Machine Learning Assignment-2

Autumn Semester, 2022

Team Members:

Anand Manojkumar Parikh (20CS10007)

Soni Aditya Bharatbhai (20CS10060)

Supervised Learning Methods

[*Refer the README file in this folder to get directions for executing the source code]

Contents:

1) Pre-processing

Pre-processing the data includes the following steps, explained in detail:

- Encoding categorical data:

  In the dataset “abalone.data” there is only 1 attribute “sex” that is a categorical variable, with the following encodings:

  Male -> 0

  Female -> 1

  Infant -> 2

  A function “encode” has been implemented to achieve this.

- Normalization:

  The data needs to be normalized in order to avoid any implicit bias in favour of any attribute. So, all attributes have been normalized as follows:

  attribute_value = (attribute_value – column_mean) / column_std

  Where attribute_value is the original value, column_mean and column_std are respectively the mean and standard deviation of all values of that attribute in the dataset.

  A function “normalize” has been implemented to mimic the working of standard scaler normalization.

- Randomization:

  To get best results, the order of given data has been shuffled row-wise using an inbuilt function “sample”.

- Train-Test Split:

  The available data has been split as follows:

  Training set – 80%

  Test set – 20%

  A function “train_test_split” has been implemented to achieve this.

2) SVM Classifier

The Support Vector Machine has been used for classification of the dataset as follows:

1. Three objects were created, one for each kernel type : Linear, Quadratic, Radial Basis Function using the “SVC” class of library “sklearn”.
2. They were fit onto the training set and used to predict the classes of the test set.
3. Then the predictions were evaluated against the actual test classes. The accuracy results for 3 consecutive runs of the program have been tabulated below-

Kernel	Run 1	Run 2	Run 3
Linear	0.26435406698564595	0.2703349282296651	0.2727272727272727
Quadratic	0.24760765550239233	0.25717703349282295	0.24282296650717702
RBF	0.2811004784688995	0.284688995215311	0.2715311004784689

3) MLP Classifier

Multi-layer Perceptron Classification has been used for the classification of the dataset as follows:

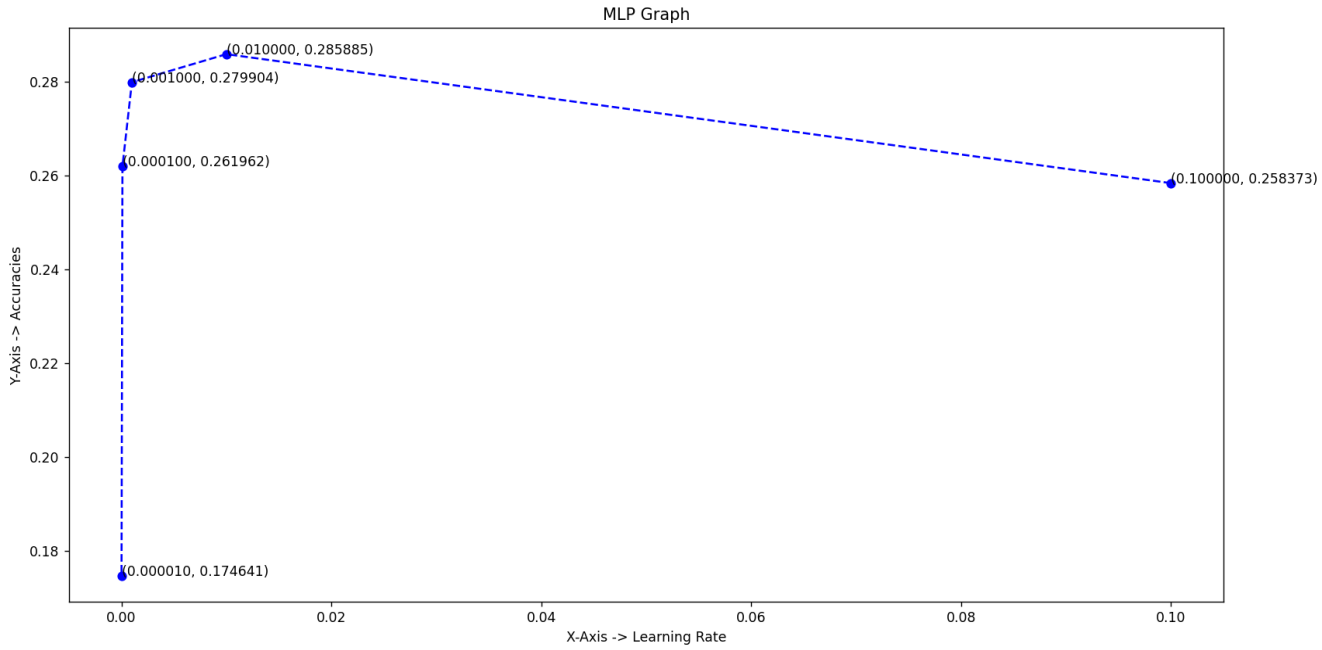
1. Two objects were created, one having a hidden layer of 16 nodes and another having 2 hidden layers of 256 and 16 nodes respectively using the “MLPClassifier” class of library “sklearn”. Both models have the same constant learning rate of 0.001 and a batch size of 32.
2. They were fit onto the training set and used to predict the classes of the test set.
3. Then the predictions were evaluated against the actual test classes. The accuracy results for 3 consecutive runs of the program have been tabulated below-

Hidden Layers	Run 1	Run 2	Run 3
16	0.2834928229665072	0.29545454545454547	0.2811004784688995
256, 16	0.2822966507177033	0.27751196172248804	0.2811004784688995

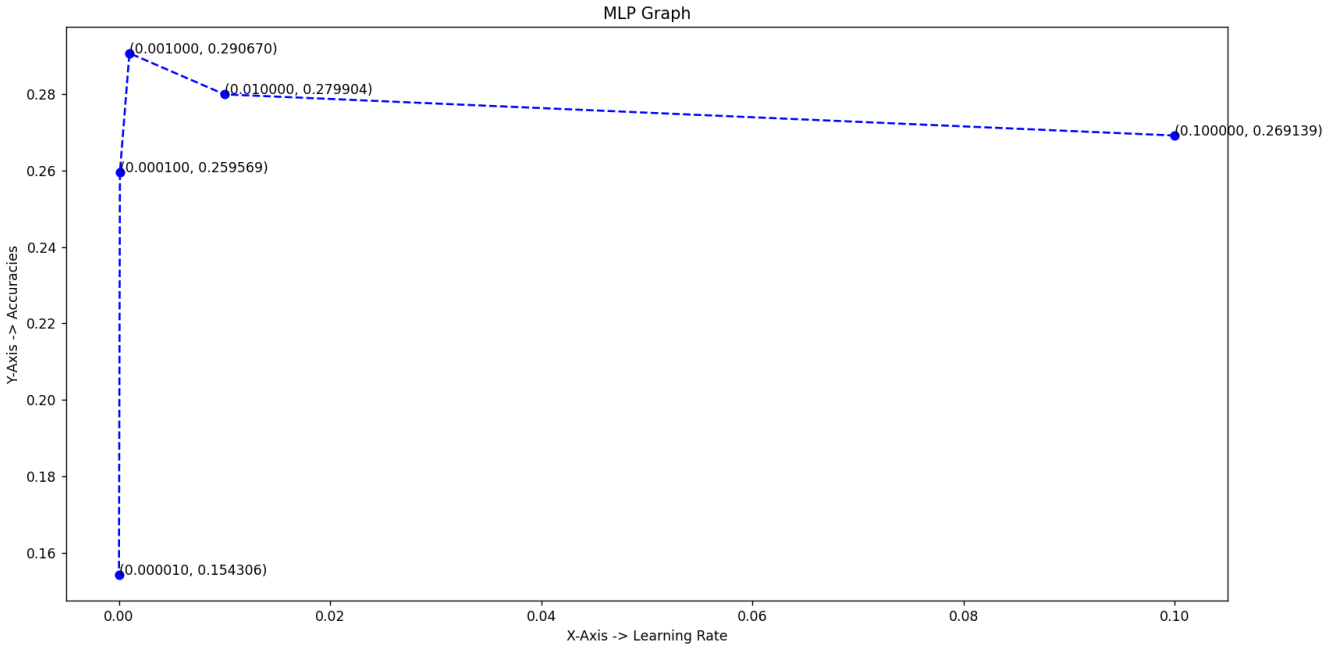
4) Graphs

The graphs obtained on 3 consecutive runs of the code are provided below.

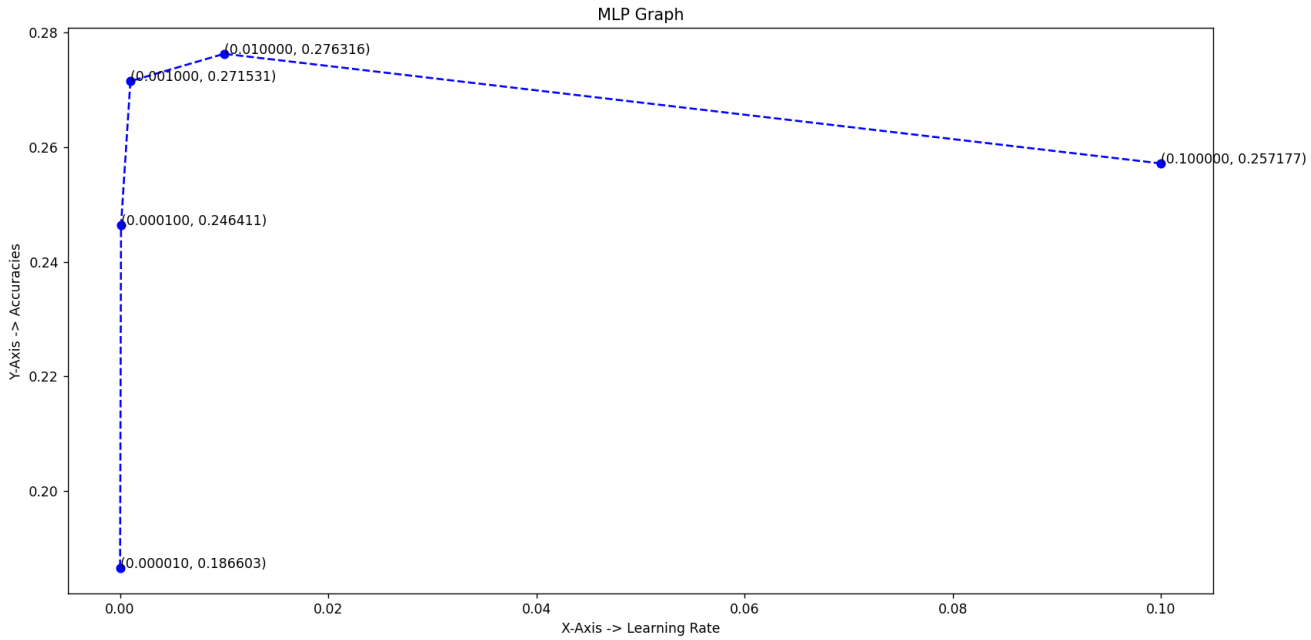
- Graph # 1



- Graph # 2



- Graph # 3



- Observations:
  The following observations and conclusions can be drawn looking at the above graphs:
  - The most important thing that we observe here is that the accuracy is at its peak at an intermediate value. If we either increase or decrease the learning rate from the optimal learning rate, then the accuracy drops on both sides.
  - On the left side, learning rate decreases, so it the procedure becomes very slow, requiring a much higher number of iterations to converge. As a result, it does not converge at all if we have an upper bound on the number of max iterations (in our case, implicitly set to 200 by “sklean”) leading to bad accuracy.
  - On the right side, learning rate increases, leading to drastic updates and might even diverge, again leading to bad accuracy.
  - So we need to maintain an intermediate optimal value of the learning rate hyper-parameter, which seems to be in the range [0.001,0.01] in our case.

5) Backward Elimination

Backward elimination is a greedy method to reduce the dimensionality of the model without affecting the accuracy on the test set. The algorithm proceeds as follows:

- Start with all attributes. Initialize previous accuracy to the current accuracy of the model trained in the training set and tested on the test set.
- In each iteration, temporarily remove one of the existing attributes. Then re-fit and re-predict the model on the train and test sets, respectively. Find the accuracy score of the model against the actual test classes.
- Do this for each existing attribute and find the maximum accuracy that can be gained by removing any one attribute.
- If this maximum accuracy is more than the previous accuracy, then permanently delete that attribute from the dataset, since it does not make sense to have more dimensions and lower accuracy. Update previous accuracy to this maximum accuracy, since this is now our actual model.
- Else (if maximum accuracy is less than the previous accuracy), terminate the process without removing that attribute, since removing any single attribute harms the accuracy of our model, which is not desirable even though the number of dimensions reduces.
- Finally, the set of features that survive through this process are our "best" set of features. Note that the greedy nature of this algorithms does not guarantee that the optimal subset is found. What we have done is a poly-time approximation of an NP-Complete problem.
- A function "backward_elimination" has been implemented for this purpose.
- The "best" subsets as per backward elimination have been tabulated below:
- Each cell in the table contains 3 items in this order:
    1. Feature, whose removal gives maximum accuracy.
    2. Accuracy achieved if that feature is removed.
    3. Final verdict – removed / not removed based on comparison with previous accuracy.

	Initial Accuracy	Iteration 1	Iteration 2	Iteration 3	Iteration 4	"Best" subset
1	0.27033	shucked_weight 0.283492 Removed	diameter 0.291866 Removed	sex 0.281100 NOT removed	-	sex, length, height, whole_weight, viscera_weight, shell_weight
2	0.27990	sex 0.285885 Removed	height 0.290669 Removed	length 0.288277 NOT Removed	-	length, diameter, whole_weight, shucked_weight, viscera_weight, shell_weight
3	0.27751	sex 0.295454 Removed	length 0.295454 Removed	viscera_weight 0.2990430 Removed	shell_weight 0.2954545 NOT Removed	diameter, height, whole_weight, shucked_weight, shell_weight

6) Ensemble Learning

Three models – support vector with quadratic kernel, support vector with RBF kernel and the best of the 2 MLPs have been used for ensemble learning as follows:

- Each of the 3 models have been trained on the training set and used to predict the classes of the test set individually.
- The independent vote of each of them is taken. The class having the maximum frequency (maximum vote) out of these 3 is declared as the final class for that test example.
- A function "ensemble" has been implemented for this purpose.
- This is done for each test example and the outcomes have been evaluated against the actual test classes. The accuracies obtained in 3 consecutive runs of the program are tabulated below:

Run	Ensemble Accuracy
1	0.27631578947368424
2	0.27870813397129185
3	0.2703349282296651
Average	0.275119617222

7) Observations and Conclusions

- Each run of the MLP Classifier on the same training and testing data produce different accuracies each time. This is because the "random_state" parameter has not been explicitly mentioned. This helps in implicitly adding randomization to the dataset.
- The Stochastic Gradient Descent Optimizer used to train the MLP Classifier throws the "convergence warning" each time the program runs (these warnings have not been registered in the output file for clarity) because even though the maximum number of iterations (implicitly set to 200 by "skelearn") have been reached but the classifier has not yet converged. One way to solve this would be to manually increase this limit to say, 500. However, this would make the code execution extremely slow and hence, is not suggested.
- The set of features removed by backward elimination are different each time. An explanation for this would be that each time the train and test split are different. This would cause different attributes to be more or less important to classification based what split becomes the training set. Also, which attributes are removed also depends on what has already been removed, making this process very non-deterministic. Nevertheless, a general pattern that can be observed is that the attribute "sex" gets removed each time, and it indicates that it is not a very strong determiner of the target "rings".