Machine Learning

---------------------------------------------------------------------------------------

# **Problem Statement :**

Create classifiers that decide the correct orientation of a given image

# 1st Classifier - KNN / Nearest :

It is a classification technique in which an instance is classified on the basis of distance from its neighbors. KNN is an instance-based classifier that classifies the object according to the similarity between the 'K' similar items.

## Training Model

**/orient.py train train-data.txt nearest\_model.txt nearest**

We just copy whole training data into a model\_file.

## Testing Model

**/orient.py test test-data.txt nearest\_model.txt nearest**

After the model file is created, we can use that file for classifications of the test inputs. Classification is done by finding the Euclidean distance between the rgb pixel vectors of test inputs and all the training data. The K minimum distances are taken. For this 'K' minimum values the actual orientation from the training data is extracted. Among this orientation the maximum occurring value is used as a classifier for the test instance(image). If the predicted value of orientation is same as the actual one for test image, then we increment a counter which is used to calculate the accuracy at the end when all the test images are classified.

### Flow of the code :

* Generate 3 vectors image\_names, orientation and rgb\_vector for both test and train\_model files
* Value of K is defined globally
* For each test image find the Euclidean distance ( We have used np.linalg.norm function to calculate the distance) between test-image and every train-image. 4) Sort the values in ascending order.
* Pick K small values and find its orientation from train orientation array using the index function.
* Pick the maximum occurring orientation value and assign that to the test image.

## Analysis

Time for training the data is negligible as we just have to copy the train data in to a file. For testing the minimum time 270 seconds i.e around 4 minutes. We tried different values of K to find the best accuracy results. For k = 25 it gave the maximum accuracy of 71.36 %.

|  |  |  |
| --- | --- | --- |
| K | Accuracy | Running Time |
| 5 | 70.0954401 | 261.6235151 |
| 10 | 69.6712619 | 262.3811409 |
| 12 | 70.4135737 | 269.398294 |
| 15 | 70.5196182 | 272.8769982 |
| 20 | 70.5196182 | 278.4933591 |
| 25 | 71.3679745 | 279.614105 |
| 30 | 70.4135737 | 279.9936941 |

## Problems Faced :

1. First we calculated the distance manually by subtracting the rgb values of train-images from test-image using for-loop. This took really long time for test\_model to run. After using numpy linear algebra library the time for execution dropped giving decent execution time of 4 minutes.

# 2nd Classifier Decision Tree

Decision tree is a type of supervised learning algorithm. In this technique, we split the population or sample into two or more homogeneous sets based on most significant splitter in input variables. Decision trees use multiple algorithms to decide to split a node in two or more sub-nodes

## Algorithm Flow:

We have used Gini algorithm to split.

* Find gini index. The gini index is found using formula

1 — P^2(Target=0) — P^2(Target=1)

* Find information gain
* Split data considering highest gain or minimum gini index
* Build decision tree model by using recursion to create nodes
* For each node found which column is giving highest information gain. Appended that column index as a dictionary key

Constraints:

* Due to high computational cost the depth of tree was limited to 5.
* Increased 0:255 value in steps of 3, to reduce iteration

## Problem Faced:

* For larger depth the computational cost is high
* Finding best split value

## Disadvantages:

The depth is limited by hard-coding. It will check for the best split instantaneously and move forward until one of the specified stopping condition i.e depth=5 is reached. This is a greedy approach. Instead pruning could have been used.

## Commands:

### Training –

**/orient.py train train-data.txt nnet\_model.txt tree**

### Testing –

**/orient.py test test-data.txt tree\_model.txt tree**

### Analysis –

Decision tree gives an accuracy of 59.06.

# 3rd Classifier NNet – Neural Network

Neural network classifier consists of neurons that are arranged in layers. This neuron convert input in to output with the help of some non-linear functions. Generally neural network is defined to be feed-forward networks. There are 2 parts in the neural net classifier Feed-Forward propagation and Backpropagation.

Network Architecture – 3 Layers

1. Input Layer
2. Hidden Layer
3. Output Layer

Hidden Nodes – 100

Number of Iterations – 20 ( We have kept this configuration considering the accuracy and execution time of program)

## Training Model

**/orient.py train train-data.txt nnet\_model.npz nnet**

We have used Stochastic Gradient Descent and Backpropagation to train the model. The whole dataset is used in one go. This was decided as the dataset was not huge i.e only 36000+ rows are there in training file. The neural network was run on 20 times to train and learn. 20 was decided after checking the accuracy for the other number of iterations. The accuracy didn’t change that much if we increase the number of iterations only the running time for program increased.

For the first weight and biased we have set those as random values using np.random.

Each training item is then propagated forward and backward to set and update the values of weight and bias for the 3 layers with the activation function. To normalize the values for RGB vectors we divided it by maximum value of the pixels i.e 255.

Learning Rate for the backpropagation is kept constant at 0.04.

### Activation Function –

As mentioned above the neuron convert the input into output using non-linear functions. Those functions are called as Activation functions. There are various activation functions which can be used in neural networks. We used Sigmoid function after testing all the other functions such as tanh, softmax and relu.

1. Sigmoid Function :

def sigmoid\_function(x):  
 return 1 / (1 + np.exp(-x))

1. Tanh Function :

def tanh\_function(x):  
 return np.tanh(x)

1. Softmax Function :

def softmax\_function(x):  
 expo = np.exp(x)  
 expo\_sum = np.sum(np.exp(x))  
 return expo / expo\_sum

1. Relu Function :

def relu\_function(x):  
 return np.maximum(0, x)

Relu function gave very low accuracy of 25% that’s why we discarded the option.

Softmax activation function gave us float value overflow due to which it gave same values of weight. i.e 1

Tanh function gave similar values compared to Sigmoid but the accuracy was better when used Sigmoid function.

Finally after all the iterations the values of weight and bias for the layers i.e input-hidden and hidden-output are saved in a model file. This file must be a .npz file as we are saving numpy arrays in that file.

## Testing

**/orient.py test test-data.txt nnet\_model.npz nnet**

* From the model file the 4 arrays are loaded and used to predict the orientation of the test image.
* We went with the 4 output configuration for orientation i.e

For 0 degree -> [1,0,0,0]

90 degree -> [0,1,0,0] etc

* Each output i.e maximum predicted value for test image is classified as the orientation of that test image.
* For eg- max [0.23348966 0.32541838 0.06710359 0.05635076] i.e 0.3254 which is at 1st index i.e the test image will be classified as 90degree oriented.
* Count is incremented when an image is classified correctly and that count is used to calculate the accuracy of the network.

## Analysis

There are various combination of the parameters that a neural network can have.

We tried varying the iteration count and also by changing the values of learning rate which can be found below in the table.

Table 1 - Varying Hidden Nodes, Learning Rate and No of Iterations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No of Iterations | Hidden Nodes | Learning Rate | Time(seconds) | Accuracy |
| 20 | 100 | 0.01 | 71.64661407 | 69.14103924 |
| 20 | 100 | 0.02 | 70.52437997 | 70.09544008 |
| 20 | 100 | 0.03 | 68.53817821 | 69.56521739 |
| 20 | 100 | 0.04 | 71.17904687 | 73.59490986 |
| 20 | 100 | 0.045 | 69.20307112 | 72.21633086 |
|  |  |  |  |  |
| 40 | 150 | 0.01 | 170.546978 | 72.42841994 |
| 40 | 150 | 0.02 | 178.2052028 | 73.80699894 |
| 40 | 150 | 0.03 | 172.6744001 | 72.95864263 |
| 40 | 150 | 0.04 | 171.117038 | 72.85259809 |
| 40 | 150 | 0.045 | 169.896544 | 72.95864263 |

Considering the execution time and accuracy we decided to go with the following config

Learning rate – 0.04

Iteration – 20

Hidden nodes – 100

Best accuracy – Marked in blue one. i.e 73.8069

## Problems Faced

1. Deciding the Activation function. Explained in activation function part.
2. Selecting the learning rate.
3. We went till 100 iterations but the execution time for training the model was much more and with no significant change in the accuracy. Hence kept the iteration value at 20 in which execution time is very low.

# 4th Best Classifier

We are using Neural Network Classifier as our best classifier with the iteration count as 20, hidden node as 100 and learning rate as 0.045.

After comparing all the 3 models Neural Network was decided as best classifier as it takes much less time with the greater accuracy.