Aggie Code of Honor:

"An Aggie does not lie, cheat, or steal or tolerate those who do."

First Name: KIRAN *Last Name:* KONDISETTI *UIN:*430000208

Any assignment turned in without a fully completed cover page will NOT BE GRADED.
Please list all below all sources (people, books, web pages, etc) consulted regarding this assignment:

CSCE 633-601 Students	Other People	Printed Material	Web Material (URL)	Other
1. Peter German	1.	1.	1. https://tinyurl.com/s485mxb	1.
2. Nelson Dsouza	2.	2.	2. https://towardsdatascience.com	2.
3.	3.	3.	3. www.spss-tutorials.	3.
4.	4.	4.	4.	4.
5.	5.	5.	5.	5.

Please consult the Aggie Honor System Office for additional information regarding academic misconduct – it is your responsibility to understand what constitutes academic misconduct and to ensure that you do not commit it.

I certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received nor given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment.

Date: _04/30/2020___________________

*Signature:*___KIRAN KONDISETTI________________________

Literature Review

`Paper Title`- Combining Decision Tree and Back Propagation Genetic Algorithm Neural Network for Recognizing Word Gestures in Indonesian Sign Language using Kinect(`http://www.jatit.org/volumes/Vol95No2/6Vol95No2.pdf`)

This research paper focuses on how machine learning algorithm can be used to translate ISL (Indonesian Sign Language) to text. In this experiment Kinect is used to capture the gestures of user skeleton. The distance values, angle values and vector values obtained from the skeletons are used as features to train the classifiers. The machine learning classifiers used in this experiment setup are decision tree and Back Propagation Neural Network (BPGANN). The resulting output obtained from these classifiers are tested by two people.

This study is conducted in two phases, namely- Training phase and the Testing phase. The sequential steps involved in these two cases are Feature Extraction, Feature Normalization, Rule Generation using Decision Tree and, Training and Testing using Neural Network. In Feature Extraction step, the features used to define sign languages are generated by capturing the skeleton joints by a Kinect (in this research only nine skeleton joints are used to generate the features). The next step, Feature normalization is used to scale the features obtained in the previous step using a Min-Max scaler. The rules used to minimize the overfitting in Neural Networks are generated using the decision tree in the next step. In the final step, two groups of training data which are grouped based on the rules, are used to train BPGANN to produce two classifiers . The resulting classifiers are tested on the testing data based on the rules generated. The results of the rules will determine whether the testing data will be classified using Classifier 1 or 2. A accuracy pf 96% is obtained by performing real time experiments.

Similar to the above experiment setup, an Ensemble learner consisting of N decision trees and an MLP learner is used for attack detection in the final project. Different type of experiments are conducted to find the best values for different hyper-parameters to get the optimum results after scaling the input data. Batch, Mini-batch, Stochastic gradient descent learning are employed to minimize the cost function

CSCE-633/601

Project: Novel Learner for Attack Detection Problem

Due: April 30, 2020

Submitted: April 30, 2020

Kiran Kondisetti (Industrial Engineering)

1 Introduction

In this project, a novel learner (ensemble learner) and MLP (multi-layer perceptron) learner are built on the attack detection data set. The Novel learner used in this project is an ensemble learner consisting of N trees and an MLP learner. This data set contains various features that describe a set of programs execution which are classified as either benign or malicious. The accuracies obtained from these models are then compared with the accuracies obtained from the models which were built in the previous assignments. Series of experiments are conducted on the above models to tune different hyper- parameters. Effect of batch, mini-batch and stochastic learning on the accuracies of the above models are also observed. Different scaling techniques are used to optimize the data set to get a better prediction accuracy.

The reminder of the report consists of `Section 2-` Structure of the code, `Section-3` Multi Layered Perceptron for attack detection, `Section 4-` Novel Learner for attack decision, `Section 5 -` Comparing the Results with other base learners, `Section 6-` Summary.

2 General remarks and the structure of the code

2.1 Code location

(`https://github.tamu.edu/kirankondisetti/CSCE633_KIRAN_KONDISETTI_FINAL_PROJECT`) The repository for the source code used to generate the results in this report is located under: Enter ur

2.2 Structure of the code

The code for this project is developed from the code of the previous assignment and is built around different python libraries. The code contains six wrapper classes for the six different types of classifiers (DecisionTreeWrapper.py, PerceptronWrapper.py, EnsembleWrapper.py, KNNWrapper.py, NovelWrapper.py, MLPWrapper.py) and DataBase.py, and `driver_script.py` that synchronizes the methods from these classes with the input file (InputDeck.py). DataBase.py handles the input data given in feature, program name and true value files.

The wrapper classes and the data base handling class are located in the `src` folder, while the driver script and input deck files have to be located in the dataset/newdata directory. The directory structure is depicted below:

```
dataset/
├─ src/
│  ├─ DataBase.py
│  ├─ DecisionTreeWrapper.py
│  ├─ PerceptronWrapper.py
│  ├─ EnsembleWrapper.py
│  ├─ KNNWrapper.py
│  ├─ MLPWrapper.py
│  └─ NovelWrapper.py
├─ outputs/
├─ driver_script.py
└─ InputDeck.py
```

There is an output directory in which outputs of the code are saved in either text or pdf file. The `driver_script.py` file contains a sequence of calls to print all the results necessary for this homework. The steps in this sequence are discussed in the corresponding chapters in detail. To execute the code one can, use the following command:

```
python3 driver_script.py arg1 arg2 arg3
```

where

`arg1`: The name of the file that contains the program names (or the root words of the program names) that we want to include in our data base.

`arg2`: The name of the file that contains the feature names we want to include in the data base.

`arg3`: The name of the file containing the output labels for each program name in the `arg1`. In this file the malware programs get a label of 1 and the harmless codes get a label of -1. The necessary files are attached in the submission/repository.

The only file the user is supposed to change is the `InputDeck.py` in the `dataset/newdata` folder. The effect of changing the parameters in this file is discussed in the corresponding subsections. The raw input file with all the available input features is the following:

```
# ----------------------------------------------------------------------------
# General parameters, must be specified!
# ----------------------------------------------------------------------------
read_csv = False # False: prepare db, True: read existing csv
confidence = 0.95 # Confidence interval for the t values
rng_seed = 0 # Starting seed for the random number generator
data_ratio = 0.7 # Ratio of the features to keep in the reduced fature data base
to_remove = "features" #"samples" # Which one to decrease ("features"/"samples"/None)

# ----------------------------------------------------------------------------
# Database parameters (they have default values)
# ---------------------------------------------------------------------------
keep_constant_features = False # Retain features with values that are the same for all samples
# DELETE THE OUTPUTS FOLDER WHEN YOU CHANGE THIS PARAMETER

# ---------------------------------------------------------------------------
# Cross Validation parameters (they have default values)
# ---------------------------------------------------------------------------
n_folds = 3 # Number of folds for the k-fold CV, default: 3
k_fold_type = "stratified" # K-fold cross validation (regular/stratified), default: regular
k_fold_shuffle = True # Shuffle before splitting, default: True

# ---------------------------------------------------------------------------
# Settings for the correlation filter (they have default values)
# ---------------------------------------------------------------------------
correlation_filter = None # Turns ("features"/"label"/None) the correlation filter
correlation_filter_value = 0.9 # The value we use to filter the features
correlation_filter_type = "pearson" # The definition of correlation we want to use (pearson/regular)
correlation_label_to_keep = 0.2 # The ratio of the most correlated (to the label) features to keep

# ---------------------------------------------------------------------------
# MLP and Novel Learner specific parameters (they have default values)
# ---------------------------------------------------------------------------
solver = 'sgd' #which solver to use options -{   lbfgs   ,   sgd   ,   adam   }, default = 'sgd'
batch_size =1 #Size of the batch
hidden_layer_sizes=(30,30,30) #number of hidden layers
max_iter  = 10000 #Maximum number of iterations. default = 200
learning_rate = 'adaptive' #{   constant   ,   invscaling   ,   adaptive   }, default=   constant
learning_rate_init = 0.001 #The initial learning rate used. default=0.001

# ---------------------------------------------------------------------------
# Data Scaling specific parameters (they have default values)
# ---------------------------------------------------------------------------
scale = False #True : if scaling is required, False  = if its not required
scale_name='StandardScaler' #type of scaling method, options: 'Min_Max' and 'StandardScaler'

# ---------------------------------------------------------------------------
# Novel learner specific parameters (they have default values)
# ---------------------------------------------------------------------------
bag_ratio = 0.3 # The number of samples used for each decision tree/total number of samples
n_trees = 30 # How many trees we want to build
```

2.3 Random number generator

The random selection of the samples in the k-fold cross validation, the random selection of samples in the stochastic gradient descent in the MLP, and the bagging in the Novel learner algorithm makes it difficult to compare different algorithms or different models. For this reason, we added an additional parameter that can be used to fix the seed of the random number generator in the input file ensuring that we can test the algorithms with the same data sets. To do this, an integer should be given to the value of `rng_seed` variable in the input file. If `None` is given, the code will use the default seed that changes at the beginning of every python run.

2.4 Overview of the data

There are 964 folders in the provided `newdata` directory. We filter out those with empty or non-existent `stats.txt` files. Altogether we get **948 samples** in the improved data base. The previous, **39 sample** data base is not affected by this, since every folder contains a non-empty `stats.txt` file. The distribution of benign and malicious samples in the two data bases is presented in Figure 2.

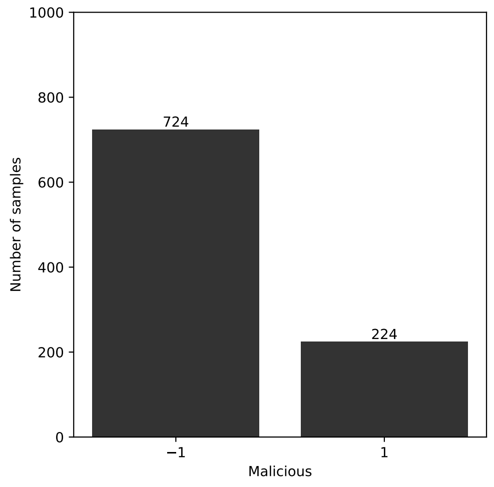


Figure 2: The distribution of the malicious (1) and benign (-1) samples for the two data bases.

By default we filter constant features out. It turns out that there are only **486 non-constant features** out of the 573 in the improved dataset.

3 Multi-Layer Perceptron for Attack Detection

3.1 Multi-Layer Perceptron

A multilayer perceptron (MLP) is a deep, artificial neural network. It is composed of more than one perceptron. They are composed of an input layer to receive the signal, an output layer that makes a decision or prediction about the input, and in between those two, an arbitrary number of hidden layers that are the true computational engine of the MLP. MLPs with one hidden layer are capable of approximating any continuous function[1].

Multilayer perceptrons are often applied to supervised learning problems,they train on a set of input-output pairs and learn to model the correlation (or dependencies) between those inputs and outputs. Training involves adjusting the parameters, or the weights and biases, of the model in order to minimize error. Backpropagation is used to make those weigh and bias adjustments relative to the error, and the error itself can be measured in a variety of ways, including by root mean squared error (RMSE).

Feedforward networks are mainly involved in two motions, a constant back and forth. In the forward pass, the signal flow moves from the input layer through the hidden layers to the output layer, and the decision of the output layer is measured against the ground truth labels[5].

In the backward pass, using backpropagation and the chain rule of calculus, partial derivatives of the error function w.r.t. the various weights and biases are back-propagated through the MLP. That act of differentiation gives us a gradient, or a landscape of error, along which the parameters may be adjusted as they move the MLP one step closer to the error minimum. This can be done with any gradient-based optimization algorithm such as stochastic gradient descent. The network keeps playing that game of tennis until the error can go no lower. This state is known as convergence.

3.2 Batch Gradient Descent

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).This is an iterative optimization algorithm for finding the minimum of a function. The algorithm takes steps proportional to the negative gradient of the function at the current point . In deep learning neural networks are trained by defining a loss function and optimizing the parameters of the network to obtain the minimum of the function.

The steps of the algorithm are:-

1. Find the slope of the objective function with respect to each parameter/feature. In other words, compute the gradient of the function.

2. Pick a random initial value for the parameters.

3. Update the gradient function by plugging in the parameter values.

4. Calculate the step sizes for each feature as : step size = gradient * learning rate.

5. Calculate the new parameters as : new params = old params -step size

6. Repeat steps 3 to 5 until gradient is almost 0.

3.3 Importance of Learning rate

The “learning rate” is a flexible parameter which heavily influences the convergence of the algorithm. Larger learning rates make the algorithm take huge steps down the slope and it might jump across the minimum point thereby missing it. So, it is always good to stick to low learning rate such as 0.01. It can also be mathematically shown that gradient descent algorithm takes larger steps down the slope if the starting point is high above and takes baby steps as it reaches closer to the destination to be careful not to miss it and also be quick enough.

3.4 Stochastic Gradient Descent (SGD)

There are a few downsides of the gradient descent algorithm. If the data set is large, then the gradient descent algorithm has to find the gradient of all the data points in the data set to take a single step thus increasing the computational time.

To overcome this problem, we use Stochastic gradient descent (SGD). SGD randomly picks one data point from the whole data set at each iteration to reduce the computations enormously.

It is also common to sample a small number of data points instead of just one point at each step and that is called "mini-batch" gradient descent. Mini-batch tries to strike a balance between the goodness of gradient descent and speed of SGD.

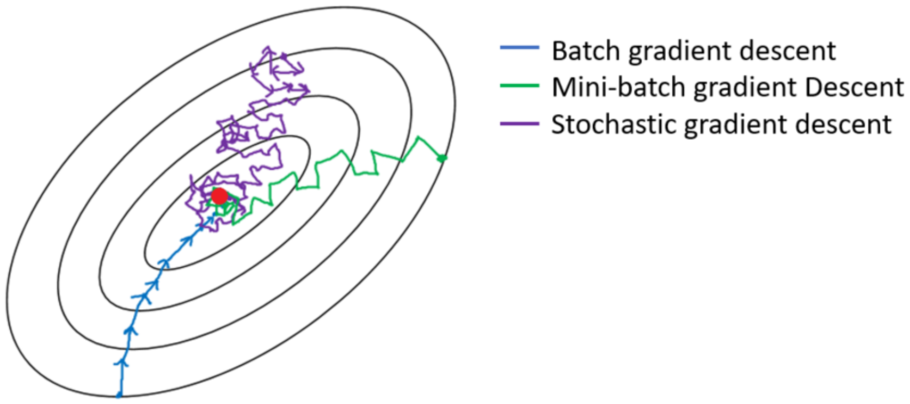


Figure 3: Gradient Descent Algorithm and Its Variants[5] .

From Figure 2, we can see that the oscillations in hypothesis space for stochastic gradient descent is more than batch and min-batch gradient descent. Due to this, there is a high probability that stochastic gradient descent in some cases do not converge to a global minimum. For this reason Min-batch gradient descent is a better option for minimizing the cost function in some cases.

3.5 Details of implementation

A wrapper class has been created (`MLPWrapper`) to handle the interfacing with `scikit-learn` libraries.

The inputs required for this wrapper are as follows: -

1. `random_state`- The random selection of the samples in the k-fold cross validation.
2. `solver`- Solver used for weight optimization. There are three types of solvers, namely - 'lbfgs' (an optimizer in the family of quasi-Newton methods), 'sgd' (stochastic gradient descent) and 'adam' (stochastic gradient-based optimizer proposed by Kingma).
3. `batch_size`- Batch size used in gradient descent.
4. `hidden_layer_sizes`- This is used to specify number of hidden layers and the hidden layer sizes. max_iter- Maximum number of iterations. The solver iterates until convergence (determined by 'tol') or this number of iterations.

5. `learning_rate_init`- The initial learning rate used. It controls the step-size in updating the weights.

Changes to these inputs can be done using `InputDeck.py`

```
# ------------------------------------------------------------------------
# MLP and Novel Learner specific parameters (they have default values)
# ------------------------------------------------------------------------
solver = 'sgd' #which solver to use options -{    lbfgs    ,     sgd    ,     adam    },
    default = 'sgd'
batch_size =10 #Size of the batch
hidden_layer_sizes=(30,30,30,30,30) #number of hidden layers
max_iter  = 10000 #Maximum number of iterations. default = 200
learning_rate = 'adaptive' #{    constant    ,     invscaling    ,     adaptive    },
    default=    constant
learning_rate_init = 0.001 #The initial learning rate used. default=0.001
```

The class contains five main functions (for more information see commented source code):

1. `fit`: Builds a decision tree using all the available data (every program/feature in the data base).
2. `select_model_cv`: Selects a model with the least using k-fold cross-validation.
3. `print_statistics`: Prints the statistics for the model selection.
4. `print_sorted_attributes`: Prints a sorted list of the used attributes, weighting them with their distance from the roots.

The outputs of the MLP are saved as –

1. **MLP-sorted-weights.txt**: : Contains the attributes and sorted weights of MLP.
2. **MLP-sorted-weights-absolute.txt** :: Contains the attributes and absolute sorted weights of MLP.

MLP model used in this project uses RELU function as the activation function (`Note`-Expect for change of layers experiment, three layered MLP with thirty hidden neurons in each layer is used for performing the experiments).

3.5.1 Reason for selecting RELU activation function

Certain activation functions, like the sigmoid function, squishes a large input space into a small input space between 0 and 1. Therefore, a large change in the input of the sigmoid function will cause a small change in the output. Hence, the derivative becomes small.

Gradients of neural networks are found using backpropagation. Simply put, backpropagation finds the derivatives of the network by moving layer by layer from the final layer to the initial one. By the chain rule, the derivatives of each layer are multiplied down the network (from the final layer to the initial) to compute the derivatives of the initial layers. However, when n hidden layers use an activation like the sigmoid function, n small derivatives are multiplied together. Thus, the gradient decreases exponentially as we propagate down to the initial layers.

Small gradient means that the weights and biases of the initial layers will not be updated effectively with each training session. Since these initial layers are often crucial to recognizing the core elements of the input data, it can lead to overall inaccuracy of the whole network. This problem is called `Vanishing Gradient Problem`. To overcome this problem RELU activation function is used.

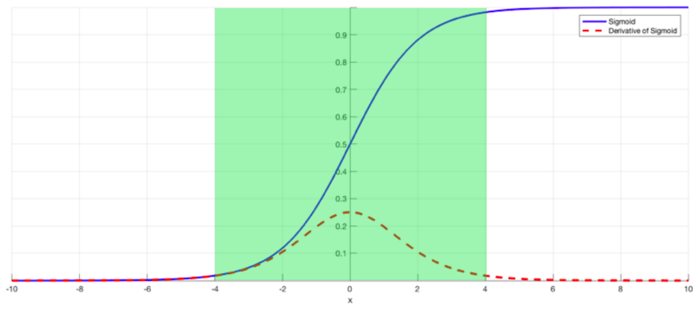


Figure 4: Sigmoid function with restricted inputs[3].

.

4 Novel Learner

4.1 Details of implementation

The Novel learner used in this project is an ensemble learner consisting of N trees and an MLP learner. When an input sample is inputted to the novel learner, first the decision trees gives predictions and then the MLP takes these predictions as input and uses relu function to give a final prediction.

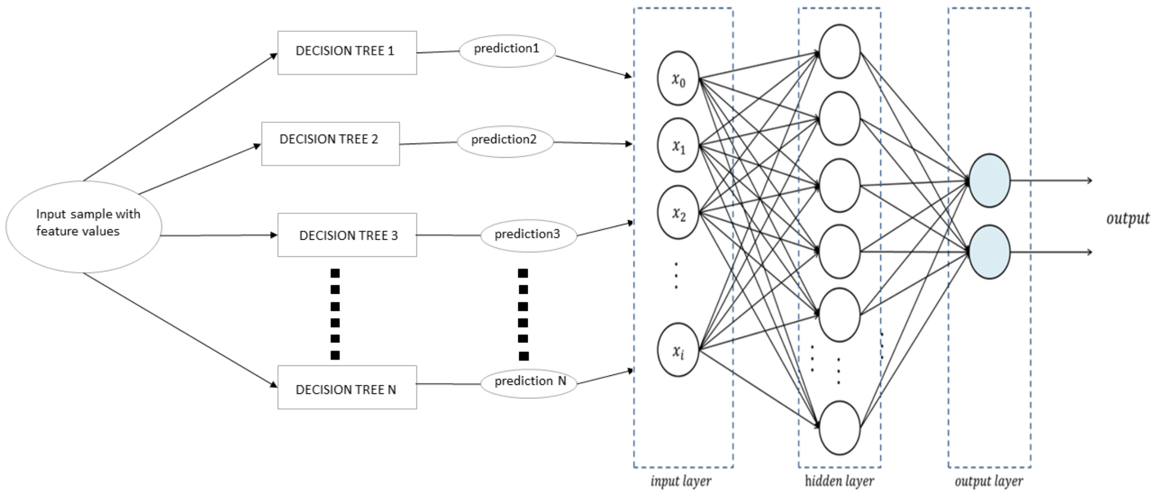


Figure 5: The structure of the Novel learner combining MLP and decision trees.

A wrapper class has been created (NovelWrapper.py) to handle the interfacing with scikit-learn and our custom functions. The main functions in the class are:

1. `init meta learner:` Initializes the list of decision trees and the MLP.
2. `fit meta learner:` Fits the Novel learner using an input training set.
3. `fit:` Fits the ensemble learner using the entire data set available.
4. `predict`: Predicts the class of an unknown data sample.
5. `select model cv:` Selects a model using k-fold cross-validation. 6.
6. `print statistics :` Prints the statistics for the model selection. Including average accuracy and confidence interval. 7.
7. `plot tree convergence:` Plots the average and best accuracy as function of number of trees used.

The inputs required for this wrapper are as follows: -

1. `random_state-` The random selection of the samples in the k-fold cross validation.
2. `solver-` Solver used for weight optimization. There are three types of solvers, namely - 'lbfgs' (an optimizer in the family of quasi-Newton methods), 'sgd' (stochastic gradient descent) and 'adam' (stochastic gradient-based optimizer proposed by Kingma).
3. `batch_size-` Batch size used in gradient descent.
4. `hidden_layer_sizes-` This is used to specify number of hidden layers and the hidden layer sizes
5. `max_iter-` Maximum number of iterations. The solver iterates until convergence (determined by 'tol') or this number of iterations.
6. `learning_rate _init-` The initial learning rate used. It controls the step-size in updating the weights.
7. `bag_ratio {` It is the ratio between the number of samples used for each decision tree to total number of samples
8. `n_trees` = Number of trees used in the novel learner.

All the inputs are given in the Inputdeck.py.

```
# --------------------------------------------------------------------------
# MLP and Novel Learner specific parameters (they have default values)
# --------------------------------------------------------------------------
solver = 'sgd' #which solver to use options -{   lbfgs   ,    sgd   ,    adam   }, default = 'sgd'
batch_size =1 #Size of the batch
hidden_layer_sizes=(30,30,30) #number of hidden layers
max_iter  = 10000 #Maximum number of iterations. default = 200
learning_rate = 'adaptive' #{   constant   ,    invscaling   ,    adaptive   }, default=   constant
learning_rate_init = 0.001 #The initial learning rate used. default=0.001

# --------------------------------------------------------------------------
# Data Scaling specific parameters (they have default values)
# ----------------------------------------------------------------------------
scale = False #True : if scaling is required, False  = if its not required
scale_name='StandardScaler' #type of scaling method, options: 'Min_Max' and '
    StandardScaler'

# ----------------------------------------------------------------------------
# Novel learner specific parameters (they have default values)
# ----------------------------------------------------------------------------
bag_ratio = 0.3 # The number of samples used for each decision tree/total number of
    samples
n_trees = 30 # How many trees we want to build
```

MLP model used in this Novel learner uses Relu function as the activation function and by setting the `learning_rate` to adaptive, the learner is able to adapt online.(`Note-` Expect for change of layers experiment, three layered MLP with thirty hidden neurons in each layer is used for performing the experiments). The reason why RELU activation function is used is explained in the section `3.5.1`.

5 Feature selection

Feature selection in this project is done by removing correlated features from the data set. For this selection, Pearson correlation coefficient is used.

5.1 Feature selection in MLP

For feature selection, we will be utilizing the correlation filter. Correlated features can be removed by using in the input deck.In this experiment, Pearson correlation coefficient is used to filter the correlated features. The accuracies obtained are plotted in a graph.

```
correlation_filter = "features"
correlation_filter_value = 0.98 # The value we use to filter the features
correlation_filter_type = "pearson" # The definition of correlation we want to use (
    pearson/regular)
```

Table 1: Number of features used with the variation of correlation filter value for 486 attributes for 948 programs.

Correlation Filter Value	Number of features used
0.5	35
0.6	39
0.7	50
0.8	70
0.9	102

We see that the number of features increases with the increase in correlation filter threshold.

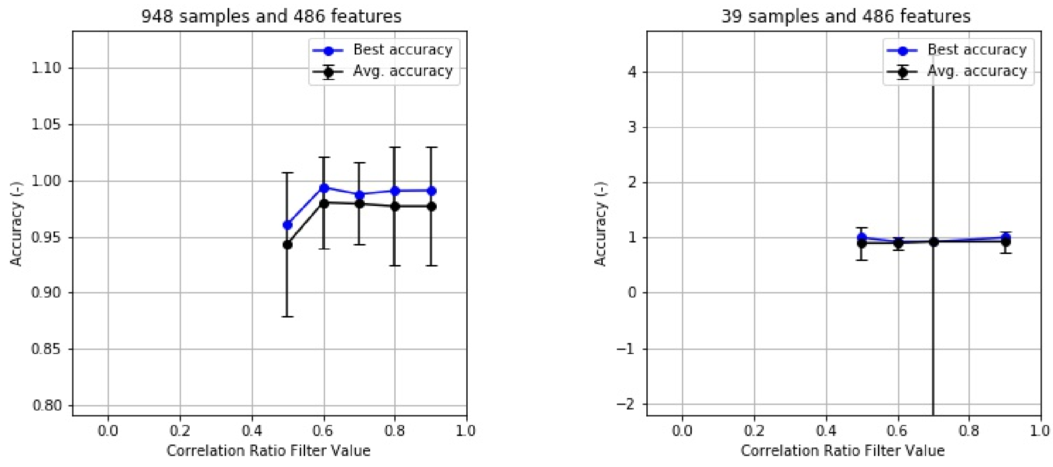


Figure 6: The average and best accuracy as functions as function of the correlation filter valueused in the data base.

For the data set containing 39 samples and 486 features, it is visible that there is an increase in average accuracy but the increase is not statistically significant. But for the bigger data set there is an increase in average accuracy when the threshold is changed from 0.5 to 0.6. After that there is no significant increase in the accuracy.
This proves that, accuracy of the model increases when the correlated features are removed from the data set.

5.2 Feature selection in Novel Learner

Feature selection for Novel learner is done by varying the correlation filter. Correlated features can be removed by using the input deck.

```
correlation_filter = "features"
correlation_filter_value = 0.98 # The value we use to filter the features
correlation_filter_type = "pearson" # The definition of correlation we want to use (
    pearson/regular)
```

In this experiment, Pearson correlation coefficient is used to filter the correlated features. The accuracies obtained are plotted in a graph.

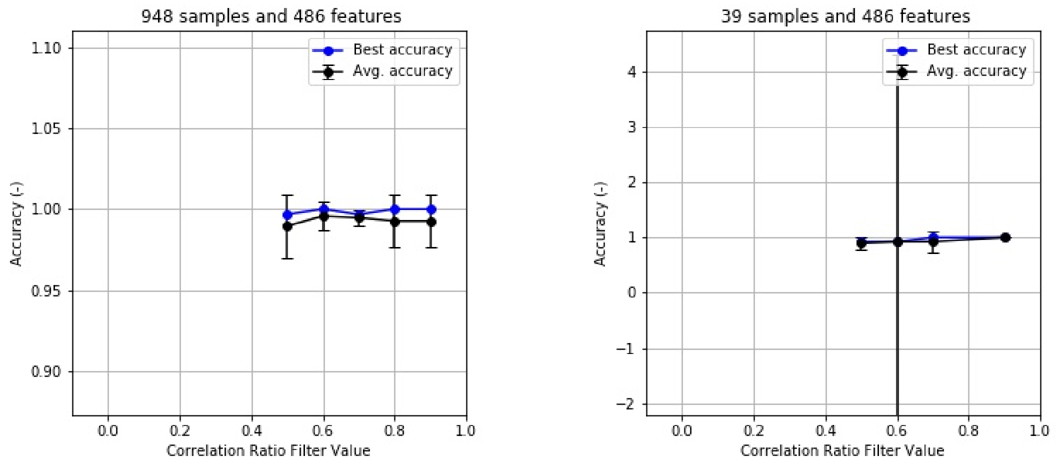


Figure 7: The average and best accuracy as functions as function of the correlation filter valueused in the data base.

From the above graph, it is clearly visible that the best and average accuracy increases as the correlation filter value increases. Similar trends can be observed in the small data set as well. But the increase is not statistically significant. This is because when the correlated features in the data set are removed, the accuracy of the model increases. This tells us that correlated features have a negative impact on the model accuracy.

`The training and testing time of different learner are tabulated before and after the features selection which is done by setting the correlation filter to 0.9. there by selecting 102 features (refer table 1),`

Table 2: Training and Testing Time before Feature Selection.

Serial.No	Learner	Training Time(sec)	Testing Time(sec)
1	Decision Tree	0.10478	0.20930
2	Perceptron	0.07610	0.19082
3	Ensemble Learner	1.7912	3.4707
4	MLP	4.9378	8.191
5	Novel Learner	4.425	8.36415

Table 3: Training and Testing Time after Feature Selection.

Serial.No	Learner	Training Time(sec)	Testing Time(sec)
1	Decision Tree	0.01296	0.0311
2	Perceptron	0.0588	0.1276
3	Ensemble Learner	0.33211	0.6921
4	MLP	1.8673	6.1362
5	Novel Learner	1.56694	3.6651

It can be seen from the tables that the training and testing time for all the learners have improved after feature selection. There is a drastic improvement in training and testing time for the MLP and the novel learner. This tells us that the feature selection not only increases the accuracy but also decreases the training and testing time of the learners.

6 Scaling the Data

In this project, Min-Max scalar and Standard scalar are used. The effect of scaling on the accuracy is observed to select the best scalar that is to be used in this project.

Min-Max scaler is used to scale the data. The Min-Max Scaler transforms features by scaling each feature to a given range. This scaler works better for cases when the standard deviation is very small.

`x_scaled` = (x-min(x)) / (max(x)–min(x))

Standard Scaler uses a strict definition of standardization to standardize data. It purely centers the data by using the following formula, where u is the mean and sis the standard deviation. `x_scaled` = (x — u) / s

6.1 Effect of Scaling the data for MLP

This section, best and average accuracies are calculated when MinMaxScaler is used to transform the input data and also when batch size is varied. Changes to the batch size and scaling the data can be done in the InputDeck.py.

```
# ---------------------------------------------------------------------------
# Data Scaling specific parameters (they have default values)
# ---------------------------------------------------------------------------
scale = False #True : if scaling is required, False  = if its not required
scale_name='StandardScaler' #type of scaling method, options: 'Min_Max' and '
    StandardScaler'
```

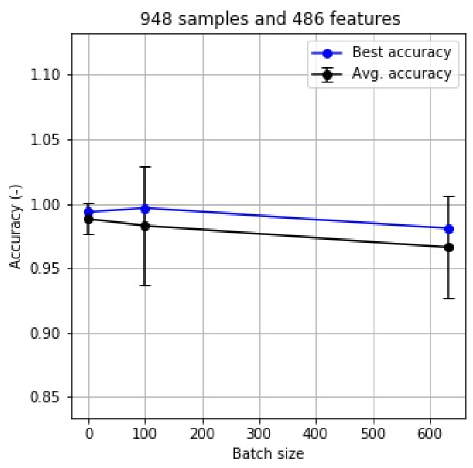


Figure 8: The average and best accuracy afer using the MAX-MIN scaler .

From the above graph it is visible that, the average accuracy decreases with the increase of batch size. The average accuracies for stochastic, Min-batch and batch gradient descent are

0.9883 +/- 0.012, 0.983122 +/- 0.046 and 0.966244 +/- 0.0395, but there is no improvement in the average accuracy when MinMax scaler is used.

In the next part of the experiment, best and average accuracies are calculated when StandardScaler is used to transform the input data and also when batch size is varied.

From the below graph, the average accuracy decreases with the increase in batch size. There is no advantage using stand scalar on the input data.

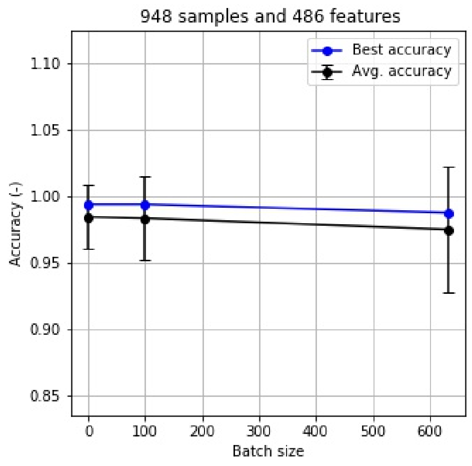


Figure 9: The average and best accuracy afer using the Standard Scaler scaler .

The best average accuracies are tabulated below.

Table 4: Comparison of Average Accuracies.

Serial.No.	Scaling Used	Learning Used	Average Accuracy
1	MinMax Scalar	Mini-Batch Gradient Descent	0.983122 +/- 0.046
2	Standard Scalar	Mini-Batch Gradient Descent	0.983122+/- 0.031

6.2 Effect of Scaling the data for Novel Learner

In this section, MinMax scalar and StandarScalar are used to scale the data and, the average and best accuracies obtained after scaling the data is compared as a function of batch size. Sklearn Preprocessing library is used to scale the data. Changes to the batch size and scaling the data can be done in the InputDeck.py.

```
# -----------------------------------------------------------------------------
# Data Scaling specific parameters (they have default values)
# -----------------------------------------------------------------------------
scale = False #True : if scaling is required, False  = if its not required
scale_name='StandardScaler' #type of scaling method, options: 'Min_Max' and '
    StandardScaler'
```

It is evident from the graph that the accuracy increases when MinMax scaler is used to scale the data. But the increase is not statistically significant. The accuracy doesn't show great improvement when Standard Scaler is used to scale the data. The reason why MinMax scaler works better is because the fact that the standard deviation is very small. The best average accuraies are tabulated below.

Table 5: Best Accuracies using Scaling.

Serial.No.	Scaling Used	Learning Used	Average Accuracy
1	MinMax Scalar	Mini-Batch Gradient Descent	0.9968 +/- 0.00786
2	Standard Scalar	Mini-Batch Gradient Descent	0.9968+/- 0.0240

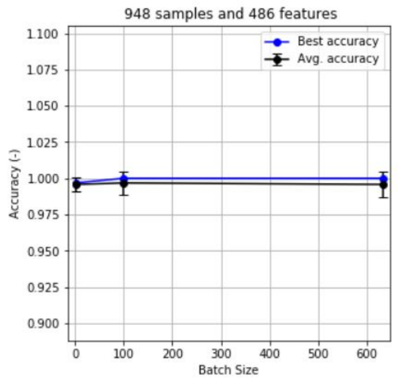


Figure 10: The average and best accuracy afer using the MAX-MIN scaler .

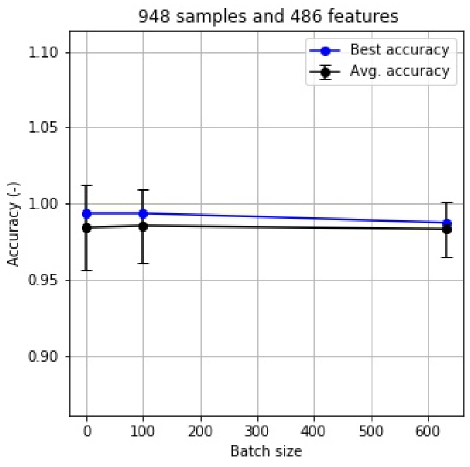


Figure 11: The average and best accuracy afer using the Standard Scaler scaler .

In conclusion, by using MinMax scalar to scale the data, the average accuracy of the model can be increased.

```
To conclude, from the above results, I am using Min-Max scalar to scale the data,
since better accuracies are obtained than standard scalar .
```

7 Hyper-parameter tuning of MLP

7.1 Results

7.1.1 Changing the number of features/samples

In this experiment, best and average accuracies are obtained by altering the number of samples and features in the data. Stratified sampling is utilized to create the cross-validation folds for every experiment in this section.The number of features and samples can be changed by using the data ratio parameter in the InputDeck.py

```
# ---------------------------------------------------------------------------
# General parameters, must be specified!
# ---------------------------------------------------------------------------
read_csv = False # False: prepare db, True: read existing csv
confidence = 0.95 # Confidence interval for the t values
rng_seed = 0 # Starting seed for the random number generator
data_ratio = 0.7 # Ratio of the features to keep in the reduced fature data base
to_remove = "features" #"samples" # Which one to decrease ("features"/"samples"/None)
```

The first experiment is conducted by changing the number of features keeping the number of samples constant. The results are then plotted on to a graph for better understanding. The graph is displayed below

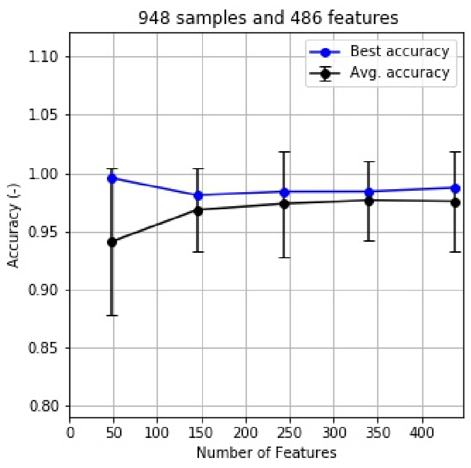


Figure 12: The average and best accuracy as function of the number of Features in the data base.

For the data set containing 948 samples and 486 features, it is visible that the mean accuracy of the model increases with the increase in the number of features used to fit the model, however it is not statistically significant. This decrease in accuracy is due to the high number of correlated features present in the data set.

In this experiment, the number of samples used to fit the MLP is altered. The obtained graph is displayed below

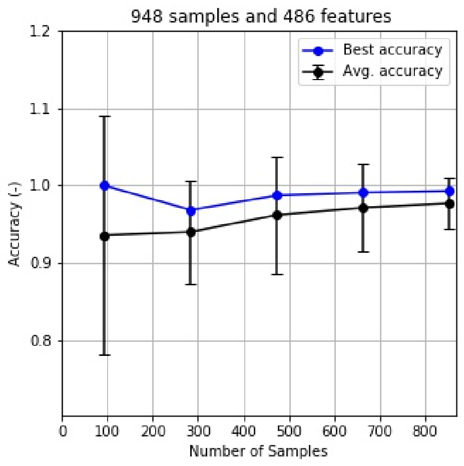


Figure 13: The average and best accuracy as function of the number of samples in the data base.

For the same data set, it is clearly visible that the average and best accuracy increases with the increase in number of samples used to fit the model. But the increase is not statistically significant

7.1.2 Changing the number of Layers

In this section, batch, mini-batch and stochastic learning of the MLP model is done when the number of layers in MLP are changed. The best and average accuracy obtained from this experiment are compared. Changes to the batch size and number of layers can be done in the InputDeck.py.

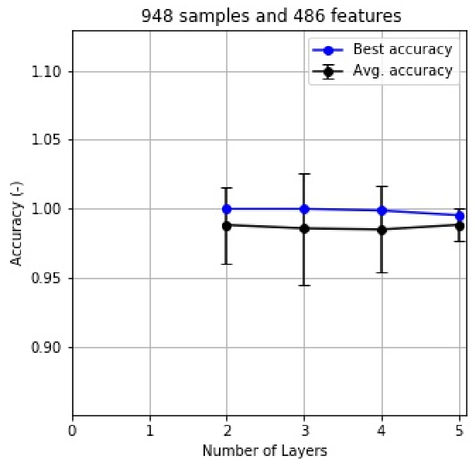
```
# ---------------------------------------------------------------------------
# MLP and Novel Learner specific parameters (they have default values)
# ---------------------------------------------------------------------------
solver = 'sgd' #which solver to use options -{   lbfgs   ,    sgd   ,    adam   },
    default = 'sgd'
batch_size =1 #Size of the batch
hidden_layer_sizes=(30,30,30) #number of hidden layers
max_iter  = 10000 #Maximum number of iterations. default = 200
learning_rate = 'adaptive' #{   constant    ,    invscaling    ,    adaptive   },
    default=   constant
learning_rate_init = 0.001 #The initial learning rate used. default=0.001
```

It is clearly visible that the average and best accuracy increases with the increase in the number of layers for Batch, mini batch and stochastic gradient descent. But accuracy for both mini batch and stochastic learning decreases for five layered MLP, but the decrease is not statistically significant.

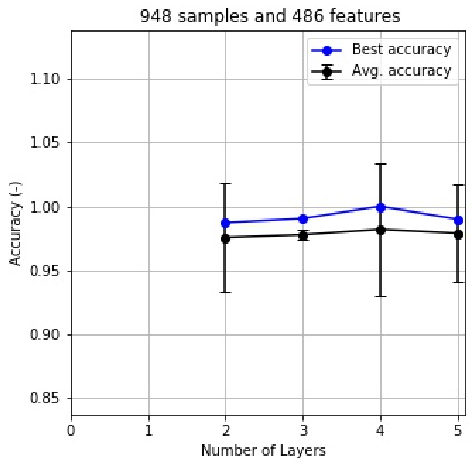
MLP model performs better with four layers, as the best and average accuracies obtained with four layers are the highest in the three experiments conducted. This tells us that the decision boundary for this data set is non-linear since introduction of hidden layers makes it possible for the network to exhibit non-linear behavior. The best average accuracies obtained using the three learning methods for the optimum number of layers are tabulated below.

Table 6: Comparison of Average Accuracies.

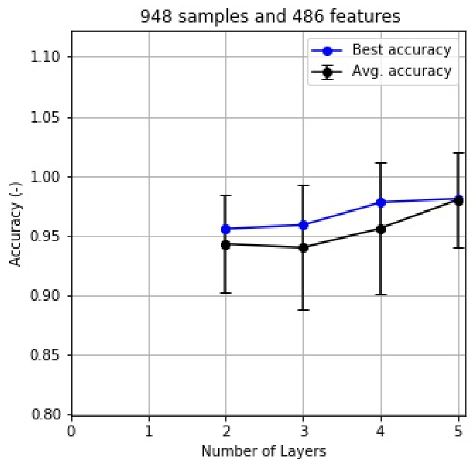
Serial.No.	Learning Used	Number of Layers	Average Accuracy
1	Stochastic Gradient Descent	4	0.98523 +/- 0.00403
2	Mini-Batch Gradient Descent	4	0.982 +/- 0.052
3	Batch Gradient Descent	5	0.966 +/- 0.03956



(a) Stochastic Gradient Descent



(b) Mini-Batch Gradient Descent



(c) Batch Gradient Descent

Figure 14: The average and best accuracy as function of the number of samples in the data base.

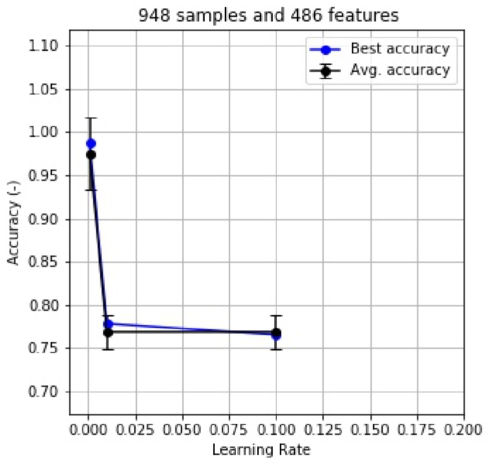
7.1.3 Changing the Learning Rate

In this section, batch, mini-batch and stochastic learning of the three layered MLP model is done when the learning rate is changed. The best and average accuracy obtained from this experiment are compared. Changes to the learning rate can be done in the InputDeck.py.

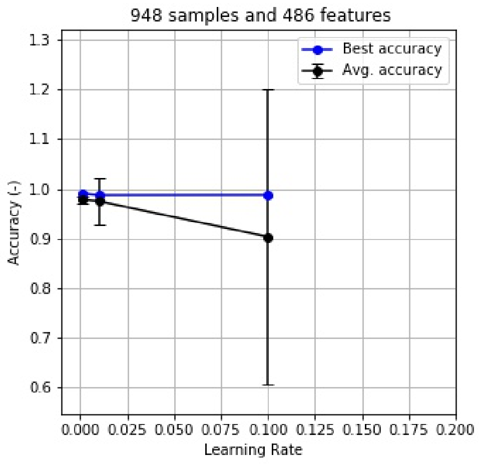
```
# ---------------------------------------------------------------------------
# MLP and Novel Learner specific parameters (they have default values)
# ----------------------------------------------------------------------
learning_rate = 'adaptive' #{    constant    ,     invscaling    ,     adaptive    },
    default=    constant
learning_rate_init = 0.001 #The initial learning rate used. default=0.001
```

The average and best accuracies are plotted in a graph for each case. The average accuracy of the MLP decreases in Stochastic gradient learning with the increase in learning rate. Because the oscillations are more and large in hypothesis space for Stochastic gradient descent, smaller learning rate ensures that the model converges to a global minimum.

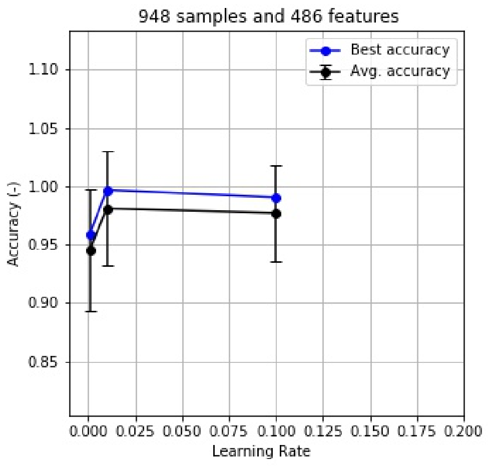
The average accuracies for batch and mini batch decreases for learning rate equal to 0.1. But the highest average accuracy is obtained at learning rate equal 0.01. This is justifiable because the oscillations for batch and mini batch are not as large as Stochastic gradient descent, hence a higher learning than in Stochastic gradient descent can be taken.



(a) Stochastic Gradient Descent



(b) Mini-Batch Gradient Descent



(c) Batch Gradient Descent

Figure 15: The average and best accuracy as function of the learning rate.

The average accuracies for the best learning rate for the three models are tabulated below

Table 7: Comparison of Average Accuracies.

Serial.No.	Learning Used	Best Learning Rate	Average Accuracy
1	Stochastic Gradient Descent	0.001	0.9873 +/- 0.0415
2	Mini-Batch Gradient Descent	0.01	0.974683 +/- 0.04781
3	Batch Gradient Descent	0.01	0.98101+/- 0.049

8 Hyper-parameter tuning of Novel Learner

8.1 Results

8.1.1 Changing the number of features/samples

Average and best accuracy of the model is measured by changing the number of features and samples in the data set. This can be done using the data ratio parameter in the InputDeck.py

```
# ---------------------------------------------------------------------------
# General parameters, must be specified!
# ---------------------------------------------------------------------------
confidence = 0.95 # Confidence interval for the t values
rng_seed = 0 # Starting seed for the random number generator
data_ratio = 0.7 # Ratio of the features to keep in the reduced fature data base
to_remove = "features" #"samples" # Which one to decrease ("features"/"samples"/None)
```

The obtained average and best accuracies are displayed in the graph below

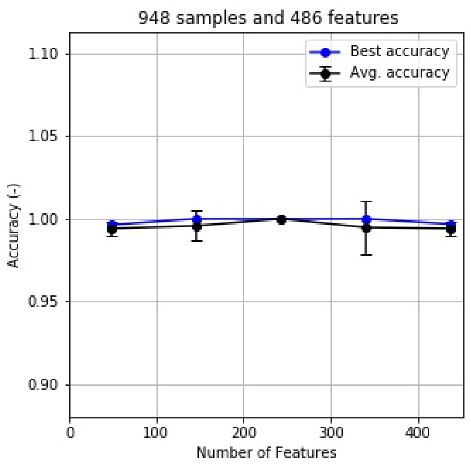


Figure 16: The average and best accuracy as function of the number of Features in the data base.

The average accuracy increases with the increase in features up to a point and then decreases, but the decrease is not statistically significant. This is because as the number of features increases, the number of correlated features in the data set increases due to which there is a decline in the average accuracy.

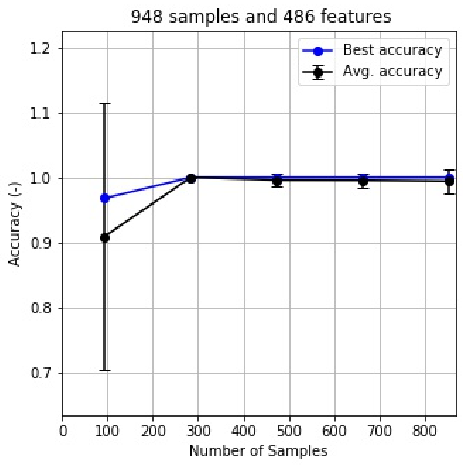


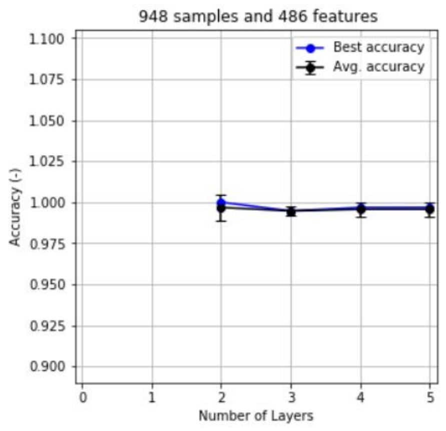
Figure 17: The average and best accuracy as function of the number of samples in the data base.

It is visible from the above graph that the average accuracy increases with the increase in the number of samples used to fit the model. But the increase is not statistically significant.

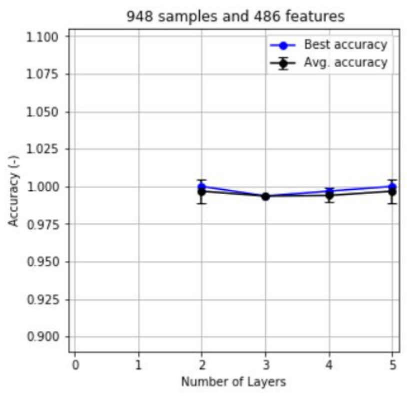
8.1.2 Changing the number of Layers

The average and best accuracies are measured but changing the number of layers in the novel learner during Stochastic, Mini batch and Batch gradient descent learning. Changes to the batch size and number of layers can be done in the InputDeck.py.

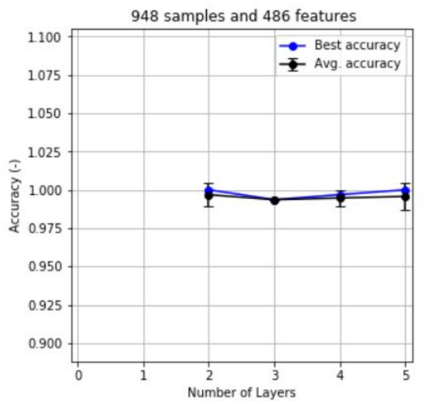
```
# ---------------------------------------------------------------------------
# MLP and Novel Learner specific parameters (they have default values)
# ---------------------------------------------------------------------------
solver = 'sgd' #which solver to use options -{    lbfgs    ,     sgd    ,     adam    },
    default = 'sgd'
batch_size =1 #Size of the batch
hidden_layer_sizes=(30,30,30) #number of hidden layers
max_iter  = 10000 #Maximum number of iterations. default = 200
learning_rate = 'adaptive' #{    constant    ,     invscaling    ,     adaptive    },
    default=    constant
learning_rate_init = 0.001 #The initial learning rate used. default=0.001
```



(a) Stochastic Gradient Descent



(b) Mini-Batch Gradient Descent



(c) Batch Gradient Descent

Figure 18: The average and best accuracy as function of the number of samples in the data base.

From the above graphs, it is clearly visible that the best and average accuracies increase with the increase in the number of layers in all the three learning algorithms. But a significant rise in the accuracy can be seen for five layered in batch and mini-batch gradient descent. The Novel learner performs better with 5 layered MLP. This because the decision boundary is not linear for the given data set. The best accuracies for the three learning algorithms are tabulated below.

Table 8: Comparison of Average Accuracies.

Serial.No.	Learning Used	Number of Layers	Average Accuracy
1	Stochastic Gradient Descent	5	0.99570 +/- 0.004536
2	Mini-Batch Gradient Descent	5	0.996 +/- 0.00078
3	Batch Gradient Descent	5	0.9957 +/- 0.0090

8.1.3 Changing the Learning Rate

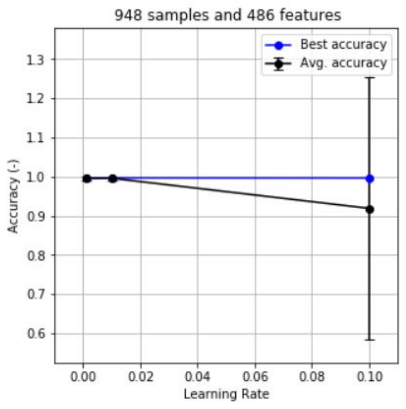
In this section, batch, mini-batch and stochastic learning of the model is done when learning rate is changed to get the optimum learning rate for the model. he best and average accuracy obtained from this experiment are compared. Changes to the batch size and number of layers can be done in the InputDeck.py.

```
# ----------------------------------------------------------------------------
# MLP and Novel Learner specific parameters (they have default values)
# ----------------------------------------------------------------------------
solver = 'sgd' #which solver to use options -{   lbfgs   ,    sgd   ,    adam   },
    default = 'sgd'
batch_size =1 #Size of the batch
hidden_layer_sizes=(30,30,30) #number of hidden layers
max_iter  = 10000 #Maximum number of iterations. default = 200
learning_rate = 'adaptive' #{   constant   ,    invscaling   ,    adaptive   },
    default=   constant
learning_rate_init = 0.001 #The initial learning rate used. default=0.001
```

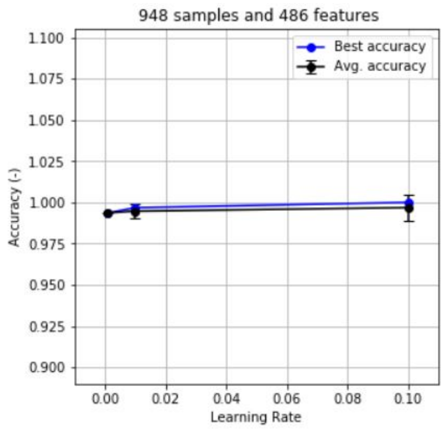
From the graph, it is clearly visible that the average accuracy decreases as the learning rate increases for stochastic descent learning, which is the same trend observed in MLP model. This observation makes sense because the oscillations in hypothesis space for stochastic gradient descent are large and taking a higher learning rate will make it difficult for the algorithm to converge to a global min. But the average accuracy increases as the learning rate increase for the remaining two learning methods since the oscillations in the respective hypothesis space are not as large as stochastic descent learning. Hence a higher learning can be considered.
The average accuracies for the best learning rate for the three models are tabulated below

Table 9: Comparison of Average Accuracies.

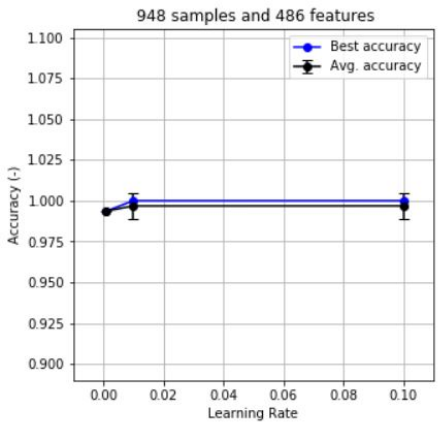
Serial.No.	Learning Used	Best Learning Rate	Average Accuracy
1	Stochastic Gradient Descent	0.001	0.9947 +/- 0.045
2	Mini-Batch Gradient Descent	0.01	0.9968+/- 0.0078
3	Batch Gradient Descent	0.1	0.9968+/- 0.0078



(a) Stochastic Gradient Descent



(b) Mini-Batch Gradient Descent



(c) Batch Gradient Descent

8.1.4 Changing the number of trees

In this section, number of trees used in the novel learner is changed and, best and average accuracies are calculated. Number of trees can be changed in the InputDeck.py.

```
# -----------------------------------------------------------------------------
# Novel learner specific parameters (they have default values)
# -----------------------------------------------------------------------------
bag_ratio = 0.3 # The number of samples used for each decision tree/total number of
    samples
n_trees = 150 # How many trees we want to build
```

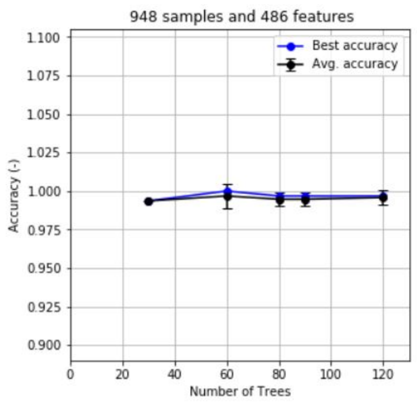


Figure 20: The average and best accuracy after changing the number of trees.

From the above graph it is clearly visible that the accuracy increases till number of trees equal to 60 and then decreases but the decrease is not statistically significant. With the increase in the number of trees, the model gets overfitted, hence the decrease in the average accuracy.

Table 10: Comparison of Average Accuracies.

Serial.No.	Number of trees	Average Accuracy
1	30	0.9936 +/- 0.0
2	60	0.9968+/- 0.00786
3	70	0.9947 +/- 0.004532
4	90	0.9947 +/- 0.004532
5	120	0.9968 +/- 0.00786

8.1.5 The Time complexity of the model

Let N_T denote the number of trees and b the ratio of the training samples used in each bag (decision tree). The **training time complexity** for each decision tree is somewhere between $O(b \cdot N \cdot D \cdot log_2(b \cdot N))$ and $O(b \cdot N \cdot D \cdot b \cdot N)$. If we use N_T trees, we get a complexity somewhere between $O(N_T \cdot b \cdot N \cdot D \cdot log_2(b \cdot N))$ and $O(N_T \cdot b \cdot N \cdot D \cdot b \cdot N)$. The training complexity of the MLP is ((N-1)/2)$^2 \in O(N^2)$.

So the Time complexity of the model is :

Training time complexity: between $O(N_T \cdot N \cdot [b \cdot D \cdot log_2(b \cdot N) + I])$ and ((N-1)/2)^2

9 Comparison of the Models

The average and best accuracy of the Novel learner is compared with KNN, Decision Tree, Perceptron, Ensemble and MLP. The Ensemble Leraner used in this project consists of N

CART decision trees and a perceptron. The perceptron, KNN, Decision Trees are developed using Sklearn package in python. The decision tree used in this project uses CART algorithm. Stratified three fold cross validation is used find the accuracies of the models.

The average and the best accuracies of different models are tabulated below.

Table 11: Comparison of Different Learners.

Serial.No.	Learner Name	Best Accuracy	Average Accuracy	Training Time(sec)	Testing time(sec)
1	Decision Tree	0.9957	0.9937 +/- 0.01210	0.01296	0.0311
2	Perceptron	0.98734	0.957+/- 0.0432	0.0588	0.1276
3	Ensemble learner	0.9968	0.9957 +/- 0.004532	0.33211	0.6921
4	MLP	0.99650	0.97890 +/- 0.00432	1.8673	6.1367
5	Novel Learner	1.00	0.9968 +/- 0.007861	1.56694	3.6651
6	KNN	0.9934	0.9926 +/- 0.0164	0.000	0.01988

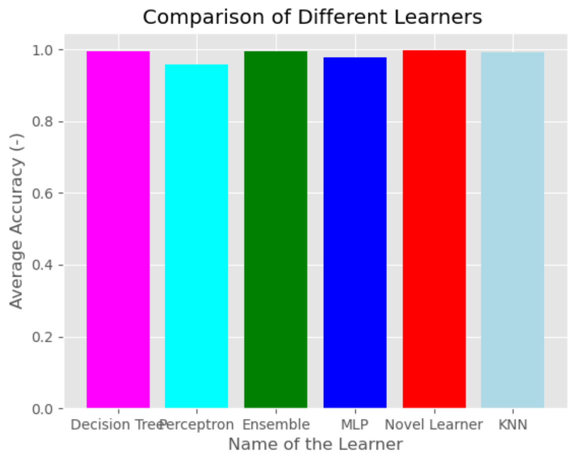


Figure 21: Comparison of Different Learners.

It is clearly visible from above table that the average accuracy of Novel learner is greater than the average accuracy of other learners. This average accuracy of Novel learner increases by
5% in feasible time and space complexity, after hyper-parameter tuning. The best values of the hyper parameters are tabulated below.

Table 12: Hyper-parameter tuning.

Serial.No.	Hyper-parameter	Best Value/Name
1	Batch Size	200
2	Number of Layers	5
3	Learning rate	0.01
4	Learning Algorithm	Mini-batch gradient descent
5	Scaling	Min-Max Scaler
6	Number of Trees	60

After performing the experiment with the number of layers, the optimum number of layers for the novel learner is found out to be five.

The optimum learning rate and gradient descent is found to be 0.01 and Mini-batch gradient descent learning. This is understandable because Mini-batch gradient descent learning has low oscillation in its hypothesis space than stochastic gradient descent and it is computationally faster than batch gradient descent. After changing the number of trees used in the novel learner, average accuracy is highest when sixty trees are used to fit the novel learner. Its also evident from above experiments that using Max Min scaler to scale the data increases the accuracy of the model.

10 Conclusion

To summarize, a Novel learning which is an ensemble learner consisting of N trees and an MLP learner is developed for attack detection. Hyper-parameter tuning of the model is done to find the optimum values for the hyper-parameters to improve the average accuracy of the model by 5%. Different learning methods are also implemented to optimize the cost function of the model. Scaling of the data is also done to increase the average accuracy.

This model is then compared with the other base models which were used in the previous assignment. Overall the average accuracy of the novel learner is greater than other base learners because of hyper-parameter tuning and scaling the input data.

11 References

[1] T. M. Mitchell, Machine Learning.

[2] https://scikit-learn.org/stable/modules/neural_networks_supervised.html

[3] https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484

[4] https://towardsdatascience.com/preprocessing-with-sklearn-a-complete-and-comprehensive-guide-670cb98fcfb9

[5] https://towardsdatascience.com/gradient-descent-show-me-the-math-7ba7d1caef09

[6] https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02