MACHINE LEARNING GROUP-40 ASSIGNMENT-3 REPORT

Neha Dalmia 19CS30055 **Hritaban Ghosh 19CS30053**

Task:

Dataset Link : https://www.kaggle.com/mathchi/diabetes-data-set

Context

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes.

Content

Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

- Pregnancies: Number of times pregnant
- Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- BloodPressure: Diastolic blood pressure (mm Hg)
- SkinThickness: Triceps skin fold thickness (mm)
- Insulin: 2-Hour serum insulin (mu U/ml)
- BMI: Body mass index (weight in kg/(height in m)^2)
- DiabetesPedigreeFunction: Diabetes pedigree function
- Age: Age (years)
- Outcome: Class variable (0 or 1)

Number of Instances: 768
Number of Attributes: 8 plus class
For Each Attribute: (all numeric-valued)

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)

Class Distribution: (class value 1 is interpreted as "tested positive for diabetes")

Data Analysis and Visualisation

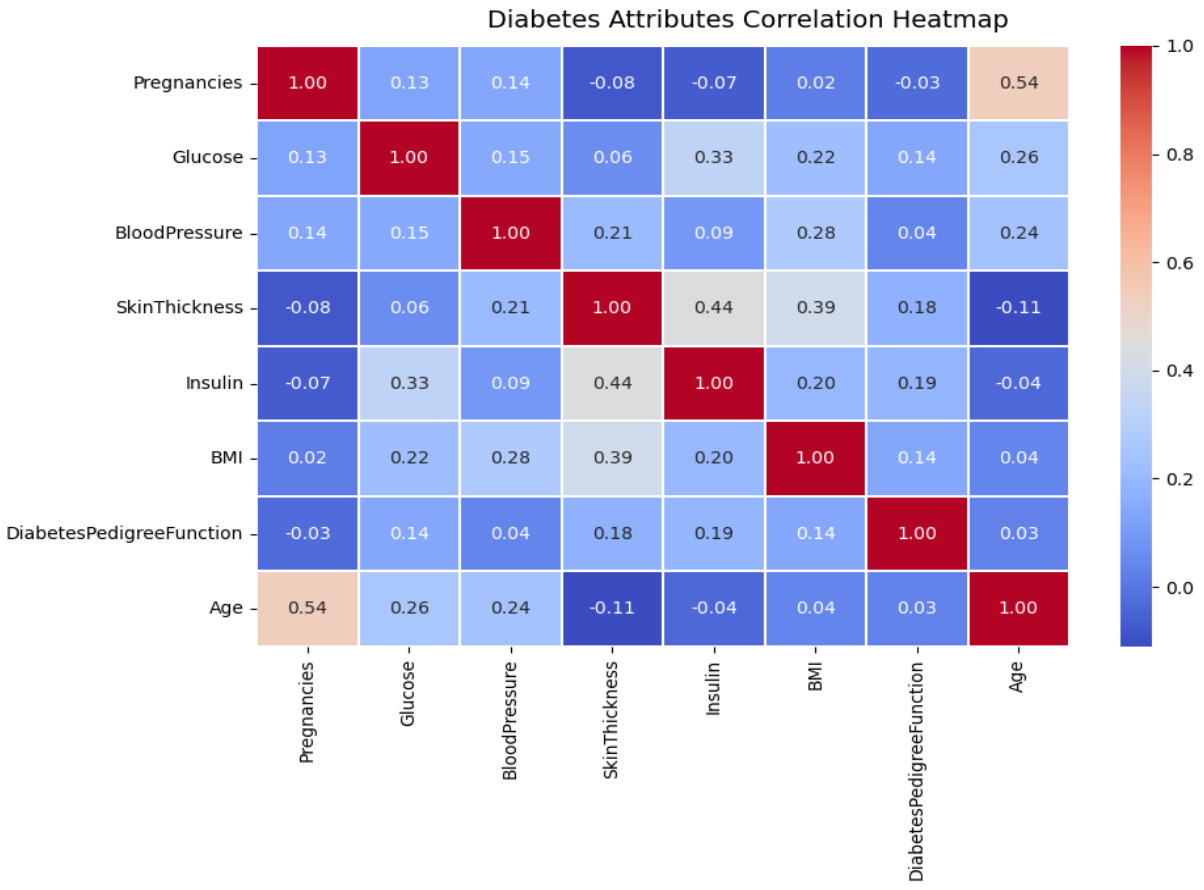
Preprocessing:

- We performed Standardisation on our attribute columns to ensure there is no dominant attribute.

We first printed the values which can be NULL or missing in any of the samples. We did not obtain any such values hence no preprocessing was required. There was also no attribute with the same value for all classes hence we did not need to remove any attribute either.

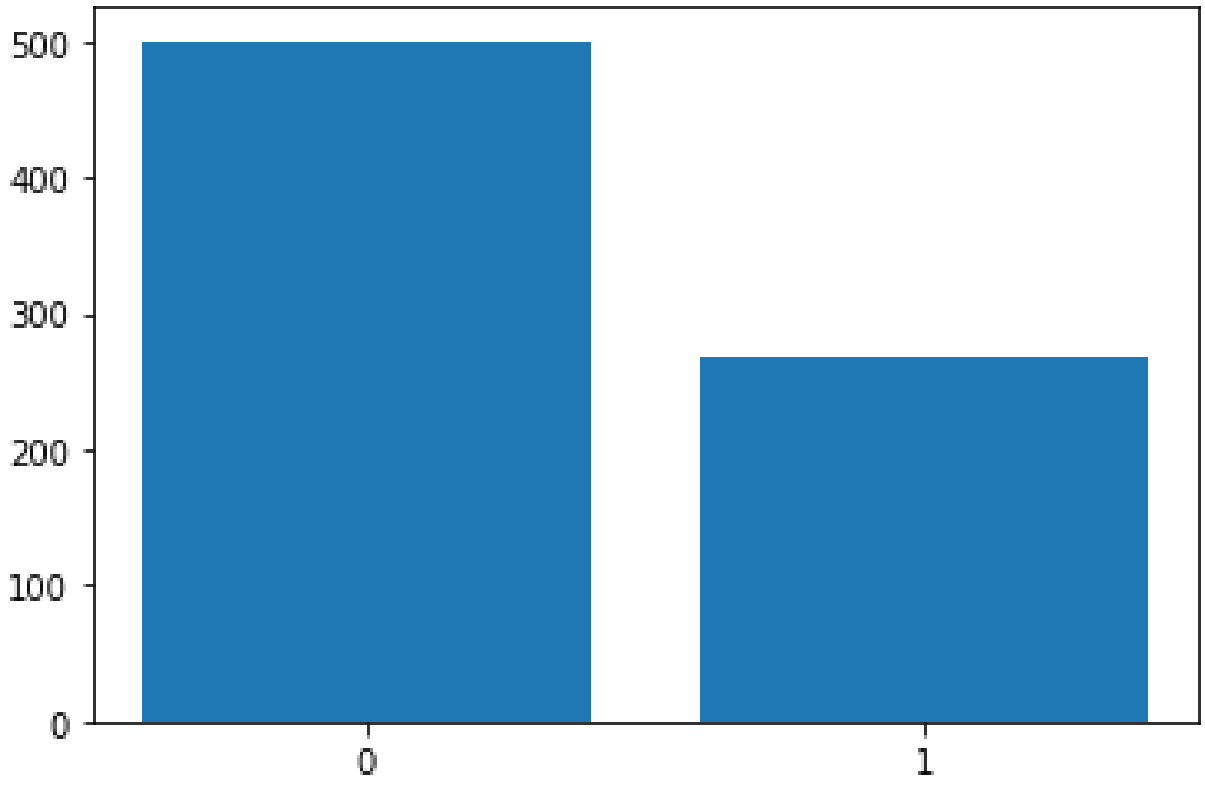
```
>>Checking for null values in the dataframe:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

The next graph that we make is the **heatmap**, this lets us know the correlation between any two attributes. This knowledge enables us to drop any columns that have high correlation with another column, so that the model doesn't have any redundancy in it.



As you can see from the heat map that there are no two distinct columns whose correlation is greater than 0.9, therefore we avoid any redundancies in our Model.

We then obtained the distribution of the class labels to ensure there is no bias in our dataset.



The number of 1s is approximately half of the number of 0s which indicates that our dataset is slightly biased towards 0. This would generally not be an issue but since our dataset is quite small (only 768) samples, we can get an idea that our model accuracy will probably not be very high.

Number of nodes in Input Layer

The number of nodes in Input Layer is 8 which corresponds to the 8 attributes of the data namely, Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, and Age.

Number of nodes in Output Layer

The number of nodes in the Output Layer is 1 which corresponds to the Class Variable (0 or 1), the Outcome.

Hyperparameters in the Multi-Layer Perceptron Classifier

We have defined the following parameters that can be changed by the user:
`n_epochs :` `number of epochs`
`learning_rate:` `the learning_rate of the MLP classifier`
`hidden layers and number of nodes:` `number of hidden layers and the nodes in them are also hyper parameters as we are dealing with neural networks`

Varying the number of hidden layers and number of nodes in each hidden layer to create the following models

Model 1 - No Hidden Layers

Model 2 - 1 Hidden Layer with 2 nodes

Model 3 - 1 Hidden Layer with 6 nodes

Model 4 - 2 Hidden Layer with 2 and 3 nodes respectively

Model 5 - 2 Hidden Layer with 3 and 2 nodes respectively

We then varied the learning rates for each of the above architectures as **0.1, 0.01, 0.001, 0.0001, and 0.00001.**

We then plotted the graphs of Accuracy for each model versus Learning rate and Accuracy for each learning rate versus Model.

Implementation Details

Number of epochs:

In order to maintain consistency in our analysis , we kept the number of epochs for all the different models and the learning rates to be the same. We applied it with values in the range of 500 to 1000 as these numbers are not low enough to prevent low learning rates from performing entirely and not very high as that would be costly with respect to time and lead to overfitting.

After training with epochs in this range, we even printed the loss obtained at every iteration, and we observed that after the first few hundred steps, the **loss function did not fall** significantly, only in the order of 1e-3, which is insignificant. This shows that training it for more epochs would not have helped as the model converged around the 500-1000 step mark for all learning rates.

Hence, in our analysis we used **500** epochs to train the model.

Using validation set:

We divided our dataset into training, validation and testing sets. During training, we calculate the **validation loss** at each iteration and using features of pytorch to save models, choose the model which gave the least validation less as our chosen model given a set of hyper parameters.

Loss Function:

We use **BCEWithLogitsLoss** as the loss function for our **back propagation** as we have a binary classification problem and this loss combines a Sigmoid layer and the BCELoss in one single class. This version is more numerically stable than using a plain Sigmoid followed by a BCELoss as, by combining the operations into one layer, we take advantage of the log-sum-exp trick for numerical stability.

For forward propagation we use Rectified Linear Activation Function for the forward propagation. The rectified linear activation function is a simple calculation that returns the value provided as input directly, or the value 0.0 if the input is 0.0 or less.

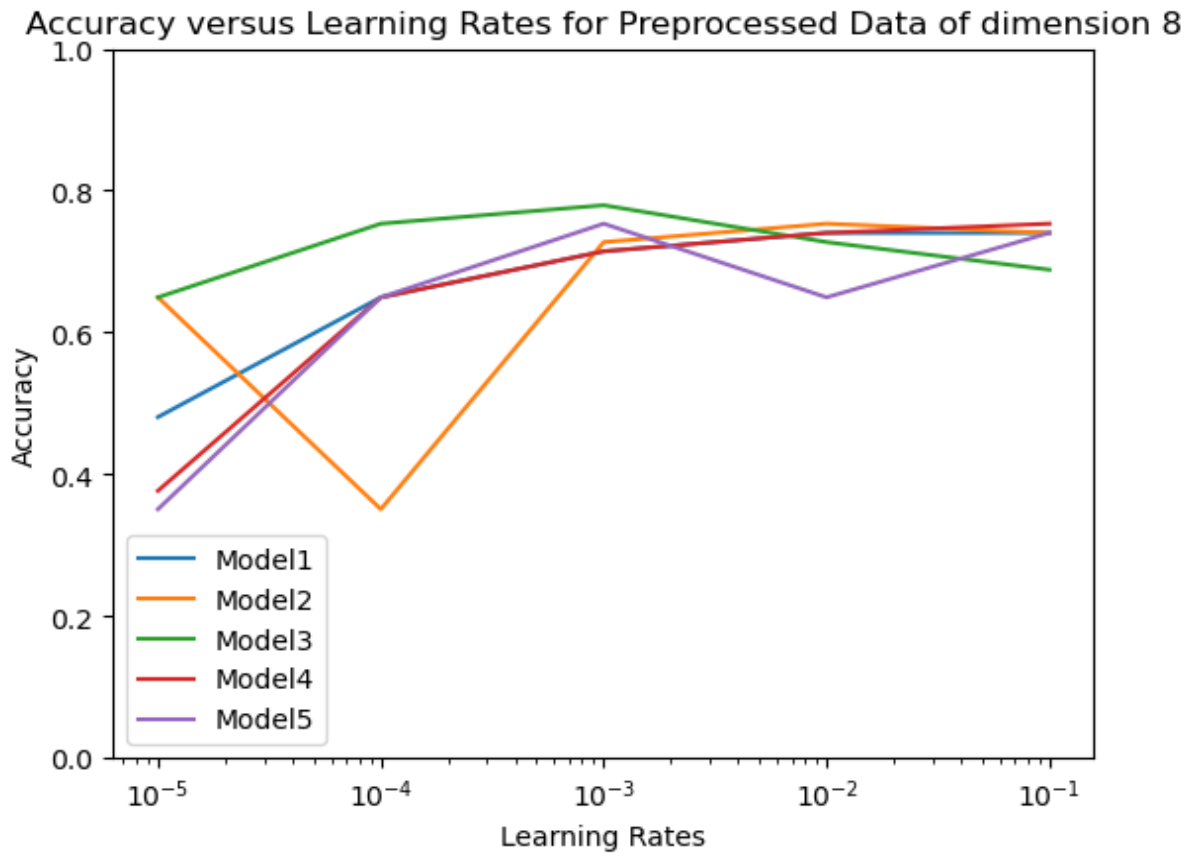
Dropout Layers:

We have added a dropout layer to our models before the output layer in order to prevent overfitting.

Sigmoid function and rounding for prediction:

Our model returns the weight present in the output layer. We need to apply the sigmoid function to this to convert this to a probability of the class being 1. After this we round off this value to obtain the exact class 0/1 as values with portability of being class 1 less than 0.5 will be assigned to 0 this way.

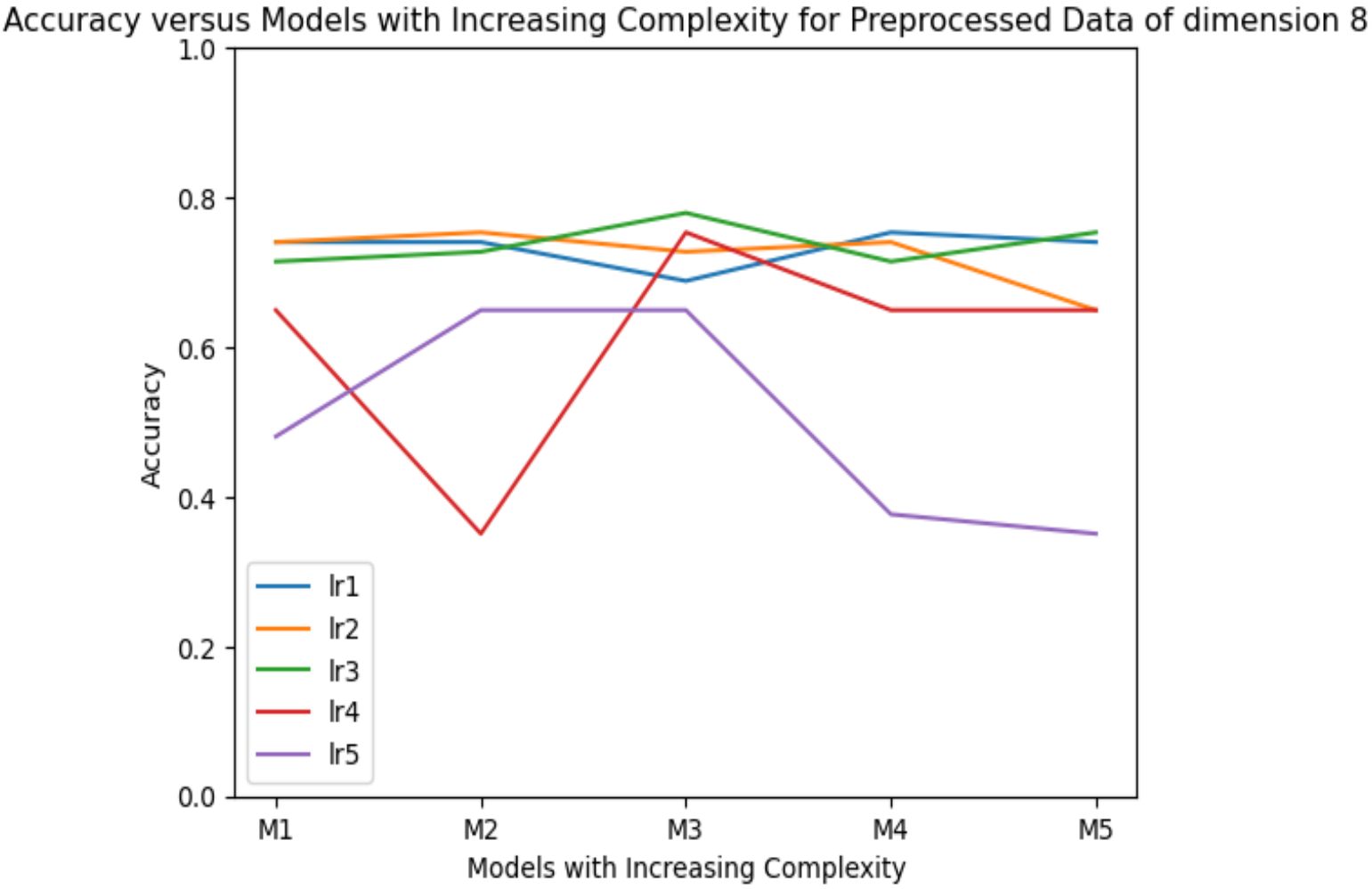
Plot of Accuracy of Each Model versus Learning Rate



Conclusion:

We see that **Model 3** with one hidden layer of 6 nodes gives the most accuracy(>80%) with a learning rate of 0.001 which is neither too high nor too low. We also see that Model 2 with one hidden layer of 2 nodes gives the least accuracy at the learning rate of 0.0001, but the accuracy increases as the learning rate increases. When the learning rate is at the highest of 0.1, all the models have an accuracy in the range of 65% to 80%. We see that Model 4 has an increase in accuracy with an increase in learning rate but still falls short of Model 3's highest accuracy. We can analyse the ideal model complexity will involve a large number of nodes as model 3 has the highest number of nodes in its hidden layers. Hence, higher complexity in terms of the number of nodes should yield good results**. As our dataset is very small, a deeper model will not necessarily yield good results. This is proven from the fact that model 3 with only one hidden layer out performs those with two hidden layers.**

Plot of Accuracy of Each Model versus Model

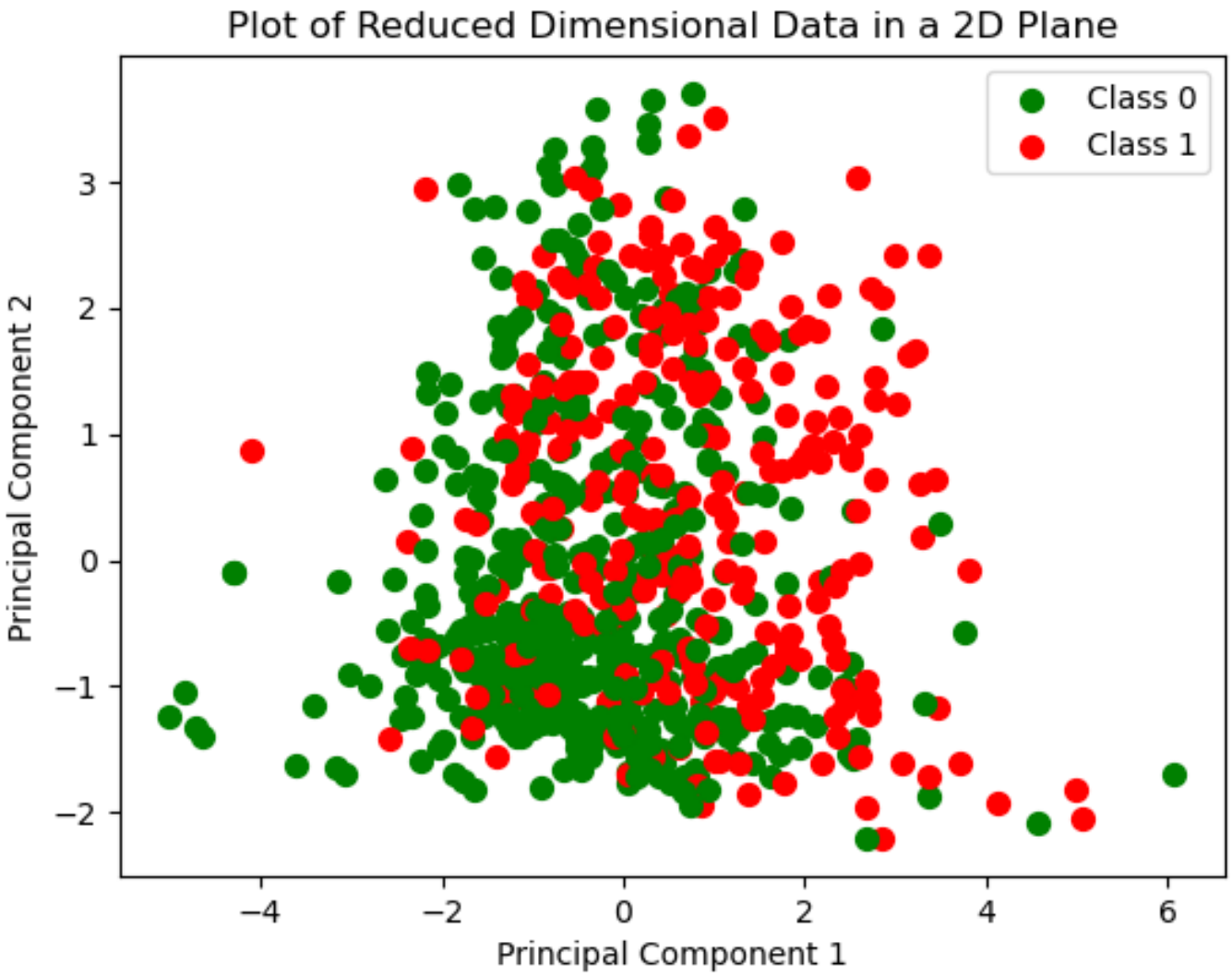


Conclusion:

We see that Model 3 with one hidden layer of 6 nodes gives the most accuracy(>80%) with a learning rate of lr3 = **0.001**. From the graph it is evident that learning rate 0.001 is the best for as across all models, it lies more or less above the other lines. Even the learning rate of 0.01 shows good performance. **These results are accurate because when the learning rate is too low, the model will converge very slowly, hence as the iterations are kept the same (for consistency) across all learning rates, the lower learning rates will fail to ensure convergence. Learning rates which are too high like 0.01 are also not efficient as the model might converge to a local minima. This is also evident from our diagram as we have obtained the best accuracy at the learning rate of 0.001.**

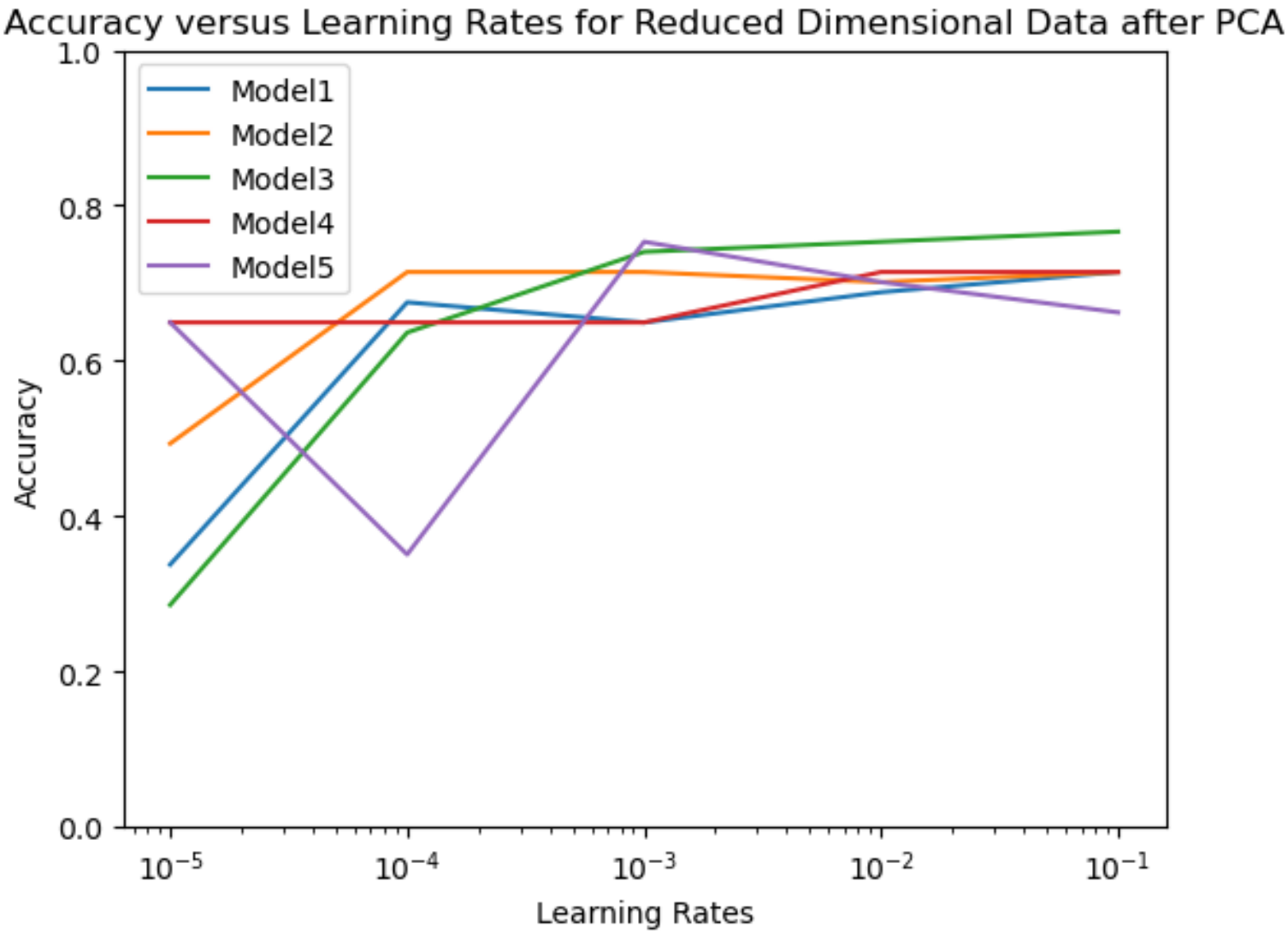
Principal Component Analysis

Now we perform PCA on the dataset of dimension 8 and reduce it to a dataset of 2 dimensions. We plot the Principal Component 1 on the x-axis and the Principal Component 2 on the y-axis. We then plot all the data samples on the 2-Dimensional Graph and color the data samples with different class labels differently. If Outcome is 0 then the color is green (Diabetes Free) and if the Outcome is 1 then the color is red (Diagnosed with Diabetes).



We see that the data is not clustered well and does not have a clear decision boundary when reduced to 2 dimensions.
Now, the number of nodes in the input layer will be **2** and the number of nodes in the output layer will remain at 2.

Plot of Accuracy of Each Model versus Learning Rate after PCA

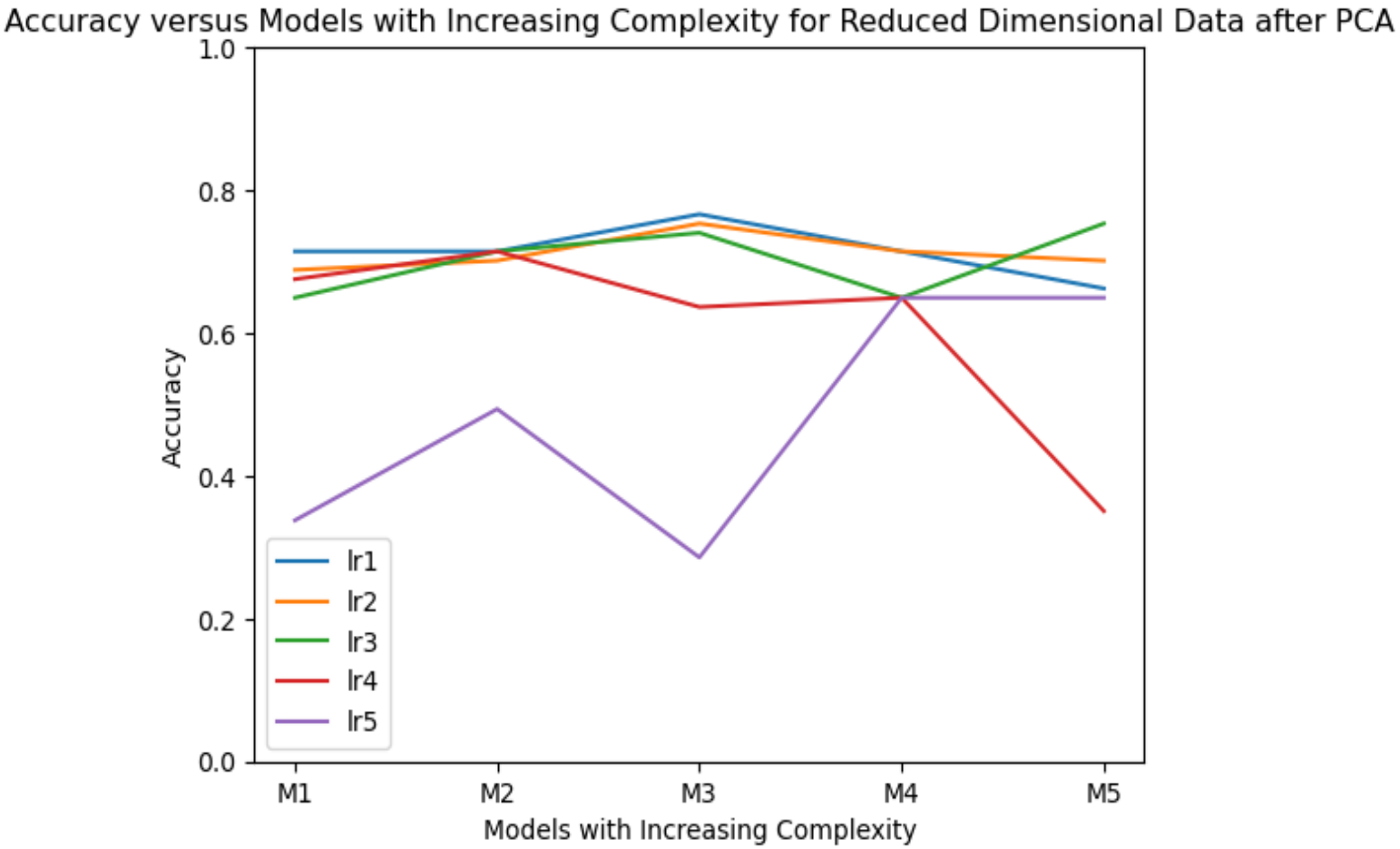


Conclusion:

We see that Model 3 with one hidden layer of 6 nodes gives the most accuracy(>80%) with a learning rate of 0.1, but the accuracy decreases as the learning rate decreases. We also see that Model 3 with one hidden layer of 6 nodes gives the least accuracy at the learning rate of 0.00001. When the learning rate is at the highest of 0.1, all the Models have an accuracy in the range of 60% to 80%.

There is not much difference after applying PCA except the models **follow trends more similarly. This is because reducing the dimensions reduces the complexity of the data significantly, which reduces the dependence on the model complexity.**

Plot of Accuracy of Each Model versus Model after PCA



Conclusion:

We see that Model 3 with one hidden layer of 6 nodes gives the most accuracy(>80%) with a learning rate of lr1 = 0.1, but the accuracy decreases as the complexity of the model increases. We also see that Model 3 with one hidden layer of 6 nodes gives the least accuracy at the learning rate of lr5 = 0.0001. For the learning rates of lr1 = 0.1, lr2 = 0.01, and lr3 = 0.001 all the models have accuracy in the range of 60% to 80%. Again, we can see that very low learning rates are not good for our model as they do not converge to a solution fast enough. **Since we have applied PCA, we can see a more clear distinction between the lower and higher learning rates since the dataset dimensions have been reduced lowering the dependence on the model complexity. This makes differences arising due to variations in learning rate more prominent.**

Best Learning Rate Before PCA

The best learning rate before performing Principal Component Analysis is 0.001 because it gives a good performance for all the models and Model 3 achieves highest accuracy using this learning rate.

Using the best learning rate to analyse the Models after PCA

Even **after performing Principal Component Analysis, all Models seem to fare well using the learning rate of 0.001.**

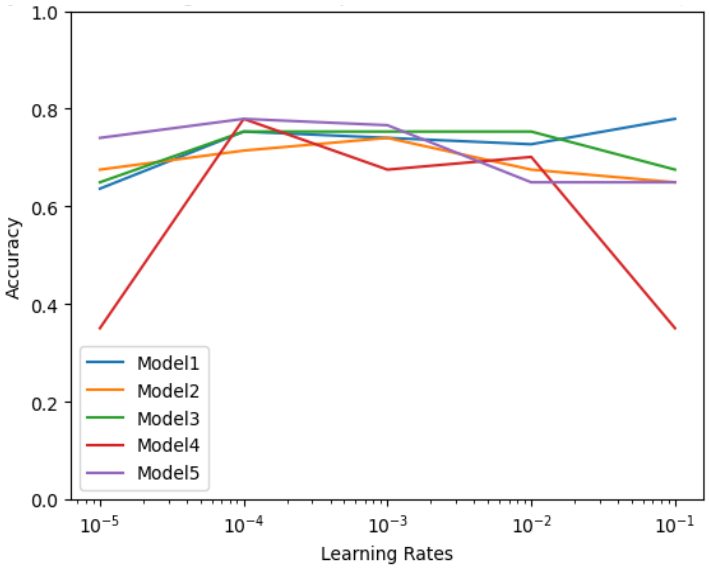
One key point to note is that all models also perform well for learning rates of 0.1 and 0.01 after PCA, while before PCA these learning did not have the best performance.

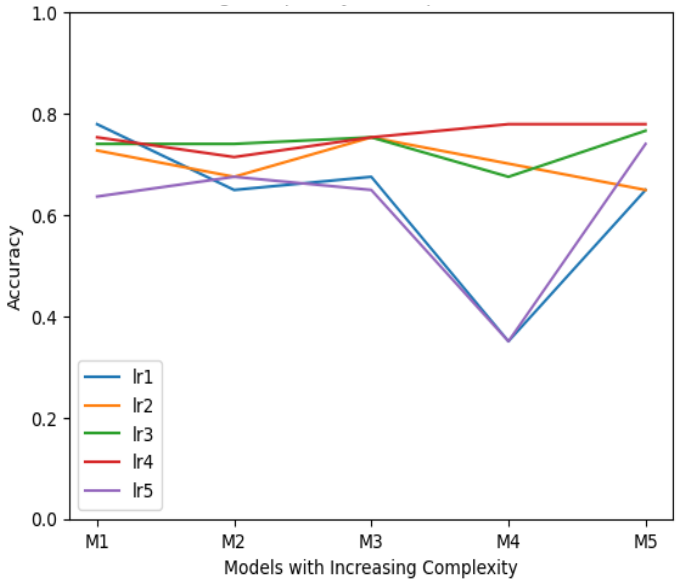
Considering all the results, our best accuracy is around 80% with the best model being Model 3 and best learning rate being 0.001 followed by 0.01.
PCA/Batch Learning/Hyper-parameter tuning did not change the accuracy significantly, which is understandable as our dataset size was too small to be trained on a neural network and it also had less number of attributes. The lack of change should be attributed to the small dataset, along with the bias in the number of samples mapped to class 0.

Extra Analysis: **Batch Learning**

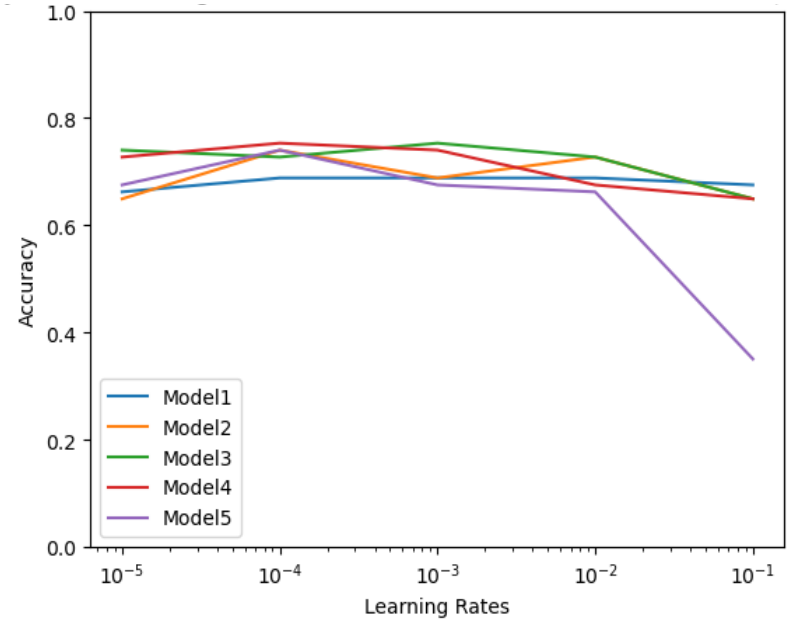
In this section, we train the model by supplying small batches of data of user defined size, we provide the analysis below for the batch size of 50 samples. We do this over several epochs, by shuffling the data and thus producing random mini-batches in each epoch.

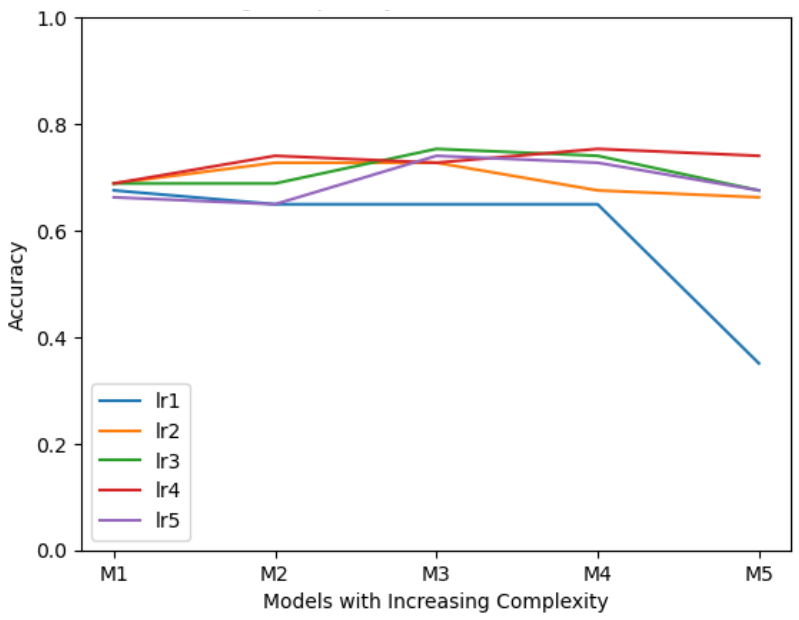
Before PCA





After PCA





Conclusion:

We observe that Batch Learning improves the overall performance of all models specially after Principal Component Analysis is performed on the dataset. Almost all models have accuracy between 60% to 80% for all learning rates except Model 5 (highest Complexity) which gives poor performance for the learning rate of 0.1 after PCA but good performance before PCA.