Brandon TerLouw & Warren Cho

Milestone B Progress Report

1. A revolutionary classifier
2. Brandon TerLouw – Began development of neural network
   Warren Cho – Began development of decision tree
3. The classifier is supposed to use data to predict the class of a specific instance
4. Neural Network – A neural network is one way to implement a classifier. It can be made up of multiple layers of perceptrons. A perceptron is a device that takes an array of weights, and an array of input values, multiplies them and sums all the values. It then either outputs a 1 if the summation is above a threshold, and 0 if it is below. Another version of a perceptron uses a sigmoid function and returns a value between 0 and 1. A neural network can be trained by changing the weights until the error is close to a minimum. Once a neural network has been trained, ideally it will be able to take data and assign it to a class.
   Random Forest – A random forest of decision trees is another way to implement a classifier. Using trees that handle all of the various features of objects, a random sample subset of features of an object can be taken and compared against these trees. This results in a prediction of an outcome and the class with the most number of votes/decisions is the final prediction of the random forest. Random forests are an ensemble learning method and is fairly accurate via bagging (bootstrap aggregating) by using randomness (entropy) to its advantage.
5. Neural Network:
   After training a single perceptron to work with 2 classes, we were able to get the neural network to identify if a plant is a Setosa or a Versicolor. The data was from the Iris Data Set, with the third plant omitted. Below is a log, our perceptron worked 100% of the time due to the data being linearly separable.

```
  Testing training data:
We compute 0, actually: 0
We compute 0, actually: 0
We compute 1, actually: 1
We compute 0, actually: 0
We compute 0, actually: 0
We compute 0, actually: 0
We compute 1, actually: 1
We compute 0, actually: 0
We compute 1, actually: 1
We compute 1, actually: 1
We compute 0, actually: 0
We compute 0, actually: 0
We compute 1, actually: 1
We compute 0, actually: 0
We compute 0, actually: 0
We compute 1, actually: 1
We compute 1, actually: 1
We compute 0, actually: 0
We compute 1, actually: 1
We compute 0, actually: 0
```

```
We compute 1, actually: 1
We compute 0, actually: 0
We compute 1, actually: 1
We compute 0, actually: 0
We compute 0, actually: 0
We compute 0, actually: 0
We compute 1, actually: 1
We compute 1, actually: 1
We compute 1, actually: 1
We compute 0, actually: 0
We compute 1, actually: 1
We compute 1, actually: 1
We compute 1, actually: 1
We compute 1, actually: 1
We compute 0, actually: 0
We compute 1, actually: 1
We compute 0, actually: 0
We compute 0, actually: 0
We compute 1, actually: 1
We compute 1, actually: 1
We compute 0, actually: 0
We compute 1, actually: 1
We compute 1, actually: 1
We compute 0, actually: 0
We compute 1, actually: 1
We compute 0, actually: 0
We compute 1, actually: 1
We compute 0, actually: 0
We compute 1, actually: 1
We compute 0, actually: 0
We compute 0, actually: 0
We compute 1, actually: 1
We compute 1, actually: 1
We compute 0, actually: 0
We compute 0, actually: 0
We compute 0, actually: 0
We compute 1, actually: 1
We compute 0, actually: 0
We compute 0, actually: 0
We compute 0, actually: 0
We compute 1, actually: 1
We compute 0, actually: 0
We compute 1, actually: 1
We compute 1, actually: 1
We compute 1, actually: 1
We compute 1, actually: 1
We compute 0, actually: 0
We compute 1, actually: 1
We compute 0, actually: 0
We compute 0, actually: 0
We compute 1, actually: 1
We compute 1, actually: 1
We compute 1, actually: 1
We compute 0, actually: 0
We compute 0, actually: 0
```

```
We compute 0, actually: 0
We compute 1, actually: 1
We compute 1, actually: 1
We compute 1, actually: 1
We compute 1, actually: 1
We compute 0, actually: 0
We compute 1, actually: 1
We compute 0, actually: 0
We compute 1, actually: 1
We compute 0, actually: 0
We compute 1, actually: 1
We compute 1, actually: 1
We compute 0, actually: 0
We compute 1, actually: 1
We compute 1, actually: 1
We compute 1, actually: 1
We compute 0, actually: 0
We compute 0, actually: 0
We compute 1, actually: 1
We compute 0, actually: 0
We compute 0, actually: 0
We compute 0, actually: 0
We compute 0, actually: 0
We compute 1, actually: 1
 Testing never seen data:
We compute 1, actually: 1
```

Random Forest: No test results yet available.

6. Neural Network – To run demo you must run 'NeuralNetCall.py". It must be in the same folder as NeuralNet.py and irisData.csv
   Random Forest – No functional demo yet available.
7. Neural Network – Below is a screen shot of the code used for a perceptron. It takes weights, a threshold, epsilon and data. It then returns either a 1 or 0.

```
29 def BinaryResponse(weights, characteristics, epsilon,theta):
30     sum = 0
31     for n in range(1,len(weights)):
32         sum += weights[n]*characteristics[n]
33     if sum > (theta[0] - epsilon): return 1
34     else: return 0
```

Random Forest – below is a snippit of code used for the class prediction using decision trees within the random forest. This returns the prediction with the highest occurrence (mode).

```
def predict(self, x):
    n_samp = x.shape[0]
    n_tree = len(self.forest)
    predict = np.empty([n_tree, n_samp])
    for i in range(n_tree): predict[i] = self.forest[i].predict(x)
    return mode(predict)[0][0]
```

8. Brandon TerLouw – So far I have learned much about how a neural network operates. I learned what it takes to program and train a simple perceptron in order for it to classify linear separable data. I have read much on backpropagation and believe I am able to combine it with my simple perceptron to create a two layer neural network. I also learned how to import data from a csv file.
   Warren Cho – I have learned that a random forest learning method is that high variance is not always a bad thing – in this case it helps immensely. The randomness of the random forest allows for various evaluations which can be controlled by calibrating the split point to our liking. I also learned that it's fairly easy to adjust the model for regression vs classification – although we are just using it for classification.
9. Neural Network – Next we need to turn our simple perceptron into a multilayer neural network so it can classify data into more than 2 categories.
   Random Forest – Ideally we would add better support/implementation of bagging for increased accuracy.
10. Neural Network – For the most part I used information on slide to program and train perceptron.
    This is the article I read to better understand math behind back propogation:
    https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

    Random Forest – This website provided very helpful high-level explanations and pseudocode for the random forest algorithm. *How the Random Forest Algorithm Works in Machine Learning* by Saimadhu Polamuri (http://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/).

    This website was also very helpful for the same reasons as the previous link but provided a more detailed case example as well as some code examples. *Random Forest Python* by Mahsa Hassakashi (https://www.codeproject.com/Articles/1197167/Random-Forest-Python).