Machine Learning

Boston Housing Data Set – MLP for Regression

Use MLP for regression on the Boston housing dataset.

download the data from https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html. There are 506 patterns and 14 features with MEDV as the target variable.
(a) Train an MLP net to get the best squared error value on the regression dataset in (a) using 1, 2, 3, and 4 hidden layers.
(b) Vary the parameters considered in Subtask 1(d) and compute the squared error.

SOLUTION:

CODE:

This Problem targets to apply MLP for regression on the Boston housing data.
Please find the code committed for this task as *01_MLP_Regression_BostonHousing_DataSet_Impl.py*

- **MLP Regressor** from *sklearn.neural_network* is used to solve this task.
- The *BostonHousingData.csv* is downloaded and used as dataset. It has 506 patterns and 14 features (13 features in data frame X and 14th feature MEDV as the target variable(y))
- **"rm"**, **"ptratio"** & **"lstat"** are most contributing features and are used here.
- Housing data is pre-processed using *StandardScaler* from *sklearn.preprocessing*
- Test Size is 0.2 and is fixed for all iterations here.
- Various MLP Regressors are created applying many permutations obtained on varying :-.
  - **Activation function** as tanh, relu, logistic.
  - **Hidden layers** as 1-Layer([64]), 2-Layers([64,32]), 3-Layers([64,32,16]), 4-layers([64,32,16,8]).
  - **Learning rate** as constant, adaptive
- **Squared error(RMSE)** is computed and documented in Result section below with each of these permutations.

RESULT:

```
Activation Function = relu
=====================================================
1 Layer having 64 nodes per layer
===> RMSE  for MLP Regressor train size 0.8 = 16.480000
2 Layer having [64,32] nodes per layer
===> RMSE  for MLP Regressor train size 0.8 = 5.400000
3 Layer having [64,32,16] nodes per layer
===> RMSE  for MLP Regressor train size 0.8 = 4.870000
4 Layer having [64,32,16,8] nodes per layer
===> RMSE  for MLP Regressor train size 0.8 = 4.500000

Activation Function = tanh
=====================================================
```

1 Layer having 64 nodes per layer
===> RMSE for MLP Regressor train size 0.8 = 17.700000
2 Layer having [64,32] nodes per layer
===> RMSE for MLP Regressor train size 0.8 = 10.610000
3 Layer having [64,32,16] nodes per layer
===> RMSE for MLP Regressor train size 0.8 = 14.380000
4 Layer having [64,32,16,8] nodes per layer
===> RMSE for MLP Regressor train size 0.8 = 18.370000

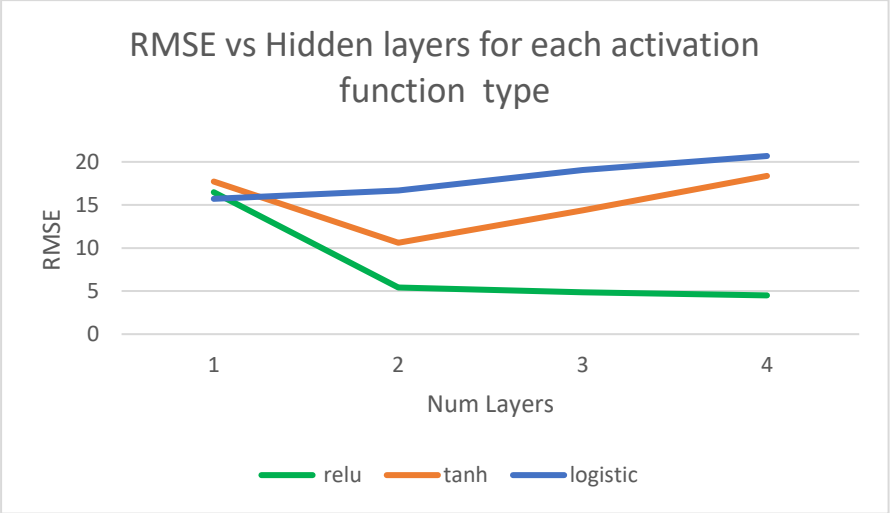
Activation Function = logistic

==========

1 Layer having 64 nodes per layer
===> RMSE for MLP Regressor train size 0.8 = 15.710000
2 Layer having [64,32] nodes per layer
===> RMSE for MLP Regressor train size 0.8 = 16.670000
3 Layer having [64,32,16] nodes per layer
===> RMSE for MLP Regressor train size 0.8 = 19.060000
4 Layer having [64,32,16,8] nodes per layer
===> RMSE for MLP Regressor train size 0.8 = 20.680000

PLOT:

Here is a plot of K (number of clusters) vs Avg Accuracy.



INFERENCE/ANALYSIS:

- **MLP Regressor** works well and provides a low RMSE on Boston Housing Data.
- Selection of most contributing features is very necessary as training with entire 13 features will be counterproductive in this real-word data.
- Data pre-processing is very vital here for the model to work properly, otherwise higher RMSE values will be seen.
- From accuracy plot, we can see that when using relu as activation function, with increase in hidden layer numbers, the RMSE is decreasing whereas the reverse is the case with logistic function.
- From accuracy plot, we can see that when using tanh as activation function, although increase in hidden layers contributes positively thereby decreasing the RMSE but the saturation is observed at Layer = 2 and further increasing num_layers is giving poorer results(higher RMSE)

So in conclusion – When MLP Classifier is applied over digit's dataset completely , the average accuracy of classification is very high. Additionally, when MLP regressor is applied on the Boston Housing Data , Low RMSE results were seen when we used data pre-processing and main contributing features for training our ML Models. Through the exercise of generating many permutations by varying key parameters, we can clearly see that tanh/relu are better performing activation functions for these data sets and with increasing number of hidden layers the performance is getting better.

RESOURCES USED FOR THE ASSIGNMENT:

• **Environment:** Anaconda, Jupyter notebook
• **Software :** Python **Python libraries/modules:** Pandas, Numpy, SkLearn, Scipy, matplotlib, etc