CS440: MP6

Perceptron - Statement of extra credit

Rahul Kunji - 26 November 2018

The Baseline Perceptron was getting me an accuracy of around 80%

```
Rahuls-MacBook-Air:CS440MP6 rahulkunji$ python3 mp6.py --
dataset mp6_data.dms
Training Done in 3.0349559783935547 seconds.
End to End Done in 3.051933765411377 seconds.
Accuracy: 0.8024
F1-Score: 0.8290657439446366
Precision: 0.8454481298517996
Recall: 0.813306177868296
```

I tried 2 different strategies to make a better version of this. Unfortunately, both methods could not beat the simple perceptron (for different reasons). In the following passages I will explain the two methods.

1) Decision Trees.

The Decision Tree was based on the CART (Classification and Regression Trees) algorithm and I followed the guide at https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python/ for the same.

However, this algorithm is unable to handle the huge dimensions and size of our training data. I verified that the algorithm gets really good accuracy (~97%) on a smaller dataset (http://archive.ics.uci.edu/ml/machine-learning-databases/00267/) but it takes way too long for it to complete on our provides dataset. So I had to reduce the size of our training data. For this reason, I chose the first 50, last 50 and random 50 training samples from the provided list. However, this adversely effects the accuracy.

```
Rahuls-MacBook-Air:CS440MP6 rahulkunji$ python3 mp6.py --
dataset mp6_data.dms --extra
End to End Done in 19.185397148132324 seconds.
Accuracy: 0.5948
F1-Score: 0.7387155016765542
Precision: 0.5956738768718802
Recall: 0.9721656483367278
```

One can note that the recall is really high with this approach usually as it tends to err on the side of a positive classification. This might be due to how the training data is split.

2) A simplified multi-perceptron system

The idea for this approach was really simple. I have included it in the classifyECalt() method.

Firstly I train 5 different perceptrons. The way I train them is by them looking at different combination of images from the training set. for example the third perceptron skips looking at every third image.

After this layer of 5 perceptrons, there is a secondary layer of 5 processing nodes which simply looks at different combinations of size 3 from these 5 (first3, last 3, random 3 etc) and tries to combine the 3 perceptron prediction inputs into a single value.

Finally the single output processing node, combines the inputs from these 5 intermediate nodes into a single value prediction for an image.

This approach did not get me results which were much better than the single perceptron. I think that maybe with more hidden layers and a better overall design/ training data split, this could have produced better results, but performance and time are a bottleneck.

```
Rahuls-MacBook-Air:CS440MP6 rahulkunji$ python3 mp6.py --
dataset mp6_data.dms --extra
Training Done in 11.161638975143433 seconds.
End to End Done in 11.333199262619019 seconds.
Accuracy: 0.7992
F1-Score: 0.8236120871398454
Precision: 0.853605243991260
Recall: 0.7956551255940258
```