Group 7
Sagnik Roy (18CS10063)
Suryansh Kumar (18CS30043)

Machine Learning Assignment 3

15th November 2020

OVERVIEW

Given the QSAR+biodegradation dataset, the task was to perform SVM and MLP classification on the dataset as per the models given and compare the results of the 2 classifiers.

GOALS

1. Split the data into 80:20 Train_Test splits
2. With the data-split obtained in the previous step, perform SVM Classification and report its findings
3. Split the data 80:20 Train_Test splits and perform MLP classification on the data using the architectures given
4. Compare the performance of the 2 classifiers

PREREQUISITES

Python Inbuilt Libraries:

The python inbuilt libraries used obtaining, modifying and visualizing data are :

- Pandas
- Numpy
- Matplotlib
- Seaborn
- Sklearn

Procedure

Part 1: (SVM Classifier)

Pre-Processing the Data:

The dataset initially had a single column with semicolon separated entries, and no header. It was reframed into multiple columns and a header was added using pandas. No further preprocessing was required.

SVM kernels used:

We have implemented the binary SVM classifiers using the following kernels:

	Kernels	Most Suitable C value obtained
1.	Linear	4.0
2.	Quadratic	2.0
3.	Radial basis function	5.0

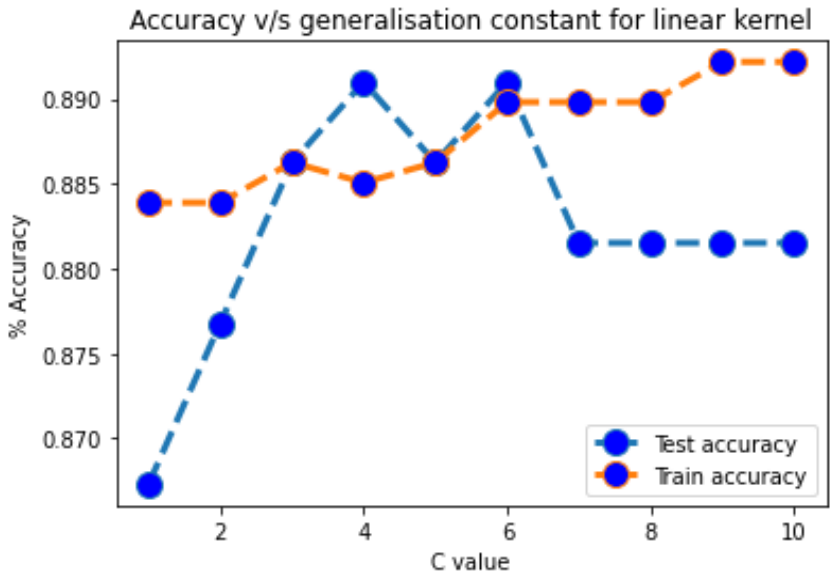
Findings:

Kernel	Training Set Accuracy	Test Set Accuracy
Linear	88.507%	89.099%
Quadratic	83.649%	86.256%
Radial Basis Function	85.308%	85.782%

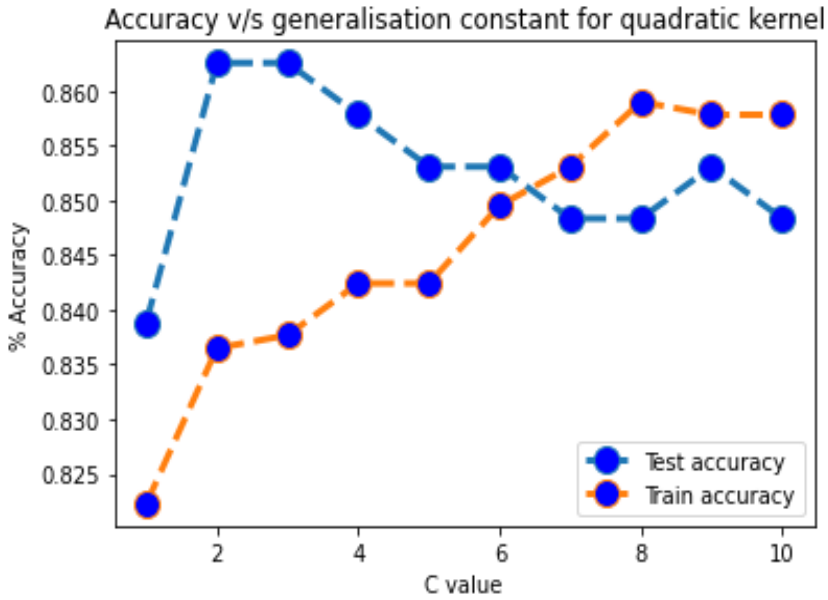
The accuracies in the above table are calculated with respect to the most suitable C value. For each kernel, we varied the C values from 1 to 10, with a step size of 1 and chose the value which provided best test accuracy.

We observe the following variation in training and test set accuracies with increase in C.

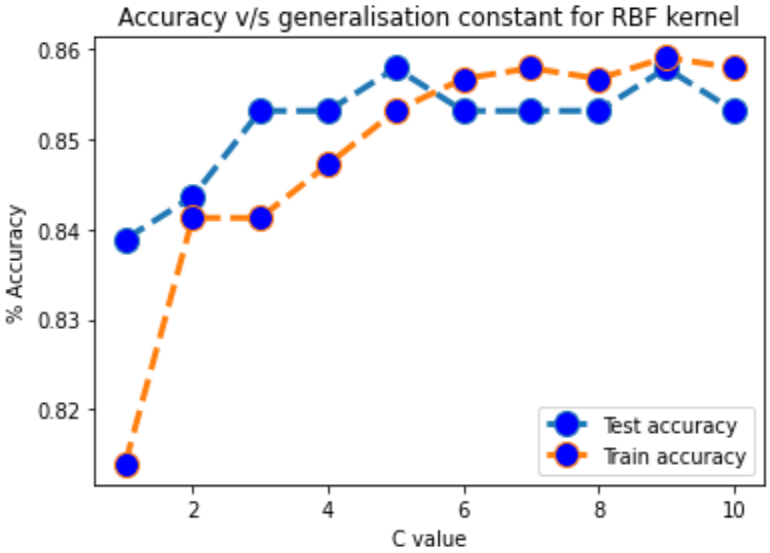
1. For linear kernel:



2. For Quadratic Kernel:



3. For rbf kernel:



Higher value of the generalisation constant, C aims at classifying all training examples correctly, whereas a low C makes the decision surface smooth and allows misclassification of some training examples. Choosing a suitable C value highlights the classic bias/variance tradeoff in machine learning. The accuracies tend to somewhat decrease with higher C values because then the model tends to overfit.

Among the three kernels, the highest test set accuracy **(89.099%)** is observed for the **linear** kernel. This suggests that our dataset might be linearly separable.

Part2: (MLP Classifier)

Pre-Processing the Data:

Pre-processing simply involved using the LabelEncoder to transform the target attribute for classification, which was given in categorical form (ready and not ready biodegradable) into numerical data (0 and 1). The dataset description mentions that all other data are numerical data and there are no missing values, hence no further preprocessing was required for the given data.

Models Used for MLP Classifier:

The models used for building the MLP Classifier, as mentioned in the assignment problem statement was;

4. 0 hidden layer
5. 1 hidden layer with 2 nodes
6. 1 hidden layer with 6 nodes
7. 2 hidden layers with 2 and 3 nodes respectively
8. 2 hidden layers with 3 and 2 nodes respectively

For each model 5 learning rates were used:

1. 0.1
2. 0.01
3. 0.001
4. 0.0001
5. 0.00001

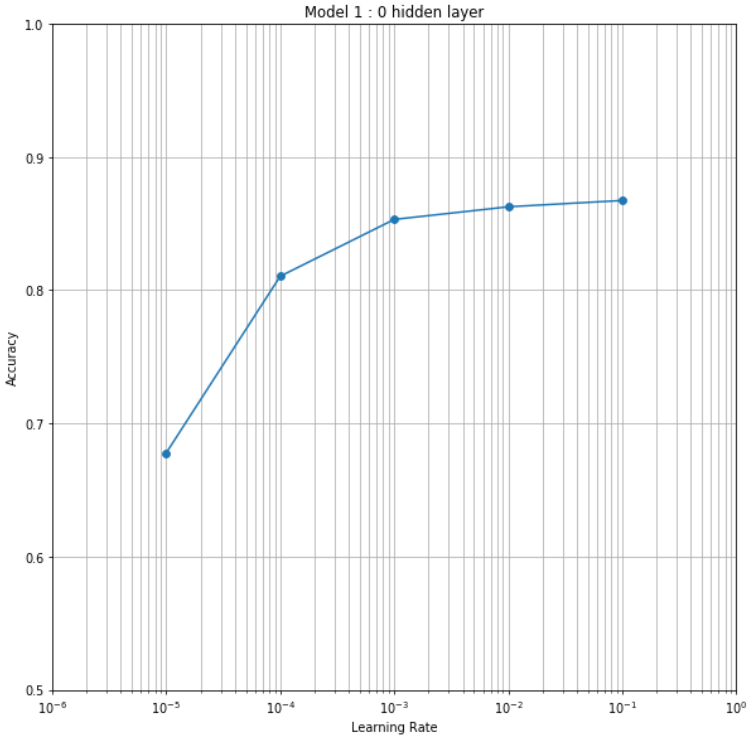
Findings:

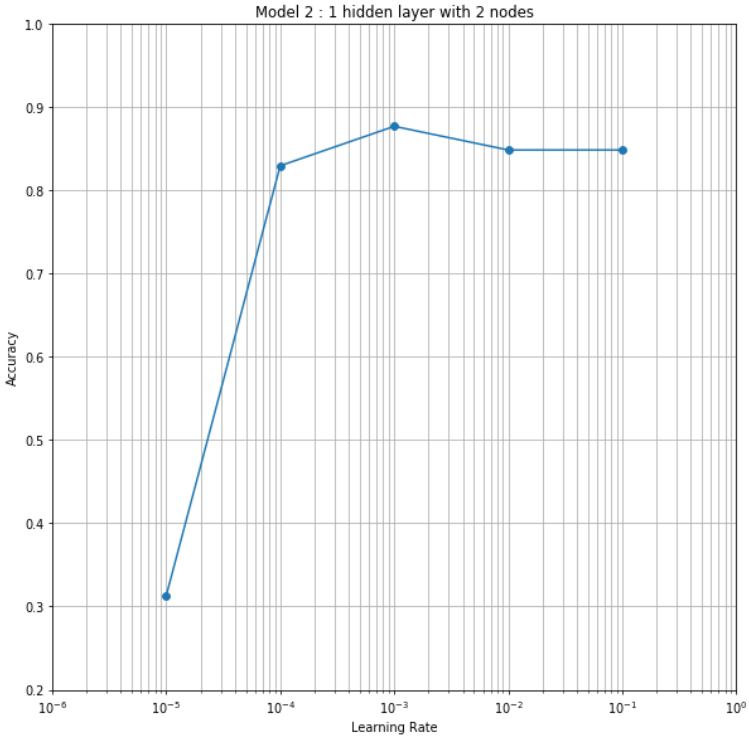
The accuracy reported for the the different models and the learning rates are:

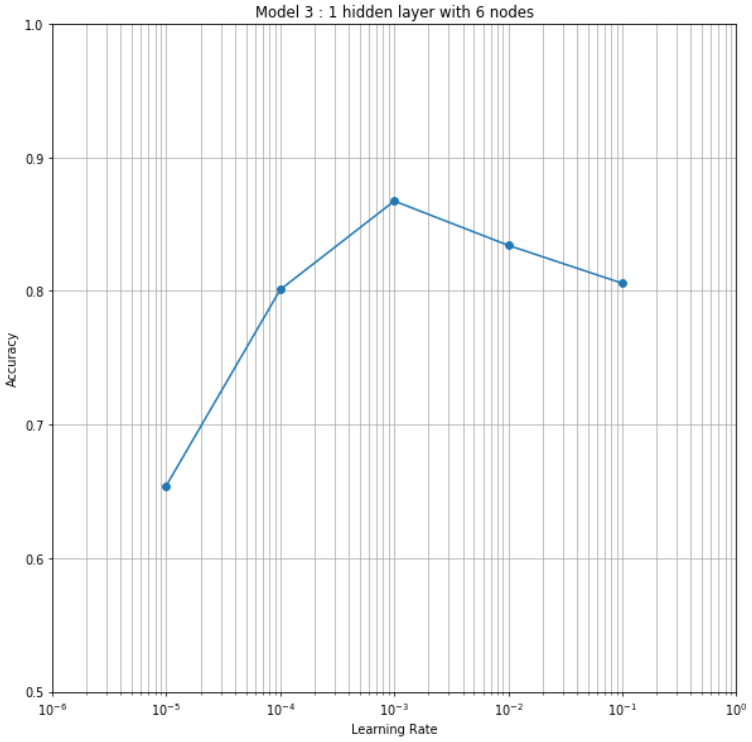
1. Model 1: 0 hidden Layer
    a. Learning Rate = 0.1: 0.8672985781990521
    b. Learning Rate = 0.01: 0.8625592417061612
    c. Learning Rate = 0.001: 0.8530805687203792
    d. Learning Rate = 0.0001: 0.8104265402843602
    e. Learning Rate = 0.00001: 0.6777251184834123
2. Model 2: 1 hidden layer with 2 nodes
    a. Learning Rate = 0.1: 0.8483412322274881
    b. Learning Rate = 0.01: 0.8483412322274881
    c. Learning Rate = 0.001: 0.8767772511848341
    d. Learning Rate = 0.0001: 0.8293838862559242
    e. Learning Rate = 0.00001: 0.3127962085308057
3. Model 3: 1 hidden layer with 6 nodes
    a. Learning Rate = 0.1: 0.8056872037914692
    b. Learning Rate = 0.01: 0.8341232227488151
    c. Learning Rate = 0.001: 0.8672985781990521
    d. Learning Rate = 0.0001: 0.8009478672985783
    e. Learning Rate = 0.00001: 0.6540284360189573
4. Model 4: 2 hidden layers with 2 and 3 nodes
    a. Learning Rate = 0.1: 0.8625592417061612
    b. Learning Rate = 0.01: 0.8483412322274881
    c. Learning Rate = 0.100: 0.8436018957345972
    d. Learning Rate = 0.0001: 0.6919431279620853
    e. Learning Rate = 0.00001: 0.33649289099526064
5. Model 5: 2 hidden layers with 3 and 2 nodes nodes
    a. Learning Rate = 0.1: 0.8199052132701422
    b. Learning Rate = 0.01: 0.8246445497630331
    c. Learning Rate = 0.001: 0.8578199052132701
    d. Learning Rate = 0.0001: 0.7488151658767772
    e. Learning Rate = 0.00001: 0.5781990521327014

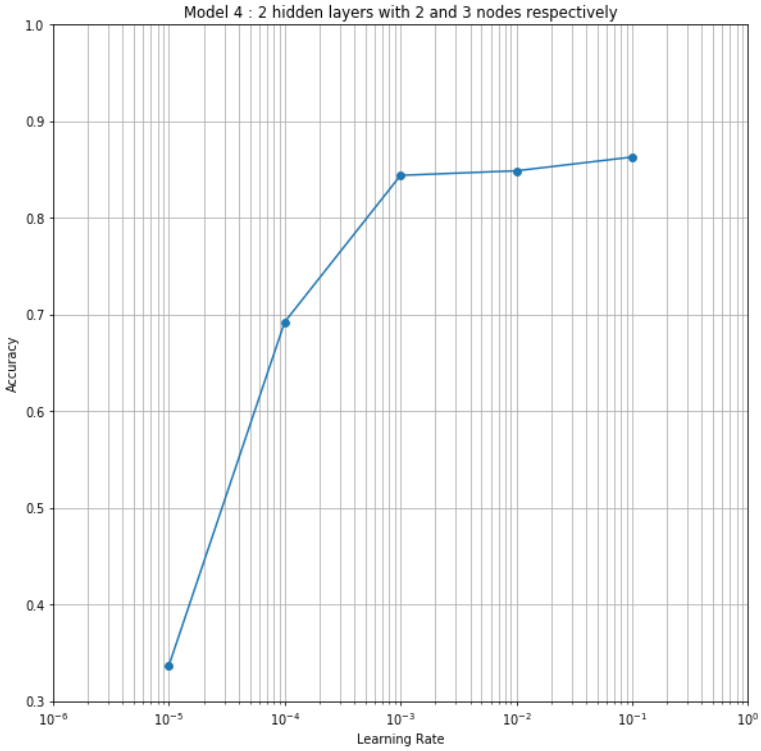
The maximum accuracy for each model is highlighted in blue and the maximum overall accuracy is highlighted in red.

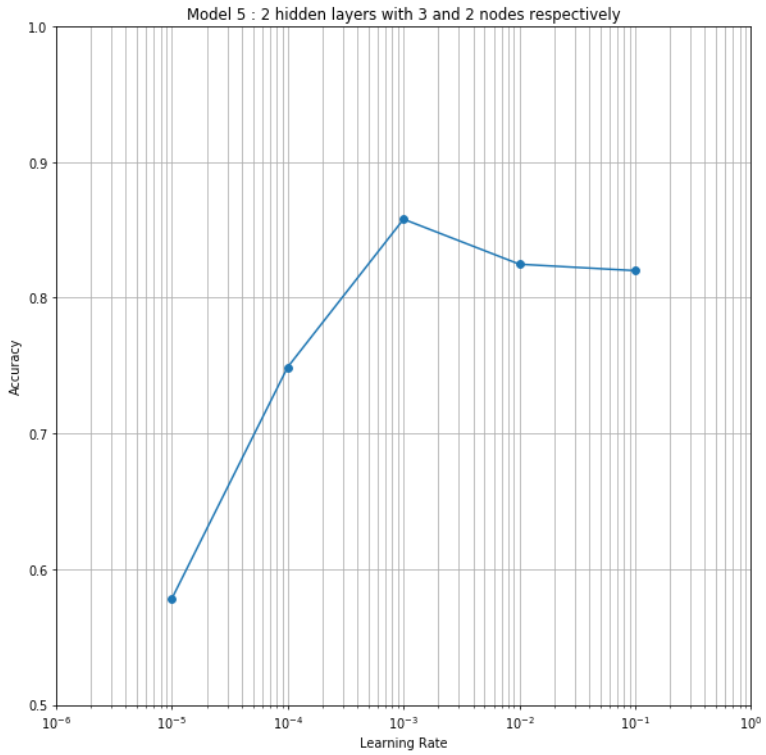
The concerned Learning Rate vs Accuracy Plots for each Model are:



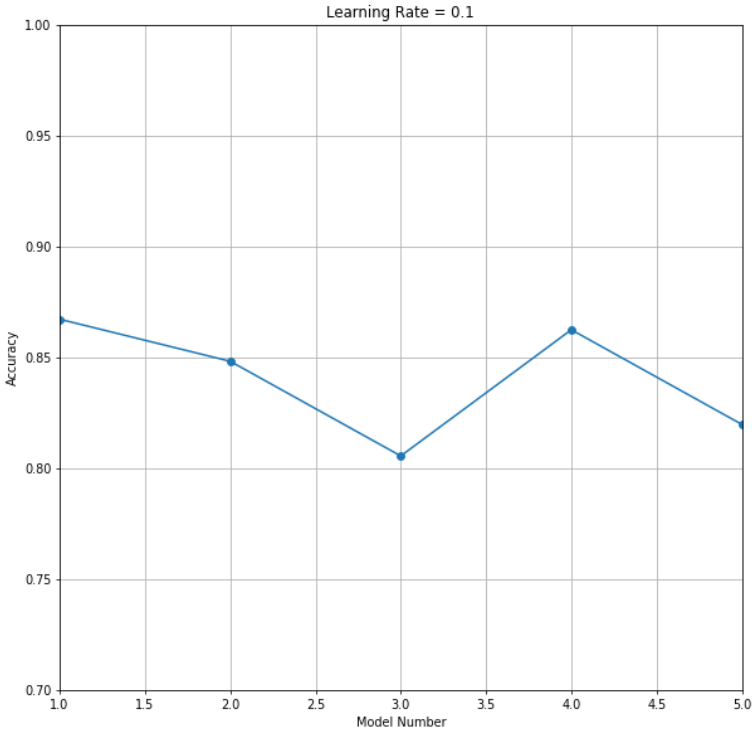


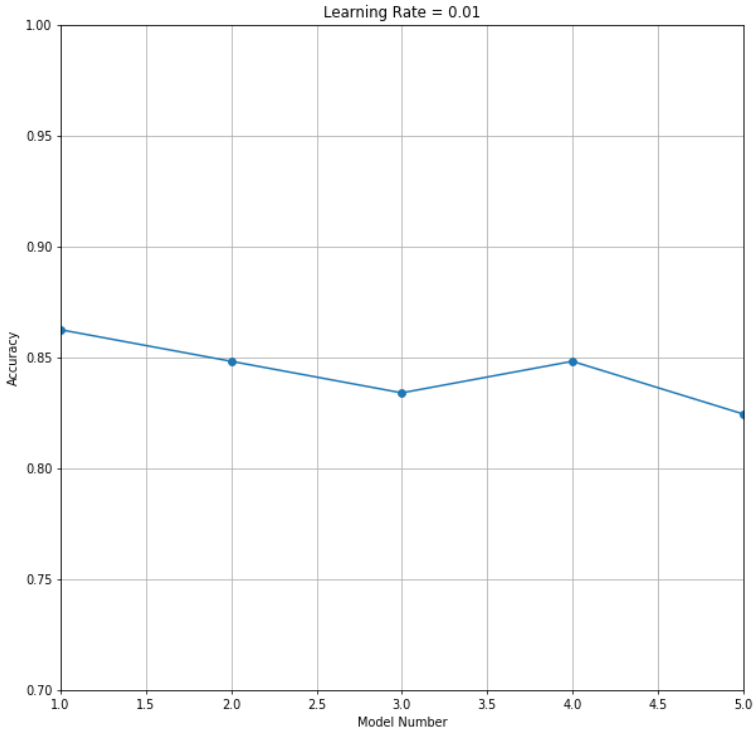


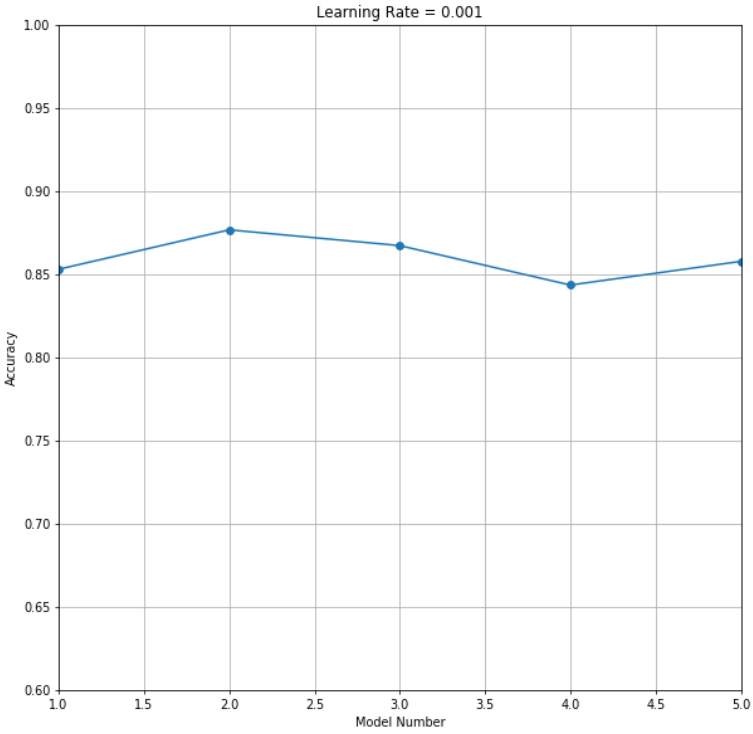


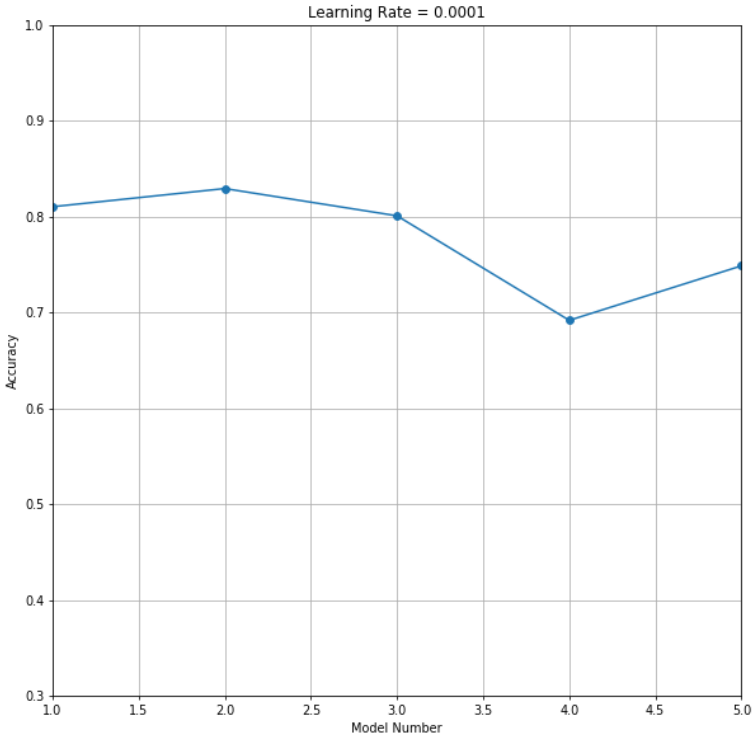


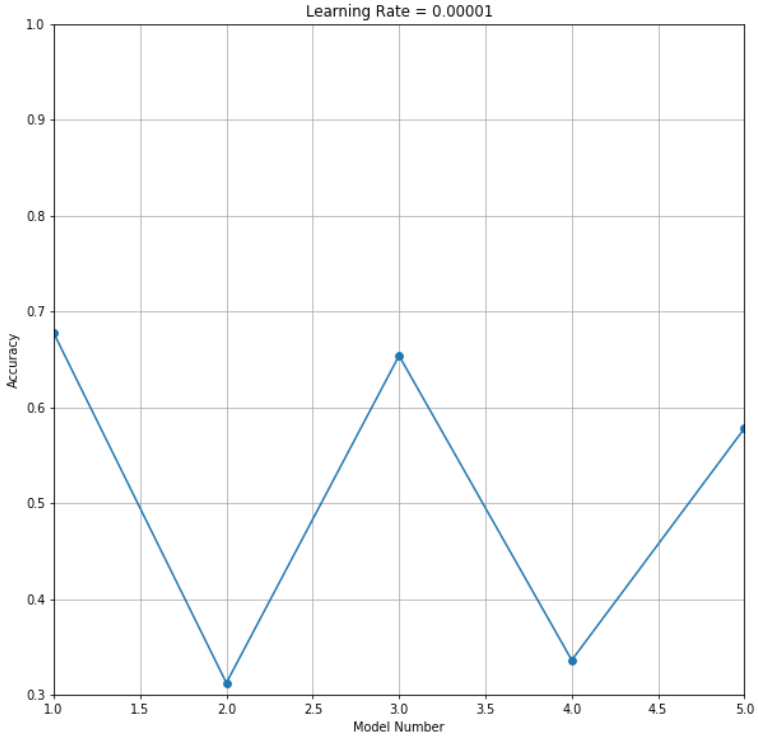
The concerned Model vs Accuracy Plots for each Learning Rate are:






Learning Rate = 0.001
Accuracy
Model Number
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Learning Rate = 0.0001
Accuracy
Model Number
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Learning Rate = 0.00001
Accuracy
Model Number
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

The best accuracy was observed for:

Model: 1 hidden layer with 2 nodes

Learning Rate: 0.01

Maximum Accuracy: 0.8767772511848341

Learning rate controls how quickly or slowly a neural network model learns a problem.Stochastic gradient descent is an optimization algorithm that estimates the error gradient for the current state of the model using examples from the training dataset, then updates the weights of the model using the back-propagation of errors algorithm, referred to as simply backpropagation.

The amount that the weights are updated during training is referred to as the step size or the "*learning rate*."

Given a perfectly configured learning rate, the model will learn to best approximate the function given available resources (the number of layers and the number of nodes per layer) in a given number of training epochs (passes through the training data).

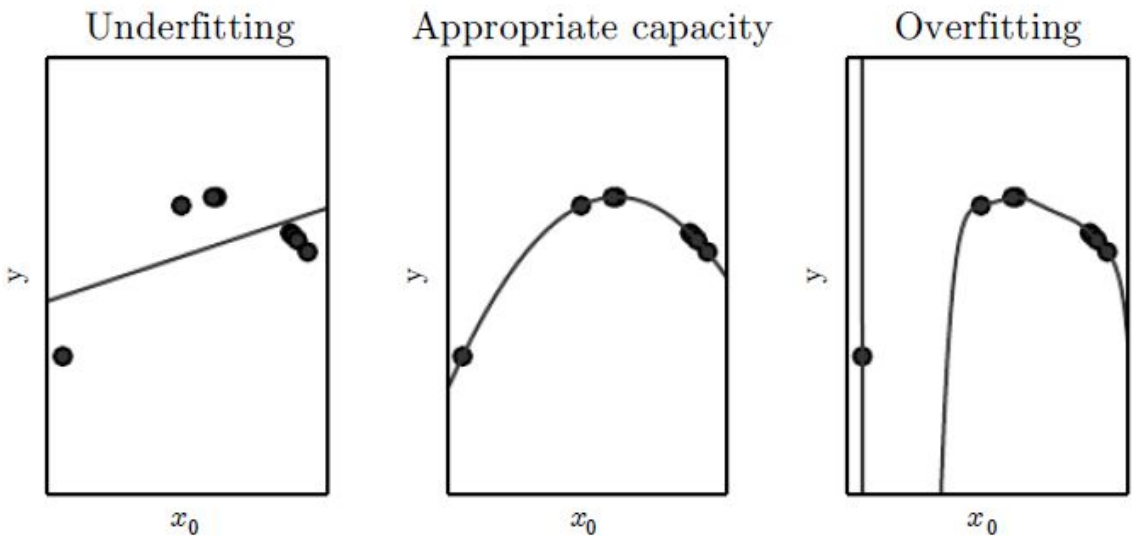
Generally, a large learning rate allows the model to learn faster, at the cost of arriving on a sub-optimal final set of weights. A smaller learning rate may allow the model to learn a more optimal or even globally optimal set of weights but may take significantly longer to train.

At extremes, a learning rate that is too large will result in weight updates that will be too large and the performance of the model (such as its loss on the training dataset) will oscillate over training epochs. Oscillating performance is said to be caused by weights that diverge (are divergent). A learning rate that is too small may never converge or may get stuck on a suboptimal solution.

So the learning rate that gave the best accuracy was **0.01** and accuracy tapered off on either side of 0.01 as explained above.

Also the best test accuracy was observed for the model which had **1 hidden layer with 2 nodes**. The accuracy was also seen to decrease significantly as the complexity was increased by increasing the number of hidden layers and the nodes in the hidden layers. Increasing the number of hidden layers much more than the sufficient number of layers will cause accuracy in the test set to decrease. It causes our network to overfit to the training set, that is, it learnt the training data, but wasn't be able to generalize to new unseen data.

The picture below gives a good intuitive idea about the concept:



Where in the left picture they try to fit a linear function to the data. This function is not complex enough to correctly represent the data, and it suffers from a bias(underfitting) problem. In the middle picture, the model has the appropriate complexity to accurately represent the data and to generalize, since it has learned the trend that this data follows (the data was synthetically created and has an inverted parabola shape). In the right picture, the model fits the data, but it overfits to it, it hasn't learnt the trend and thus it is not able to generalize to new data.

The input layer has nodes = Number of features in the dataset (in our case 41)

The output layer has nodes = Number of output labels (in our case 2)

Usually when the dataset is not too large, as in our case (1055 instances), a simpler model performs better over unseen examples as compared to more complex models. The complex models are very prone to overfitting, wherein they will have very high training accuracy but compromise over test accuracy.

Comparison:

The SVM Classifier performed better overall as compared to the MLP Classifier. The MLP Classifier had a maximum test accuracy of **89.099%** whereas the highest test accuracy for the MLP Classifier was **87.678%.** Although the 2 maximum values are pretty close, the overall average performance was much better for the SVG Classifier as it showed accuracy over 80% almost constantly whereas the MLP Classifier had accuracy as low as **31.269% (minimum)** with several values also lying in the range of **50%-70%.** Hence for the QSAR+biodegradation dataset, the SVM performed better than the MLP Classifier.