm2d(2, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(2, 3, kernel_size=(3, 3), stride=(1, 1), padding=same)
    (6): Dropout2d(p=0.3, inplace=False)
    (7): ReLU(inplace=True)
    (8): BatchNorm2d(3, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(3, 4, kernel_size=(3, 3), stride=(1, 1), padding=same)
    (11): ReLU(inplace=True)
    (12): Flatten(start_dim=1, end_dim=-1)
  )
)
```

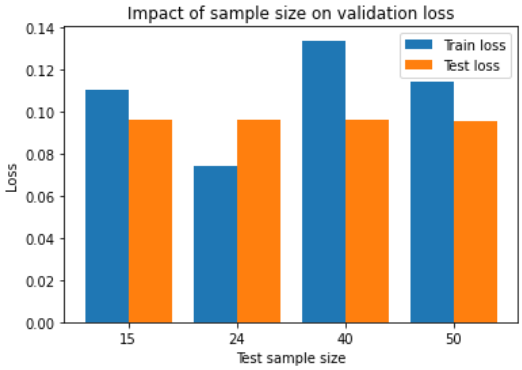


Fig. 9. Impact of sample size on the model 2 performance

From the previous model's performance, we can say that the MLP doesn't perform well for this type of dataset, So, a CNN model was used in this case. The input for the model is given as n, c, h, w format where n is the batch size, c is the number of channels which was given as, h is given as 4, and w is given as 4000. Then the data is passed to the CNN model with three convolutional layers with channels transitions (1,2), (2,3), (3,4) for the layer1, layer2, and layer3 respectively. When the number of channels were increased for the layers, the loss was also increasing, so only minimal number of channels were used. And the novelty in this model is that there is no classifier for computing logits as maxpooling layer was used after convolutional layer 1 and after convolutional layer 2 to reduce the dimensions of the input i.e. (128, 1, 4, 4000) to (128, 2, 2, 4000) and to (128, 3, 1, 4000). Which makes it equal to the dimensions of the class label. And when classifier with linear layers were used, the loss was very high. When it comes to choosing the batch size for minibatches there was no difference in loss no matter what ever we choose. And the SGD optimizer with the learning rate of 0.001 gave better convergence than other optimizers like RMSprop, SGD with momentum, and Adam. The combination of batchnorm2d and Dropout2d gave better performance than batchnorm2d alone, so used the combination with Dropout2d p=0.3 after first convolutional layer and p=0.3 after second convolutional layer. The ReLU () activation was used in this model, but other activations also gave same performance.

From the Fig. 9. we can see illustrate that here also the validation loss is not impacted by the sample size of the test set because the model is training fully in the first batches itself so no matter what sample size, we provide we get the same validation loss.

➔ Model3:

```
CONVAE(
  (encoder): Sequential(
    (0): Conv2d(1, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.01)
    (2): Conv2d(4, 6, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (3): LeakyReLU(negative_slope=0.01)
    (4): Conv2d(6, 6, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (5): LeakyReLU(negative_slope=0.01)
    (6): Conv2d(6, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): Flatten(start_dim=1, end_dim=-1)
    (8): Linear(in_features=8000, out_features=64, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=64, out_features=8000, bias=True)
    (1): Reshape()
    (2): ConvTranspose2d(8, 6, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): LeakyReLU(negative_slope=0.01)
    (4): ConvTranspose2d(6, 6, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (5): LeakyReLU(negative_slope=0.01)
    (6): ConvTranspose2d(6, 4, kernel_size=(3, 3), stride=(2, 2))
    (7): LeakyReLU(negative_slope=0.01)
    (8): ConvTranspose2d(4, 1, kernel_size=(3, 3), stride=(1, 1))
    (9): Trim()
    (10): Conv2d(1, 2, kernel_size=(3, 3), stride=(1, 1), padding=same)
    (11): Dropout2d(p=0.3, inplace=False)
    (12): ReLU(inplace=True)
    (13): BatchNorm2d(2, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (15): Conv2d(2, 3, kernel_size=(3, 3), stride=(1, 1), padding=same)
    (16): Dropout2d(p=0.3, inplace=False)
    (17): ReLU(inplace=True)
    (18): BatchNorm2d(3, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (20): Conv2d(3, 4, kernel_size=(3, 3), stride=(1, 1), padding=same)
    (21): ReLU(inplace=True)
    (22): Flatten(start_dim=1, end_dim=-1)
  )
)
```

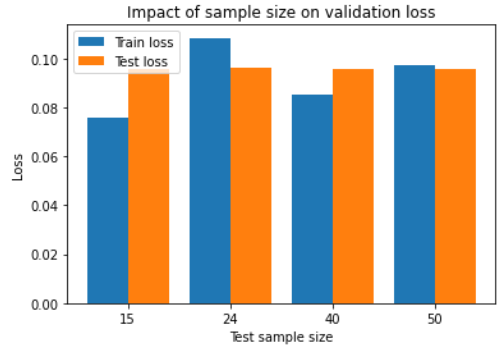


Fig. 10. Impact of sample size on the model 3 performance

In this model we have used convolutional autoencoder with another convolutional network by thinking that the model will learn useful parameters and weights with the help of latent space and from the previous model we can depict that CNN works well for this dataset. So, combing these two models would give great results and this is the novelty for this model. In the encoder block four convolutional layers and one linear layer was used, by narrowing down the 4*4000 input to a 2*4000 size and encoding it into a 64 feature latent space. The LeakyReLU was used as the activation function in the encoder and decoder as it performs well on autoencoder models. In the decoder is the 64 feature latent space is decoded into its original shape with the help of a linear layer and four convolutional transpose layers. After the decoder part the same CNN model setup as the previous model was used, as it performed better and gave good loss. The ReLU is used as an activation function for the convolutional part as it gave better loss in the previous model and Batchnorm2d was used in the convolutional part as the loss didn't converge as expected when it was removed. The RMSprop optimizer with learning rate of 0.001 was used as it gave better convergence than Adam and SGD.

From the Fig. 10. We can see that the validation loss is not impacted by the sample size of the test set as all the training is happening in the early batches itself so there nothing for the model to learn.

➔ Model4:

```
Resnet10(
  (block_1): Sequential(
    (0): Conv1d(1, 2, kernel_size=(2,), stride=(1,))
    (1): BatchNorm1d(2, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv1d(2, 1, kernel_size=(1,), stride=(1,), padding=(1,))
    (4): Trim()
    (5): BatchNorm1d(1, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (block_2): Sequential(
    (0): Conv1d(1, 2, kernel_size=(2,), stride=(1,))
    (1): BatchNorm1d(2, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv1d(2, 1, kernel_size=(2,), stride=(1,), padding=(1,))
    (4): BatchNorm1d(1, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (block_3): Sequential(
    (0): Conv1d(1, 2, kernel_size=(1,), stride=(1,))
    (1): BatchNorm1d(2, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): ReLU(inplace=True)
    (4): Conv1d(2, 1, kernel_size=(2,), stride=(1,), padding=(1,))
    (5): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): BatchNorm1d(1, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
```

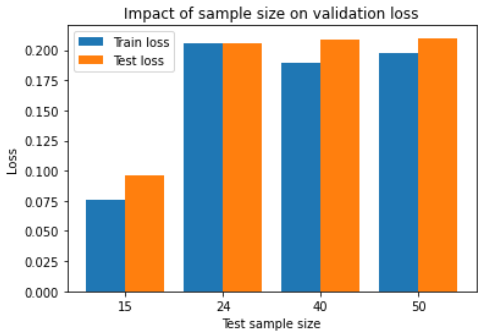


Fig. 11. Impact of sample size on the model 4 performance

For this case the residual neural networks are used to build the model as from the previous model we can see that CNN models are working well for this type of dataset. For this model two residual blocks (block 1 and block 2) for performing skip connections of the network were used with a final block (block 3) which converts (128, 1, 16000) input to a (128, 1, 4000) size output which is the same as the size of the class label. The novelty in this model is that it is a combination of the regular residual neural network and a convolutional reducer, which uses the skip connection advantage of resnet and useful feature extraction from the CNN block. The two residual blocks are having two convolutional layers with channels transitions (1,2), (2,1) for the layer1, layer2 respectively, and Batchnorm1d was used after each convolutional layer. Here all 1d layers were used as the data was transformed to 3 dimensional by multiplying 4 and 4000 of the input data. Also, in block 3 two convolutional layers with channels transitions (1,2), (2,1) for the layer1, layer2 respectively, and Batchnorm1d was used after each convolutional layer. Also, for reducing the size of the input two maxpool1d layers were used after each convolutional layer. The 'He' weight initialization helped the model to start the loss from a very less value. Adam optimizer with 0.001 learning rate gave better convergence than SGD with momentum, RMS prop, but gave same convergence as SGD. Lastly the ReLU activation was used after each convolutional layer as it gave better performance than other activation functions.

From the Fig. 11. We can illustrate that the validation loss is lowest if the sample size is given low as 15 and it increases if the sample size is increases. But stays constant after the 40 size mark. For better performance of the model, we must choose the sample size as low as 15.