CS - 7641 - Machine Learning

February , 2021

Assignment - 1

Introduction:

The main intention of this assignment is going to be the implementation and analysis of the below algorithms, on two different datasets.

1. Decision Trees
2. Artificial Neural Networks
3. Boosting
4. Support Vector Machines(SVM)
5. K-nearest neighbours

Datasets:

The two different datasets that I have picked as part of the assignment are,(Both have been picked from UCI machine learning repo)

1. **Wireless Indoor Localization Data Set (WILDS):**
   Collected to perform experimentation on how wifi signal strengths can be used to determine one of the indoor locations. Each attribute is wifi signal strength observed on smartphones. This data has 2000 instances with 7 features.
2. **Letter recognition: (LR)**
   In contrast to the above dataset, I have picked LR dataset which has 20000 instances and 16 features. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15.

Decision Trees:(Implemented with sklearn `cost_complexity_pruning_path`)

It was interesting to also look into cost_complexity_pruning_path technique as part of sklearn DT implementation. CCPP is a post pruning technique and is parametrized by `ccp_alpha`.Greater values of `ccp_alpha` increase the number of nodes pruned. Finding the right ccp_alpha was important as it affects the accuracy of the model. So we would have to be very careful in choosing ccp_alpha in making sure we don't overfit the data and at the same time prune the tree and still maintain a good accuracy score.

Below are the decision trees(using Entropy) on both the datasets:

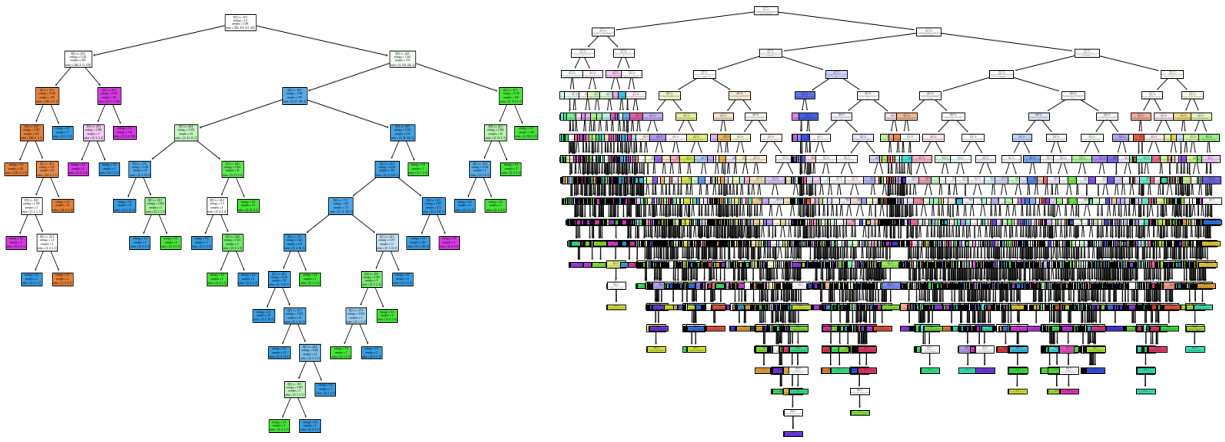
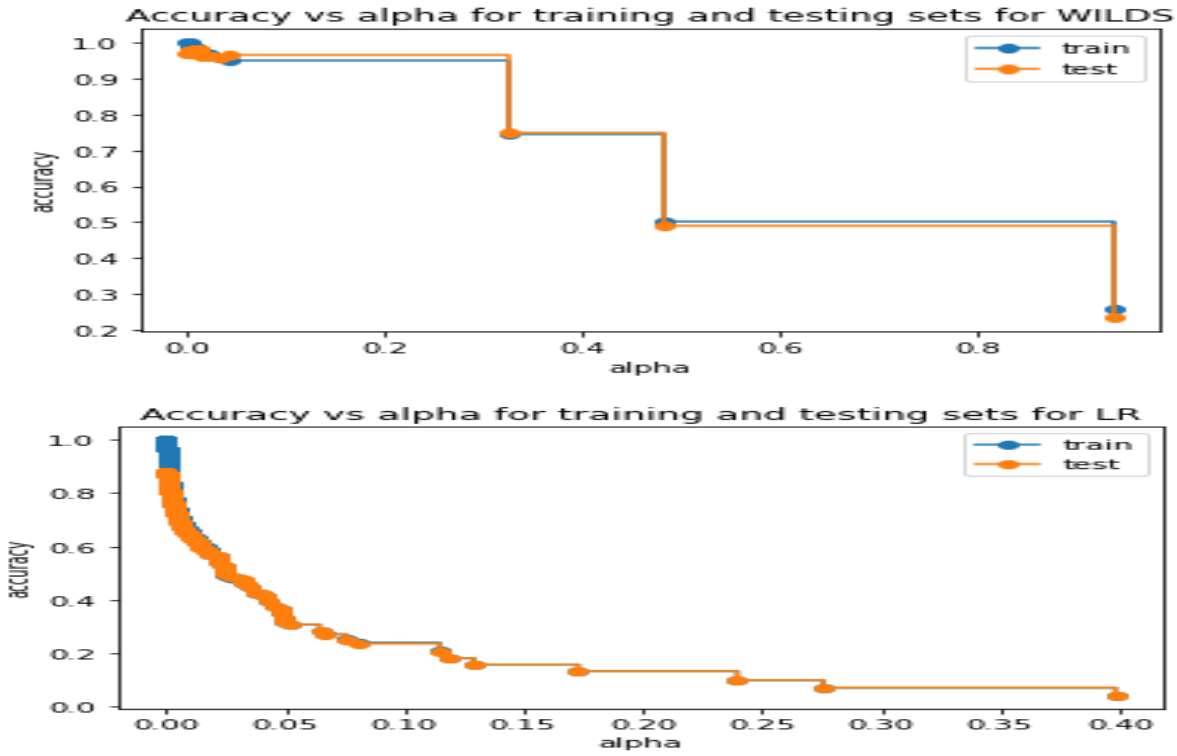


Figure 1: Complete decision trees(**No Pruning**) of both datasets **WILDS**(Left) and **LR**(Right)
Accuracy scores: WILDS(`0.9716666666666667`) and LR(`0.88125`)

After leveraging `cost_complexity_pruning_path` for obtaining the impurities and several ccp_alpha values, these ccp_alpha values were all trained on the DT for compare the training and testing test accuracy, below are the graphs,



Above two charts have helped in picking the **right sweet spot based on training and testing accuracy scores and alpha values**. Logic behind the pruning technique used is, minimal-cost-complexity-pruning.

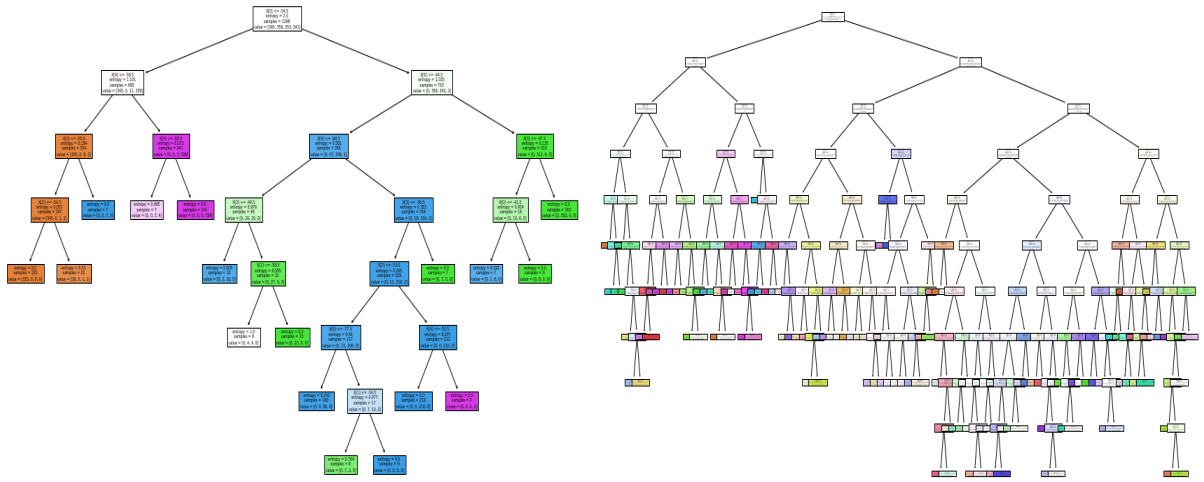


Figure: Decision Trees **after applying pruning** based on `cost_complexity_pruning_path`. Accuracy scores: WILDS(`0.9766666666666667`) and LR(`0.88125`)

Accuracy scores have increased slightly after applying pruning to WILDS dataset decision trees. So pruning helps in generalizing by avoiding too much overfitting on the data and but was not the similar case with LR dataset, even after picking the right alph value, the accuracy of the model did not improve.

Artificial Neural Networks:

For implementing Artificial neural networks, I have opted to use KERAS. A lot of methods come in handy using keras.
Different hyper-parameters tweaks have been used in order to optimize the Neural network model,

1. Normalization of data.
2. Different activation functions (Sigmoid and Relu).
3. Different combinations of layers and nodes per layer.
4. Momentum.
5. Number of Epochs and batch_size
6. Optimization algorithms(Gradient descent vs mini batch gradient descent).

Initial run consideration,
Dataset: **WILD**

Layers = 3, L1_Units = 10, L2_Units = 5, L3_output = Classification categories

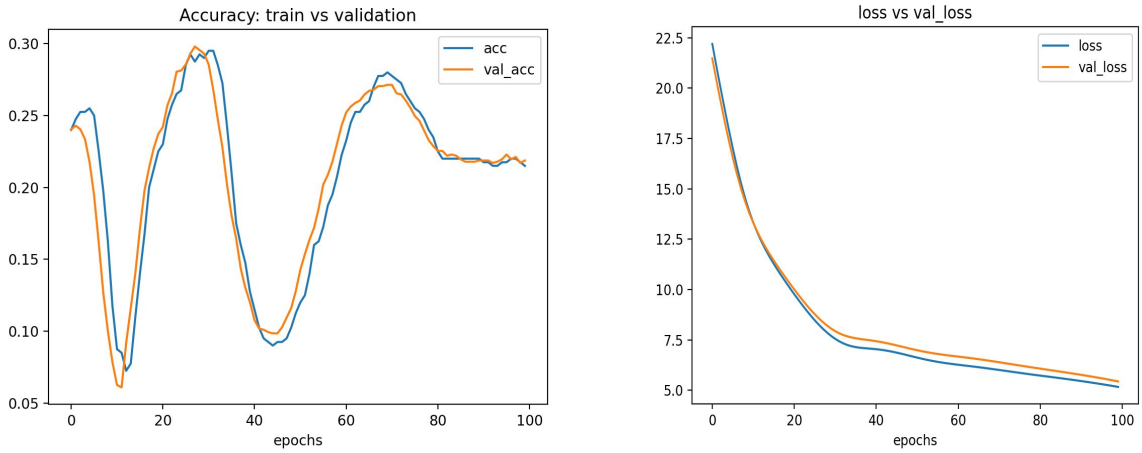
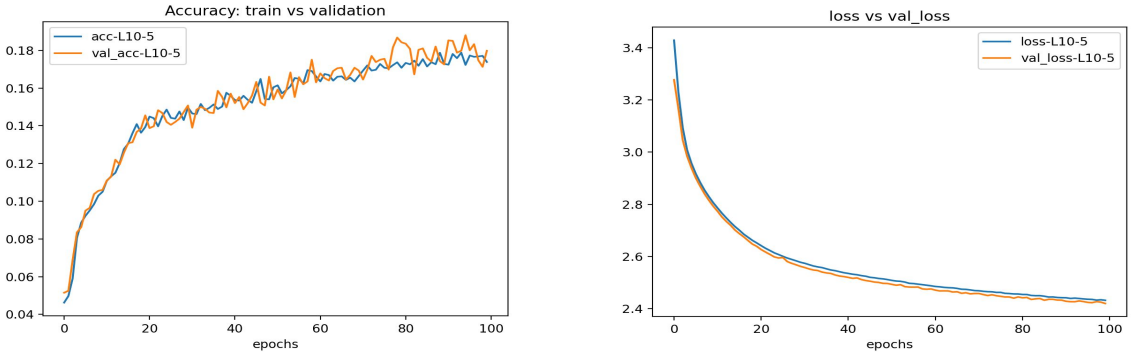


Figure-5

Dataset: **LR**

Layers = 3, L1_Units = 10, L2_Units = 5, L3_output = Classification categories

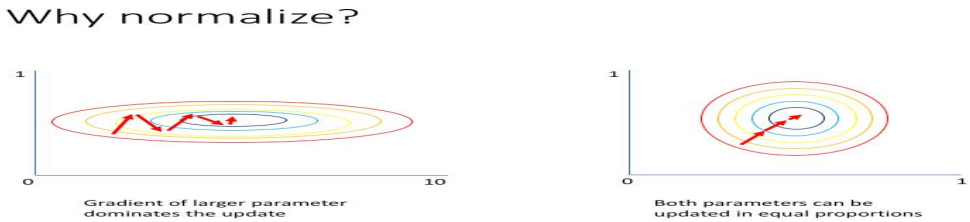


Optimization-1 (Normalize the input data):

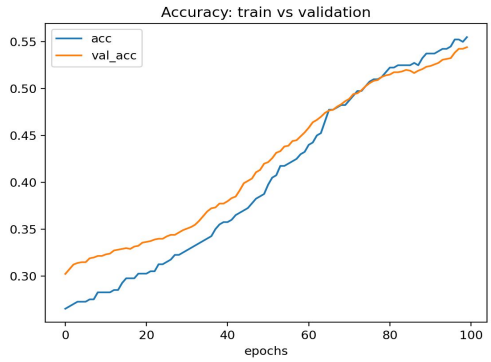
Dataset: **WILD**

Layers = 3, L1_Units = 10, L2_Units = 5, L3_output = Classification categories

Normalizing the input data is helpful during loss/optimization phase, as the loss function if plotted looks like a sphere and is easy to traverse for local minima.



There has been a little improvement with normalizing the inputs, but definitely need a lot more optimizations to be done in order to get better with accuracy scores of both the train and validation sets.



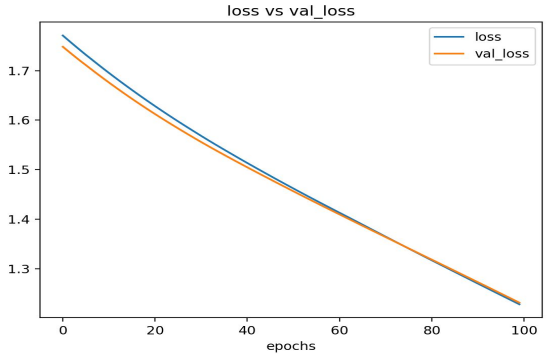
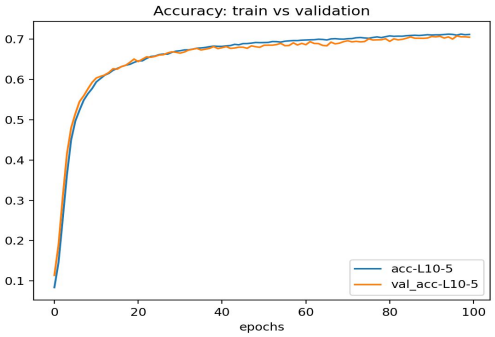
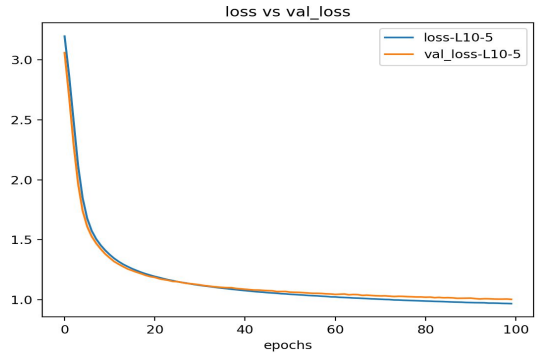


Figure-6

Dataset: **LR**

Layers = 3, L1_Units = 10, L2_Units = 5, L3_output = Classification categories

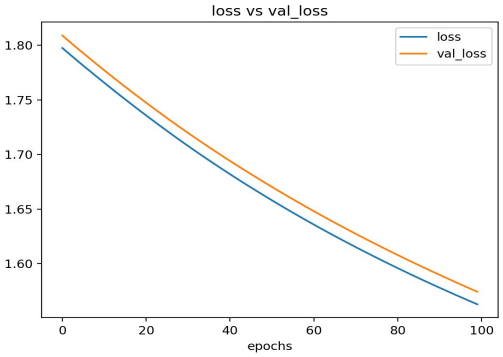


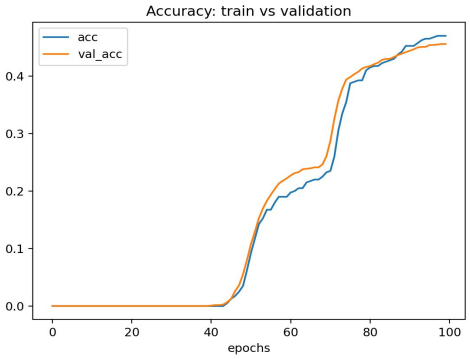


Optimization-2: (Activation functions RELU vs Sigmoid)

Dataset: **WILD - Sigmoid**

Layers = 3, L1_Units = 10, L2_Units = 5, L3_output = Classification categories

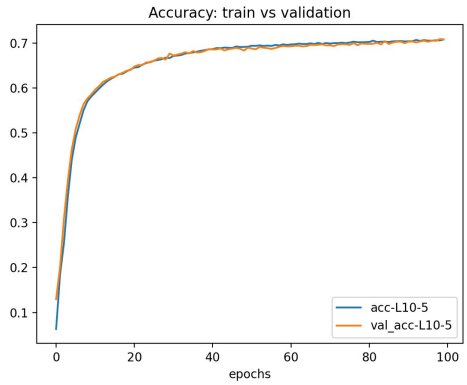


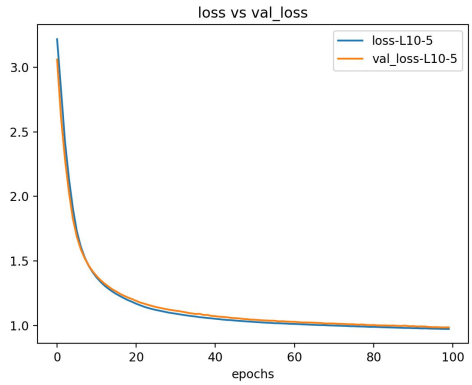


Clearly comparing the plots with Firgure-6, accuracy has been depleted in Figure-7, with minute differences in loss plots. So have opted in considering RELU for activation.

Dataset: **LR - Sigmoid**

Layers = 3, L1_Units = 10, L2_Units = 5, L3_output = Classification categories





On the other hand, there has not been any difference in the above plots between RELU and Sigmoid for LR Dataset.

Optimization-3: (layers and nodes per layers)

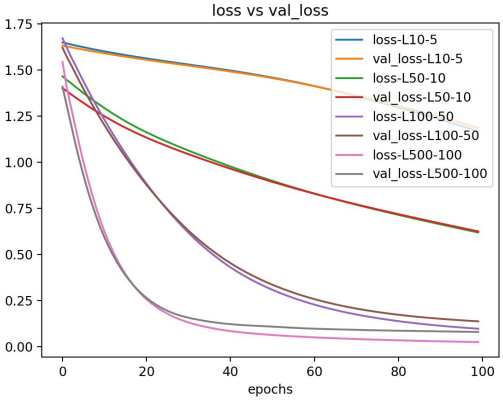
Dataset: **WILD - Relu**

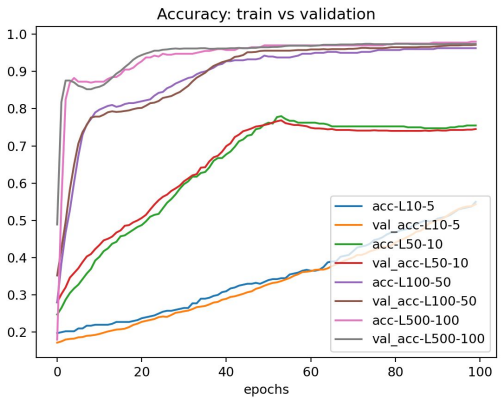
Layers = 3, L1_Units = 10, L2_Units = 5, L3_output = Classification categories

Layers = 3, *L1_Units = 50, L2_Units = 10, L3_output = Classification categories*

Layers = 3, *L1_Units = 100, L2_Units = 50, L3_output = Classification categories*

Layers = 3, *L1_Units = 500, L2_Units = 100, L3_output = Classification categories*





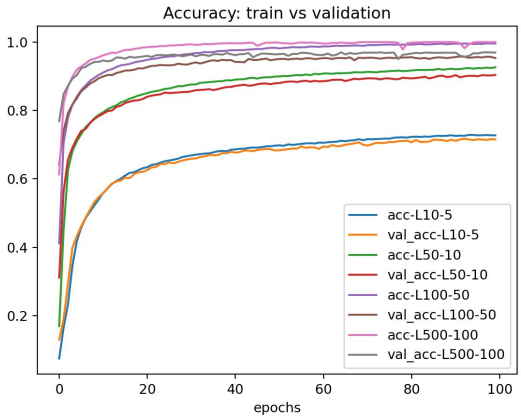
Dataset: **LR - Relu**

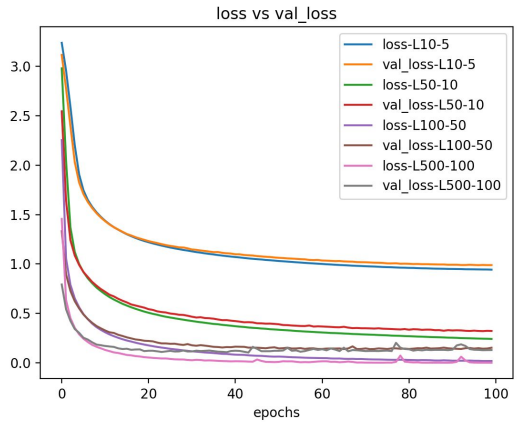
Layers = 3, L1_Units = 10, L2_Units = 5, L3_output = Classification categories

Layers = 3, *L1_Units = 50, L2_Units = 10, L3_output = Classification categories*

Layers = 3, *L1_Units = 100, L2_Units = 50, L3_output = Classification categories*

Layers = 3, *L1_Units = 500, L2_Units = 100, L3_output = Classification categories*





Above plots clearly show the distinction between different units under 3 layer network and it is very clear that with more number of nodes in layers, the accuracy of the models got much better, although, wrt the end results, there was not much difference between the largest two networks(L500-100 and L100-50). Even with loss, the largest two networks(L500-100 and L100-50) resulted in very minimum loss compared to other types of networks used for comparison.

Optimization-4: (Optimization algorithms(Gradient descent vs mini batch gradient descent), Epoch's: 50.

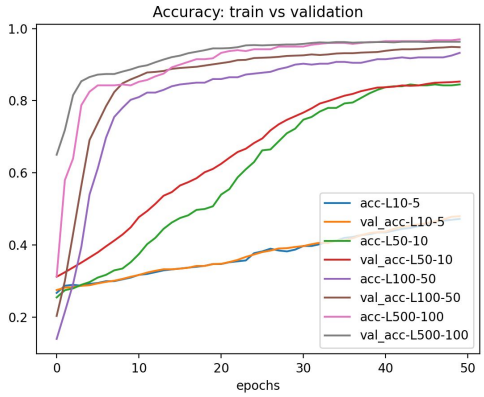
Dataset: **WILD - Relu**

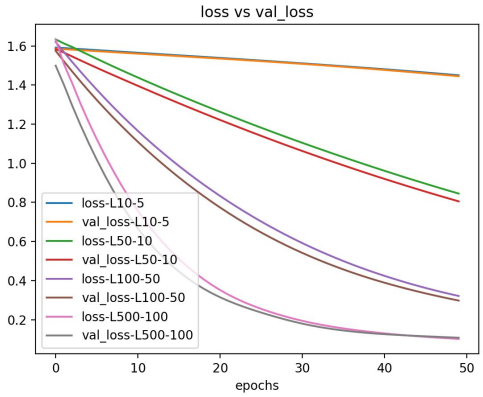
Layers = 3, L1_Units = 10, L2_Units = 5, L3_output = Classification categories

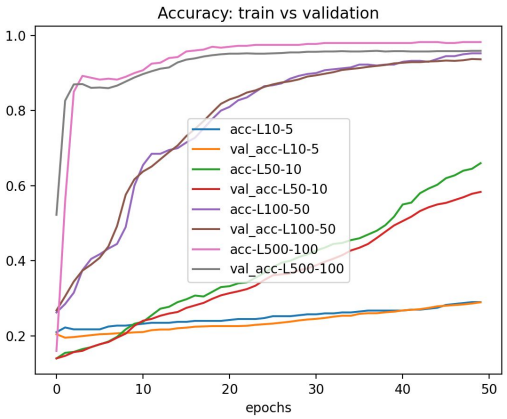
Layers = 3, *L1_Units = 50, L2_Units = 10, L3_output = Classification categories*

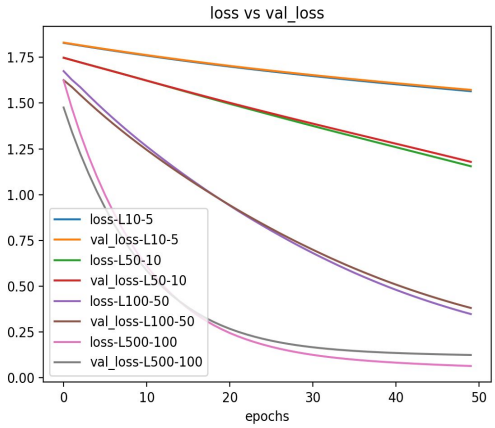
Layers = 3, *L1_Units = 100, L2_Units = 50, L3_output = Classification categories*

Layers = 3, *L1_Units = 500, L2_Units = 100, L3_output = Classification categories*









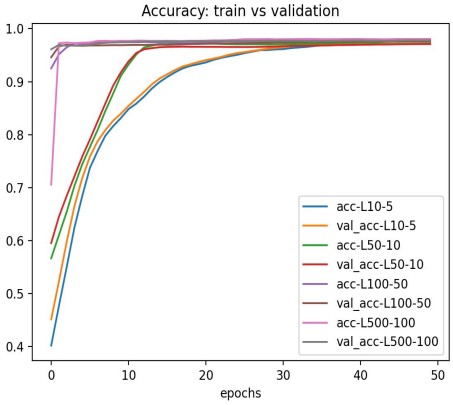
Dataset: **LR - Relu**

Layers = 3, L1_Units = 10, L2_Units = 5, L3_output = Classification categories

Layers = 3, *L1_Units = 50, L2_Units = 10, L3_output = Classification categories*

Layers = 3, *L1_Units = 100, L2_Units = 50, L3_output = Classification categories*

Layers = 3, *L1_Units = 500, L2_Units = 100, L3_output = Classification categories*



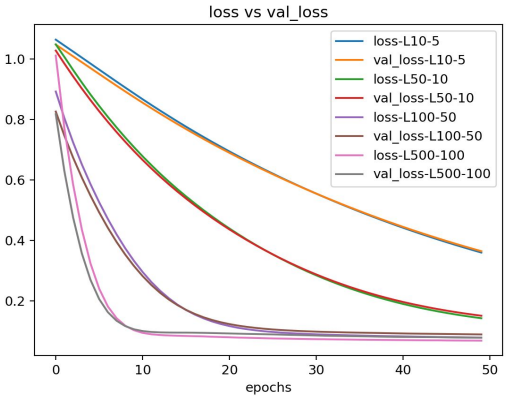


Figure-Gradient descent

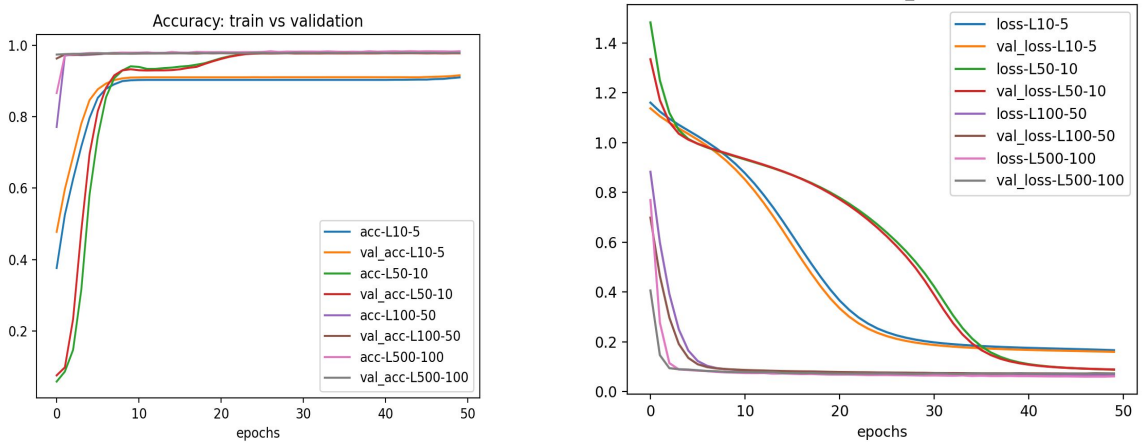
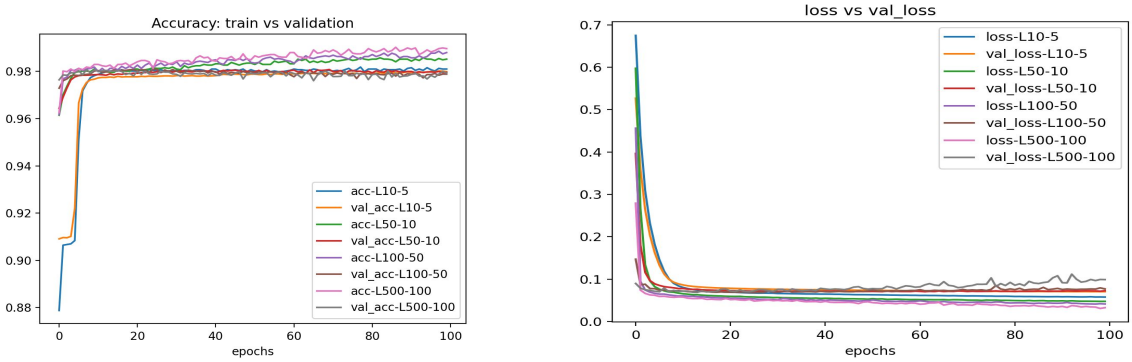


Figure-Mini-Batch GD

One major observation is that there is not much of difference with GD and Mini-GD loss and accuracy, and which makes sense and the differences is basically that when we only consider a batch and update the parameters and bias on all layers and because the gradient is approximated and does not go straight into the lower point of loss function, so we see the plots kind of have step breaks if carefully noticed. The above plots are too small to notice the mini batch friction curves, but below is a clear picture of not being smooth but having **friction across the lines**.



Mini-Batch is going to have clear advantages when our dataset is too huge, in case of a smaller dataset we can use GD directly considering the entire dataset and the gradient is going to move towards the lowest point smoothly.

K-nearest neighbours:

Have used Sklearn to implement KNN. My analysis for KNN mainly focuses on,

- Right K to pick
- Performance of two distance function(Euclidean ,Manhattan)

Firstly in order to pick or even analyze the number of K that would be performing well for the model, have created plots using KNN with K between 1 to 8.

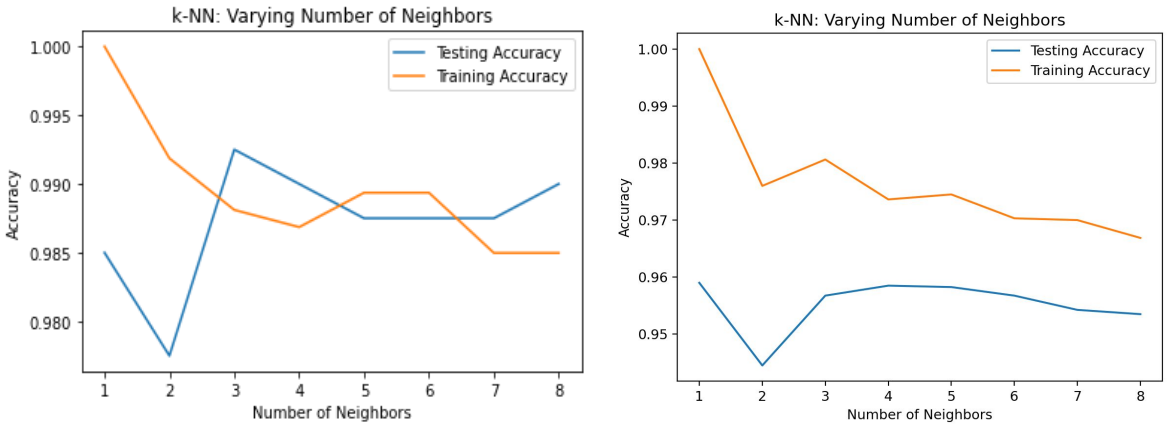


Figure:Left(WILD dataset) and Right(LR dataset)

From the above plots we can pick the right spot to decide on K, we would have to pick the spot where Testing and Training lines are consistently close with good accuracy scores. Clearly for LR data 4-5 is a good candidate for K and for WILD 5-6 is a good spot for K.

Euclidean vs Manhattan:

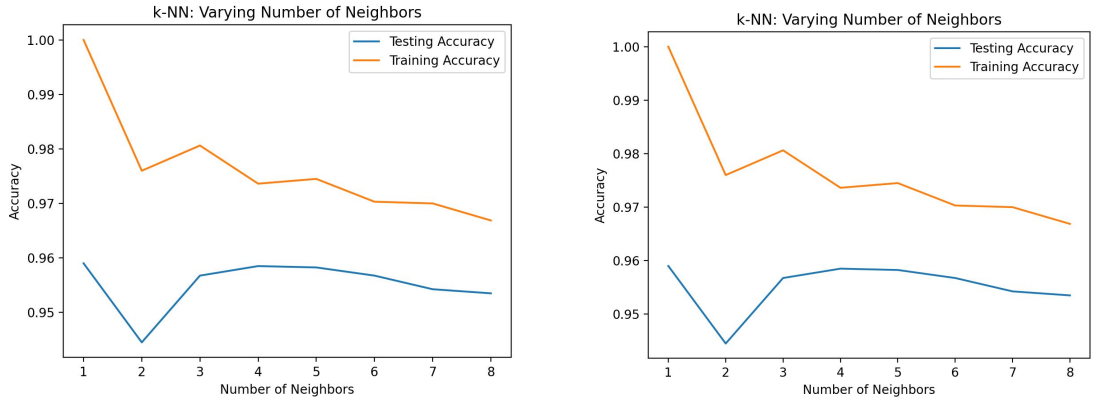


Figure:Distance function Euclidean(Left) vs Manhattan(Right) on LR dataset

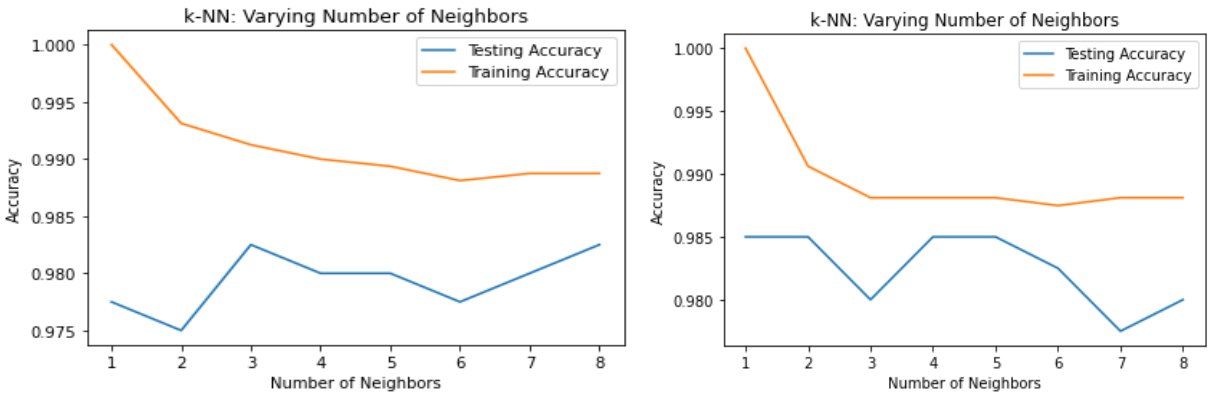


Figure:Distance function Euclidean(Left) vs Manhattan(Right) on WILD dataset

Boosting:

Have applied boosting(AdaBoosting) using sklearn package. Sklearn package has `AdaBoostClassifier` which also takes any initial model. As per assignment instructions have passed `DecisionTreeClassifier` as initial model and have applied Boosting with multiple estimators.

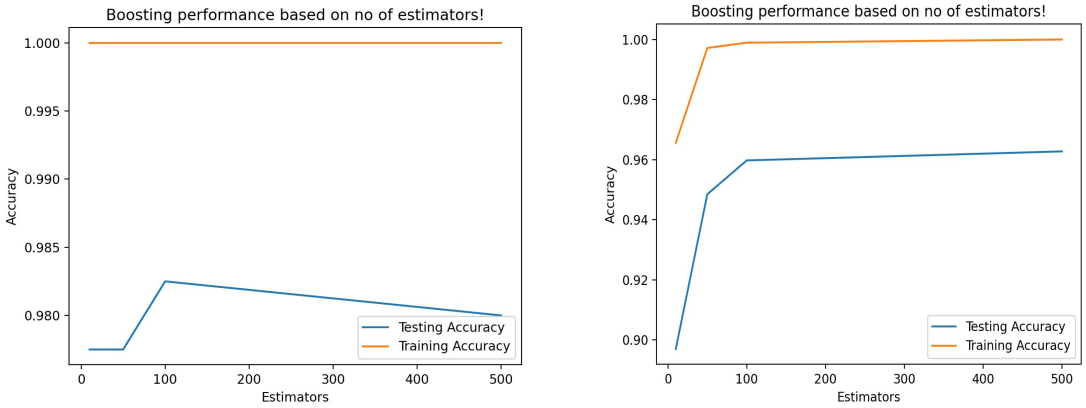
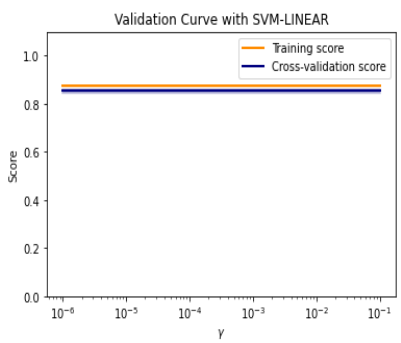


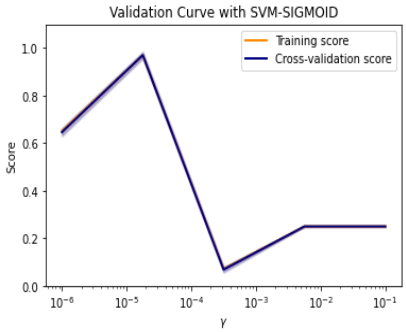
Figure: WILD dataset(Left) , LR dataset(Right)

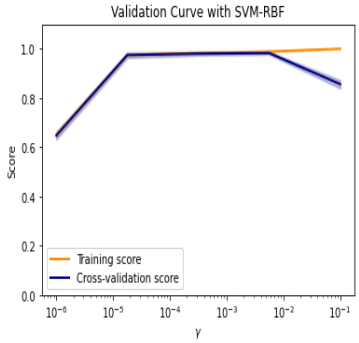
From the WILD dataset and LR dataset, boosting clearly shows high bias towards the training data, but when it comes to the validation/testing accuracy there is a clear difference in accuracy. With WILD one interesting fact is that it performed well with 100 estimators and started depleting beyond 100. On the other hand with LR dataset the accuracy of the validation set has been constant after 100 estimators.
Surprisingly Boosting has really performed well on the accuracy scores for both the models.

SVM:

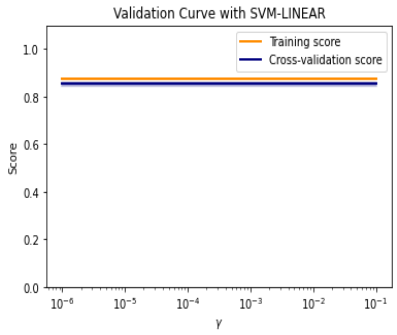
Have implemented SVM using the sklearn package. Have used 3 kernels (‘linear’, ‘rbf’, ‘sigmoid’). Below are plots of 3 different kernels for WILD dataset.

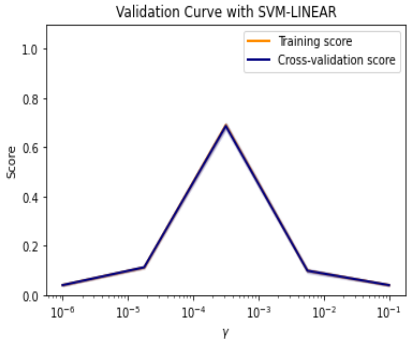


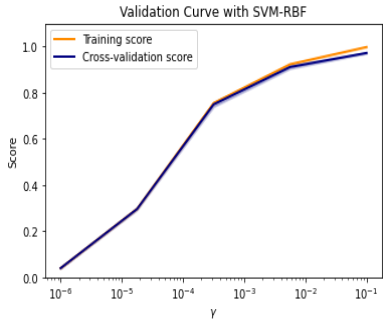




Below are the plots of the same kernels for LR dataset.







From the above plots, for RBF and SIGMOID, we can observe that for low values of gamma the training and CV scores are low which is clearly due to underfitting and for medium values of gamma values both train and CV scores looked close and uniform, which shows the model performing better compared to low gamma values and with more increase in the gamma value the models were getting overfitted(RBF) which was leading to CV decrease in score.

Conclusion:

Overall this assignment has helped try out different ML algorithms with two different datasets and have tuned almost all of the hyper-parameters available to get to the results above.
ANN has out-performed the classification task on both the datasets, compared to all of the other ML algorithms. Below is a table for prediction score on all above algorithms,

Algorithm	LR - Prediction score	WILD - Prediction score	LR-Clock-Time	WILD-Clock-Time
Decision Trees	0.88125	0.976667	4 mins	50 seconds
ANN	0.97323	0.9821134	3 min 47 sec	47 seconds
KNN	0.95251	0.9775	50 seconds	11 seconds
Boosting	0.94556	0.98374	2 min 16 sec	14 seconds
SVM	0.89655	0.976556	12 mins 5 sec	14 seconds