**B551 Assignment 4: Machine learning**

**Classifier 1: K-Nearest Neighbours**

The data consisting of the different parameters used with their accuracy and runtime for different training data sizes is tabulated below: -

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter(K) | Accuracy (%) | Time (in seconds) | Training dataset size |
| 10 | 68.50 | 1.52 | First 1000 images |
| 10 | 70.94 | 6.82 | First 10000 images |
| 10 | 71.36 | 14.4 | Complete |
| 11 | 69.35 | 1.54 | First 1000 images |
| 11 | 71.15 | 5.75 | First 10000 images |
| 11 | 71.89 | 20.77 | Complete |
| 41 | 67.86 | 1.99 | First 1000 images |
| 41 | 69.98 | 4.57 | First 10000 images |
| 41 | 71.04 | 22.27 | Complete |
| 101 | 68.08 | 1.62 | First 1000 images |
| 101 | 70.73 | 6.75 | First 10000 images |
| 101 | 71.04 | 13.84 | Complete |
| 191 | 68.8 | 1.57 | First 1000 images |
| 191 | 70.83 | 4.86 | First 10000 images |
| 191 | 71.38 | 13.48 | Complete |

After running the model for several runs, we see that the highest accuracy is achieved when k is taken to be 11.

I would suggest the client to go with an odd K values (reason being to avoid clash of dominant angle) for their dataset. KNN algorithm is pretty fast and efficient as seen from the above running time and its accuracy.

We observe that as the training size increases, the time required for KNN model to predict increase which is pretty obvious. But there is always a trade-off. Here, with increase in running time, we are also getting some improvement in the accuracy.

Few images which were classified correctly/incorrectly: -

 Correct: 0 Predicted :0

 Correct: 0 Predicted: 270

 Correct: 270 Predicted: 270

**Challenges faced**

We tried 3 approaches to find the nearest neighbours. First, we were trying to find the distance of test image with each and every train image. This consumed a lot of time and usually took 15-20 mis for execution. Then we though directly subtracting the array of test with train. But, even that took a minute or so. To further reduce the time, after announcement on Piazza, used the Scipy cdist function to calculate the distance which is very effective and quick. This reduced the code runtime to a maximum of 20 seconds (worst case scenario).

**Classifier 2: Neural Networks**

**Challenges faced**

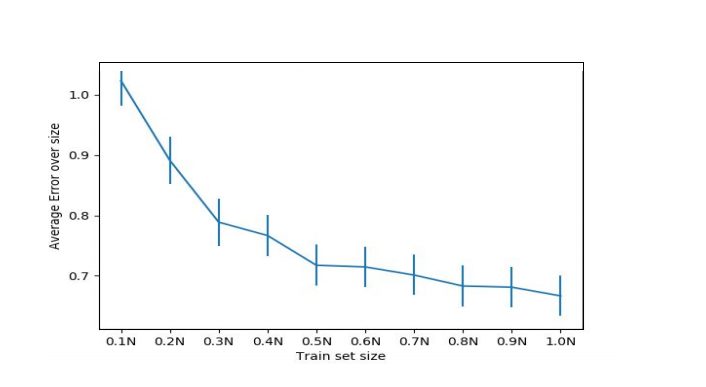
The difficult part of implementing neural net was about making sure the backpropagation works right. By this I do not mean implementing correct algorithm. It was easy to implement the algorithm. However, the problem was getting the underlying problems like

1. Avoid Nan values for loss or gradient
2. Implementing Numerically stable activation functions and loss functions
3. Implementing Numerically stable derivative functions of activations and their loss function
4. Normalizing input data to avoid explosion of logs or exponents or sigmoid, etc.

To tackle problems mentioned above, I looked up on how to numerically tweak the values to keep them mathematically same. This helped the output of these functions to not reach NaN or infinity!

Training the neural net also was a little challenging task as it was difficult to decide the right size of parameters/complexity of the network so that it doesn’t overfit the data. To tackle this, we followed the cross validation technique and selected the model which gave least accuracy on validation set.

**Observations**

Following is the graph for average loss over different sizes of dataset. [](https://i.ibb.co/vwmf07D/Capture.png)

All these losses were recorded after the error was almost constant. For lower dataset size this happened after about 1000 epochs, however for full dataset it needed about 500 epochs only.

This shows that the neural net was able to learn even for the small dataset size however, it was overfitting to the trainset and did not perform better than 30% accuracy. However, for full trainset it reached the lowest loss faster and test accuracy was more than 65%

References

1. <https://google-developers.appspot.com/machine-learning/crash-course/backprop-scroll/>
2. <https://stackabuse.com/creating-a-neural-network-from-scratch-in-python/>

**Classifier 3: Decision Trees**

**Challenges faced:**

The difficult part of implementing decision tree was selection of features and depth of the tree. Because, the deeper the tree is, more complex and computationally expensive it is. Also, to find the split point and calculating entropy, it is very expensive to consider all the 192 features because each element of feature will be compared with all the elements of other features for calculating split point and entropy. To avoid this computation, we used only 20 random features out of 192.

**Flow**:

First, train data and labels are read. To build a decision tree, we find the split point and corresponding feature index. To find the split point, we first consider 20 features, and for every feature entropy is calculated and lowest entropy feature is considered. Using this split point and feature, we divide the tree into left and right node. To find out which label to give to the leaf node, when the data is classified properly, we calculate count for each label and whichever label dominates the most based on probability criteria, we assign that label (i.e. node value) to the leaf node. This process is recursively called to build a decision tree till depth 5. Any further going into the depth, training time will be increased exponentially.

**Observations**

* As the depth of the tree increases, decision trees become more accurate however it also takes more time to train the tree.
* If we train decision tree with very small levels, it won't be able to make accurate results, since we haven't been able to classify it properly as we did not go very deep in tree levels. It will only depend on very small number of features and small number of features does not represent our whole data behaviour resulting in poor predictions.
* Training time of decision tree is around 30 minutes using depth as 5 and considering 20 random features.
* By increasing the features or depth of the tree results in very longer training time.
* Testing time of decision trees is very fast. We are getting accuracy of around 64%.

**Improvements**

* Better logic to calculate split point.
* Selecting smaller training samples based on some unique value criteria to reduce running time which in turn will allow to go into more depth and to use more sample features.
* Using multi-node tree instead of binary tree and calculating split point in a different way.

**References**

1. https://paragmali.me/building-a-decision-tree-classifier/
2. https://wiki.python.org/moin/UsingPickle
3. https://medium.com/@rishabhjain\_22692/decision-trees-it-begins-here-93ff54ef134
4. https://towardsdatascience.com/decision-tree-overview-with-no-maths-66b256281e2b
5. http://www.r2d3.us/visual-intro-to-machine-learning-part-1/