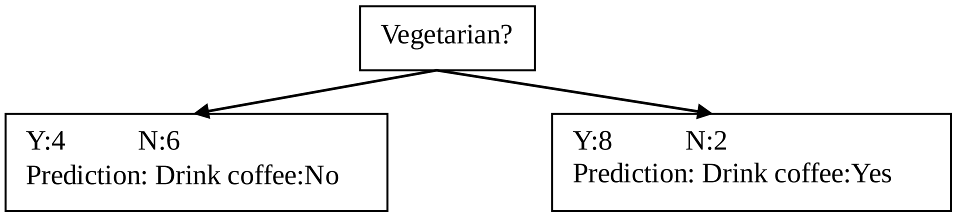
Zhuo Liu
CSCI-630-02 Foundations of FIS
Homework4

1 (15 pts) Consider using Adaboost with decision stumps (using information gain) for the decision tree problem of HW 3.5.

a. What weight would each example (the **w**s in the Adaboost pseudocode) have initially?

The initial weight of each example is $\frac{1}{12+8}=\frac{1}{20}$

b. What would Adaboost select as the first stump? (Hint: this should be very easy...) How would this stump make predictions (that is, what prediction would it make, based on the value of what attribute?



The prediction is based on the number of Y and N, for instance, the number of Y is 4, the number of N is 6, 6 is greater than 4. So our prediction is

Drink coffee:No

c. After the first iteration, what would the example weights be for each example that the first stump got right, and for each example that the first stump got wrong?

$$error=\frac{2+4}{20}=\frac{3}{10} \qquad w=\frac{1}{20}*\frac{error}{1-error}=\frac{3}{140}$$

$$\frac{3}{140}*(6+8)+\frac{1}{20}*(2+4)=\frac{3}{5}$$

The first stump got right:

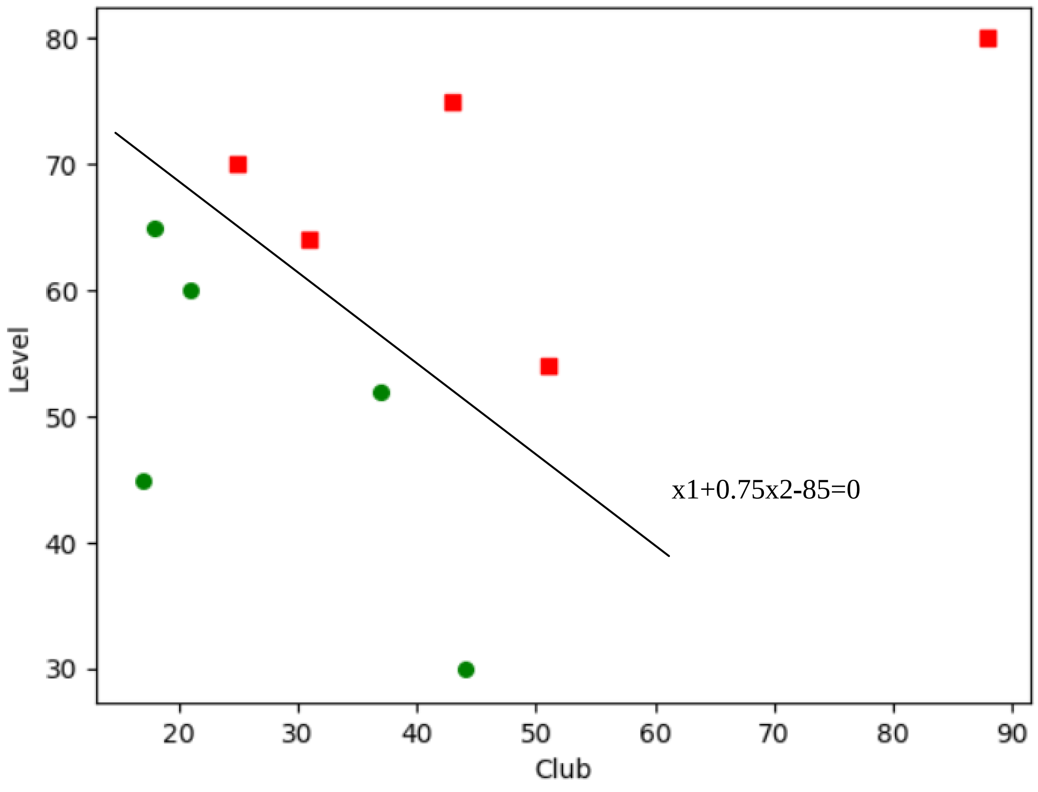
$$w=\frac{3}{140}*\frac{5}{3}=\frac{1}{28}$$

The first stump got wrong:

$$w=\frac{1}{20}*\frac{5}{3}=\frac{1}{12}$$

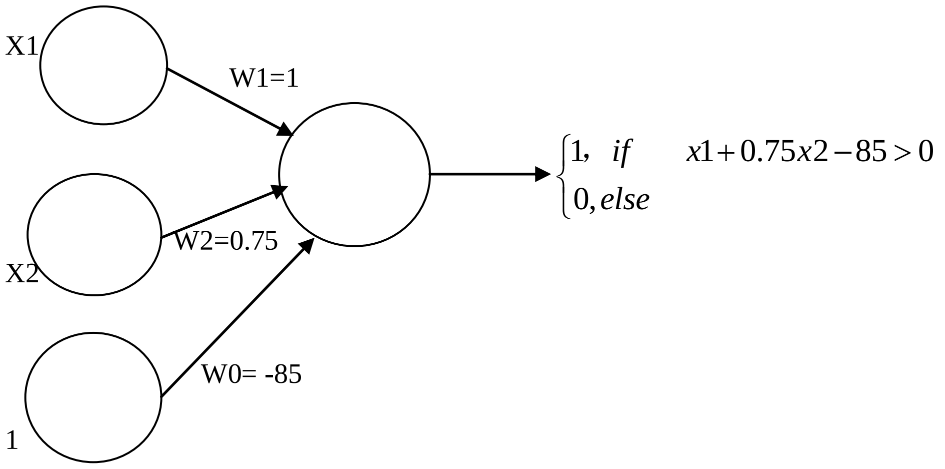
2 (15 pts) Is the following data set linearly separable into the two classes "Fun" and "Not fun"? Show why or why not.

| People in club? | Decibel level? | Fun or not fun? |
|---|---|---|
| 25 | 70 | F |
| 43 | 75 | F |
| 17 | 45 | NF |
| 88 | 80 | F |
| 37 | 52 | NF |
| 31 | 64 | F |
| 44 | 30 | NF |
| 21 | 60 | NF |
| 51 | 54 | F |
| 18 | 65 | NF |



From the figure shown above, the data set is linearly separable.

3 (10 pts) Based on your answer to the previous question, create (presumably by hand) a network of one or more perceptrons with step thresholds that can classify the given data (you should have one output value representing "Fun" as 1 and "Not fun" as 0).



$$\begin{cases} 1, & if \quad x1 + 0.75x2 - 85 > 0 \\ 0, & else \end{cases}$$

4 (15 pts) Consider a neural network with 100 input nodes, one hidden layer of 100 nodes fully connected to the input, and one output. How many total weights will be learned in this system? Compare this network to one with two fully-connected hidden layers of k nodes each. What value of k gives a network with approximately the same number of weights as the first network?

1) neural network with 100 input nodes, one hidden layer of 100 nodes fully connected to the input, and one output
   Total weights=100*100+100*1=10100

2) neural network with two fully-connected hidden layers of k nodes each
   Total weights=100*k+k*k+k*1
   Let 100*k+k*k+k=10100
   k≈62

5 (20 pts) Over the last couple of years, I have been investigating the idea that the gas mileage of my (hybrid) car is dependent on temperature. Since I take the same route to work most days, and there is little traffic to change the character of the drive itself, this seemed like a good opportunity to test the hypothesis. Here is the data I collected two years ago.

For this problem, we are only concerned with the columns "Temperature", "MPG", and "Notes". The string "sch" in the Notes column are days when I had just taken one of my kids to school, so (presumably) the engine would be a little bit warm before driving to RIT, but the data in those rows is still just from my home to RIT. It is up to you to decide what to do with those rows, and you should explain your decision.

a. What is the best-fit linear function to explain the relationship between Temperature and MPG? You should use the least-squares error, and you are free to use the closed-form solution in this case if you wish.

   I choose to use the examples with "sch" which is in the column.

   Engine temperature can also be a key factor which can affect the calculation, sometimes it can influence the MPG.

   The best-fit linear function should be y=kx+b

   y represents MPG and x represents Temperature

   The least-squares error is shown as followed:

$$S=\sum_{i=1}^{n}[y_i-(kx_i+b)]^2$$

   We can let the gradient or partial derivative equal to 0 to get the value of k and b, such that

$$\frac{\partial S}{\partial a}=-2\sum_{i=1}^{n}x_i[y_i-(kx_i+b)]=0$$

$$\frac{\partial S}{\partial b}=-2\sum_{i=1}^{n}[y_i-(kx_i+b)]=0$$

   Since we have data for each $x_i$ and $y_i$, according to these values we can get k and b easily.

   k$\approx$0.323 , b$\approx$40.65, so the best-fit linear function should be $y=0.323x+40.65$

b. Here is some data collected last year. Compute the prediction error on these data points using the line created in part a, and then compute the error when using these data themselves to define the best fit line. Do you get what you expect?

The linear function of 2017 data should be  y=0.323x+40.65

The prediction error of 2017 data should be :

$$error = \sum_{i=1}^{62}[y_i - (kx_i + b)]^2 = \sum_{i=1}^{62}[y_i - (0.323x_i + 40.65)]^2 = 254.76$$

The best fit linear function of 2018 data should be  y=0.333x+41

The error should be :

$$error = \sum_{i=1}^{26}[y_i - (kx_i + b)]^2 = \sum_{i=1}^{26}[y_i - (0.33x_i + 41)]^2 = 125.79$$

As the result shown above, we can see the error is much lower than the prediction error, the reason is it uses 2018 data themselves to give the linear function.

c. Consider using the 2018 data to predict the 2017 data. Are there any issues that you would foresee with this? (aside from the time-travel issue, let's assume that the actual years were swapped.)

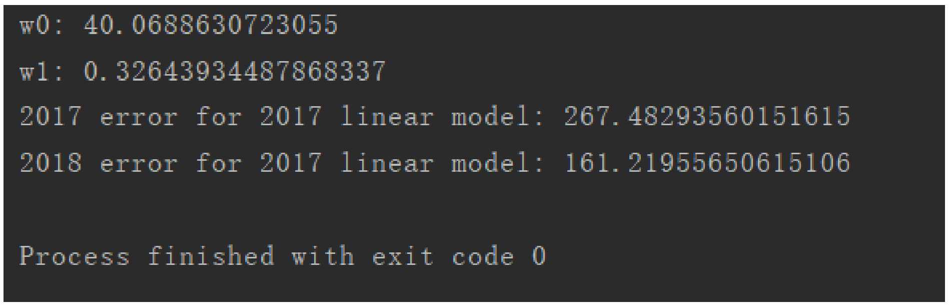
Since the size of two different sets are different, so the this may affect the result.

6 (25 pts) Now let us consider a slightly more complex model for the data of the previous problem. In particular, we will consider a logarithmic function to approximate the data. For this, you will need to implement a gradient descent approach. You may wish to start by implementing gradient descent on a linear function and confirming that it converges to something close to your answer for the previous question.

Recall that if your model is y' = w1x + w0, and you use least-square error, the error function per data point is (y-y')^2 [ or equivalently (y - (w1x + w0))^2 ] and so its gradients are 2*(y-y')*x for w1 and 2*(y-y') for w0. If your model adds a logarithmic term w2*ln(x+w3), there will be an additional gradient term for the w2 weight, 2*(y-y')*ln(x+w3) and for the w3 weight, 2*(y-y')*w2*1/(x+w3). With these additional parameters you may have to be more careful about choosing your initial weights.

As with the previous problem, use the 2017 data as the training data, and measure both its error and the error on the 2018 data set using the 2017 model. How do these compare to the errors for the linear model?

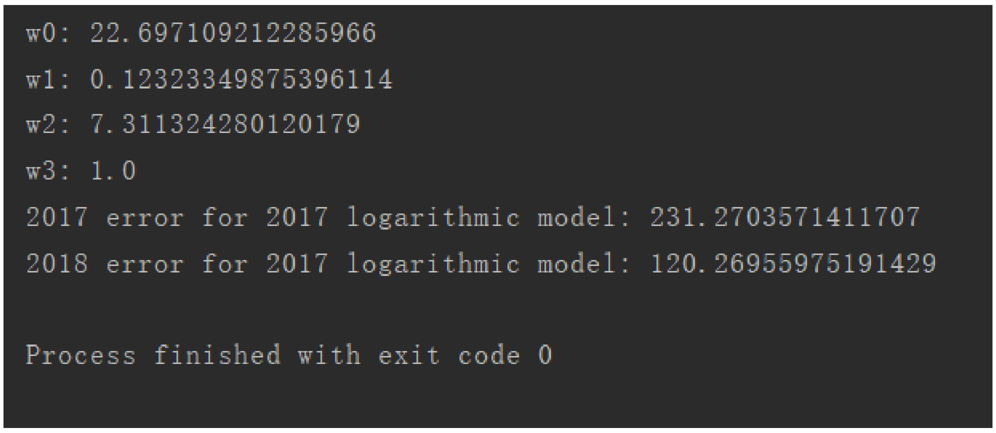
Submit your code as well as your results for these models



```
w0: 40.0688630723055
w1: 0.32643934487868337
2017 error for 2017 linear model: 267.48293560151615
2018 error for 2017 linear model: 161.21955650615106


Process finished with exit code 0
```

As the figure shown above, we can see with 10000 iterations and learning rate 0.0001, The 2018 error for 2017 linear model is less than the 2017 error for 2017 linear model.



```
w0: 22.697109212285966
w1: 0.12323349875396114
w2: 7.311324280120179
w3: 1.0
2017 error for 2017 logarithmic model: 231.2703571411707
2018 error for 2017 logarithmic model: 120.26955975191429

Process finished with exit code 0
```

As the figure shown above, we can see with 10000 iterations and learning rate 0.0001,
The 2018 error for 2017 logarithmic model is less than the 2017 error for 2017 logarithmic model.

In conclusion, we can see overall logarithmic model error is less than overall linear model.
Using logarithmic model is proven to be much better.