1 Linear Classifiers and Boolean Functions

1. Yes it is Linearly Separable. LTU: $x_1 - x_2 + x_3 \geq 0$
2. Yes it is Linearly Separable. LTU: $x_1 - 2x_2 + x_3 - 2 \geq 0$
3. Yes it is Linearly Separable. LTU: $x_1 - 2x_2 + 3x_3 - 1 \geq 0$
4. Not Linearly Separable.
5. Yes it is Linearly Separable. LTU: $-x_1 + x_2 - x_3 - 1 \geq 0$

2 Mistake Bound Model of Learning

(Part 1)

A. The concept class is from 1 to 80, so the size is **80.**

B. First off: $|x_1{}^t| \leq L \wedge |x_2{}^t| \leq L$
$0 \leq L - |x_1{}^t| \wedge 0 \leq L - |x_2{}^t|$
$\text{sgn}(L - |x_1{}^t|) + \text{sgn}(L - |x_2{}^t|) \geq 2$
this is equal to the true label: $\text{sgn}(\text{sgn}(L - |x_1{}^t|) + \text{sgn}(L - |x_2{}^t|) - 2)$,
so:
y^t * sgn(sgn(L - $|x_1{}^t|$) + sgn(L - $|x_2{}^t|$) - 2) < 0

C. To update L, I will first determine if there is an error, then you will initialize a Count to be equal to L. If there is an error, you will first set Count equal to itself divided by 2. If the predicted label is +1, you will decrement L by count, but if the predicted label is -1, you will increment L by Count. Then you will iterate through the same thing, and keep dividing the Count by two before you decrement/increment by Count. You will continue this process until you do not have an error anymore. This is similar to a binary search.

D. The maximum number of mistakes that the following algorithm can make for this example is log(80) which is 7. For any other concept class, the maximum number of mistakes it will make is log(|C|).

```
main ()
    min = 1
    max = 80
    x1 = the x1 value
    x2 = the x2 value
    L = roundUp((max + min) / 2)
    count = L
    True_Label = (the acutal label of the x1, x2, and True_L)
    Actual_L  = Funct(L, count, x1, x2)

Funct (L, count, x1, x2, True_Label)
    Label_guess = CalculateLabel(x1, x2, L)
    error = (True_Label * sgn(sgn(L-|x1|) + sgn(L-|x2|) - 2) < 0)
    if(error)
        count = RoundDown(count / 2)
        if(guess = +1)
            L = L - count
            return Funct (L, count, x1, x2)
        else(guess = -1)
            L = L + count
            return Funct(L, count, x1, x2)
    else
        return L
```

(Part 2)

M is the total number of perfect experts, so we can initially set M = $|C_n|$ because initially in class we set it equal to 1 because we only had 1 perfect expert (in class), but in this case we have M perfect experts.

M = $|C_n| < \frac{1}{2}|c_{n-1}|$

$< \frac{1}{2} * \frac{1}{2}|c_{n-2}|$

$<$:

$< \frac{1}{2^n}|c_0| = \frac{1}{2^n}|C|$

M < $\frac{1}{2^n}|C|$

$2^n * M < |C|$

$2^n < \frac{|C|}{M}$ It was given that |C|, the concept class, is equal to N elements, so you can replace |C| with N:

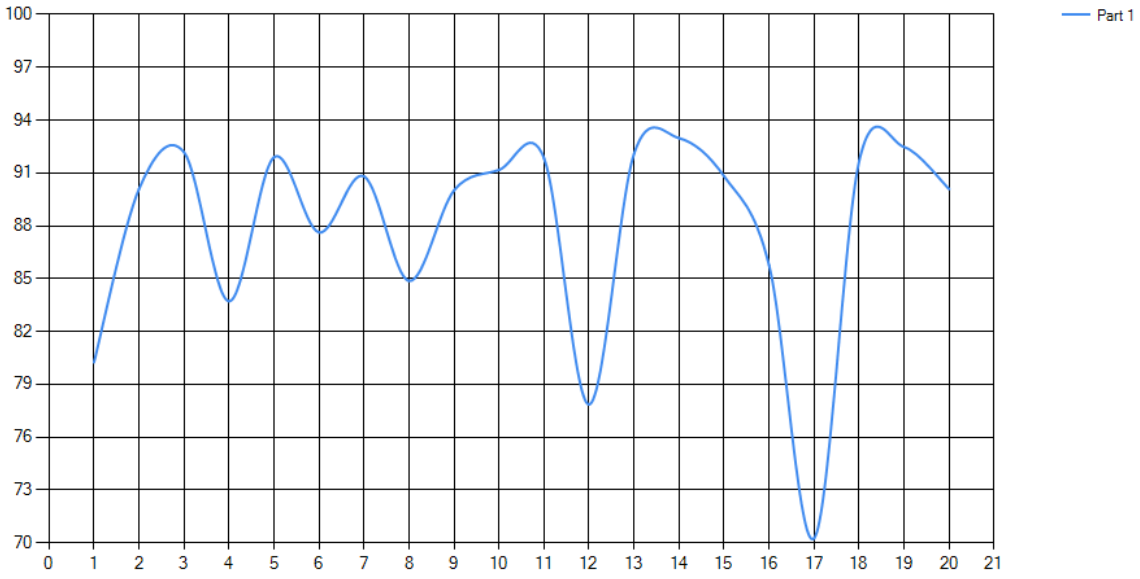
$\log_2 2^n < \log_2 \frac{N}{M}$ Take the log of both sides

$n < \log_2 \frac{N}{M}$ n is the total number of mistakes that happened

At this point the most mistakes that can happen is $\log_2 \frac{N}{M}$ because n is less that $\log_2 \frac{N}{M}$

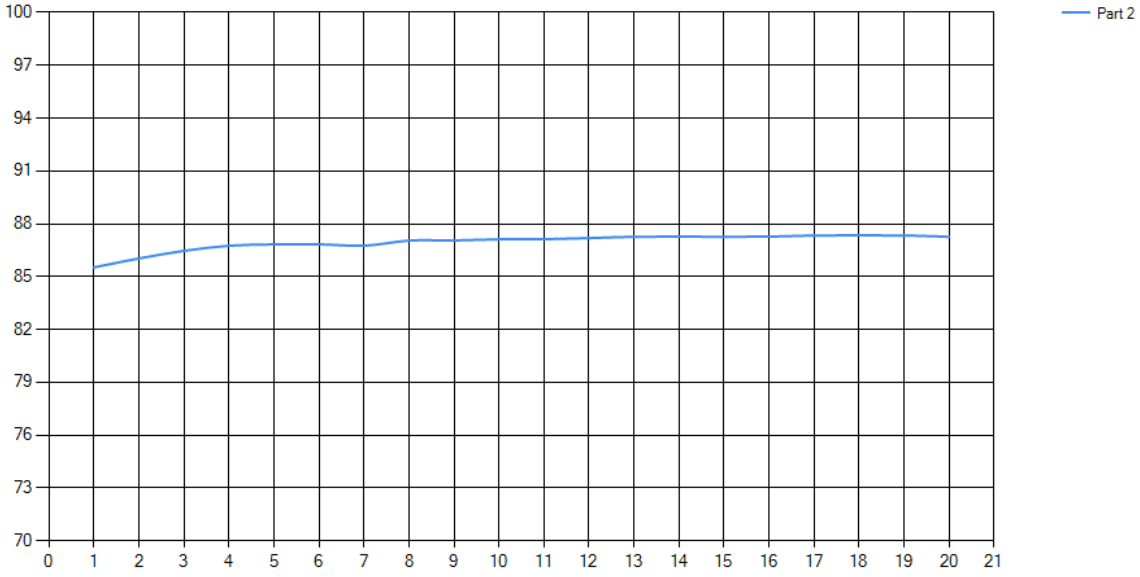
3 The Perceptron Algorithm and its Variants

1. I decided to use C# in this programming assignment just because I am most comfortable with C#. For my vectors, I represented them as an array of double. My bias was a double. I then created a WeightBias object that stored the weight array, the bias, and a counter for the total updates. When I stored each of the accuracy's with their associated weight and bias, I stored them in a Dictionary where the key was the epoch number and the value was an AccuracyWB object which held the WeightBias object and the accuracy. I then calculated the largest accuracy from Dictionary and used its WeightBias to test my data. Each line in the file was stored as an Entry that had the sign and the vector of that line. I then stored each Entry in a List of Entry. While I am doing my algorithm, I am looping through each item in the List of Entry.
2. The Majority Baseline Accuracy for the Test set is: **57.308%**
   The Majority Baseline Accuracy for the Development set is: **54.920%**
3. The following is the report for each part (the report is also ran with the code):
   Part 1
   a. The best hyper-parameter is: **0.01 (Learning Rate)**
   b. The cross validation accuracy for the best hyper-parameter is: **92.908%**
   c. Total number of updates for the best weight and bias: **9864**
   d. Development Set accuracy: **92.981%**
   e. Test Set accuracy: **91.027%**
   f. Graph:



Part 2

a. The best hyper-parameter is: **1 (Learning Rate)**
b. The cross validation accuracy for the best hyper-parameter is: **91.883%**
c. Total number of updates for the best weight and bias: **20280**
d. Development Set accuracy: **87.337%**
e. Test Set accuracy: **88.35%**
f. Graph:

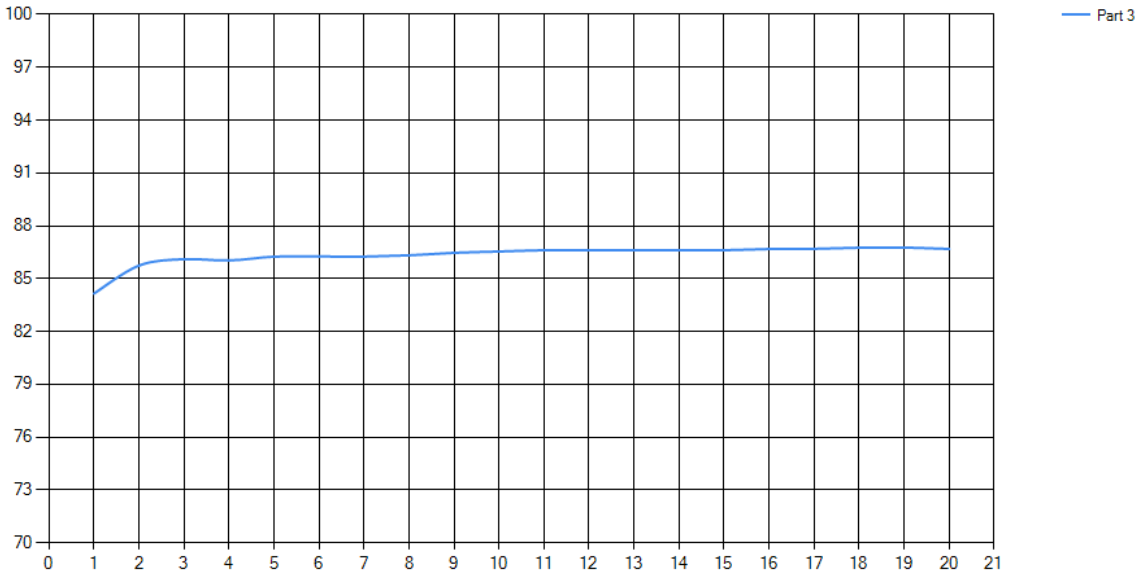


Part 3

a. The best hyper-parameter combinations are: **1 (Learning Rate), 0.01 (Margin)**
b. The cross validation accuracy for the best hyper-parameter is: **90.328%**
c. Total number of updates for the best weight and bias: **20196**
d. Development Set accuracy: **86.758%**

e. Test Set accuracy: **88.061%**

f. Graph:



Part 4

a. The best hyper-parameter is: **0.01 (Learning Rate)**

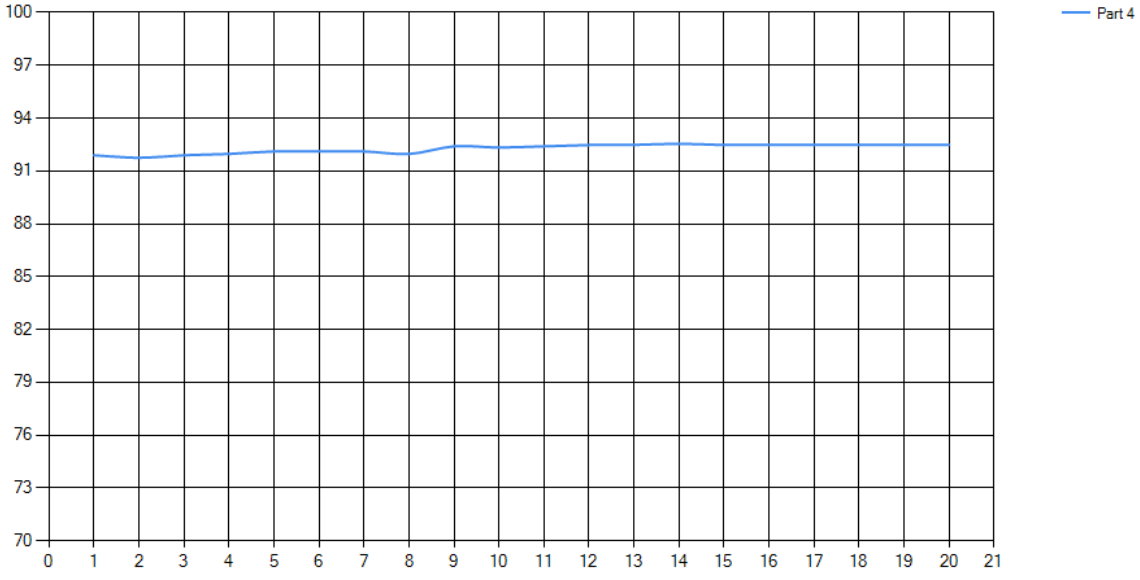
b. The cross validation accuracy for the best hyper-parameter is: **94.042%**

c. Total number of updates for the best weight and bias: **13849**

d. Development Set accuracy: **92.547%**

e. Test Set accuracy: **93.054%**

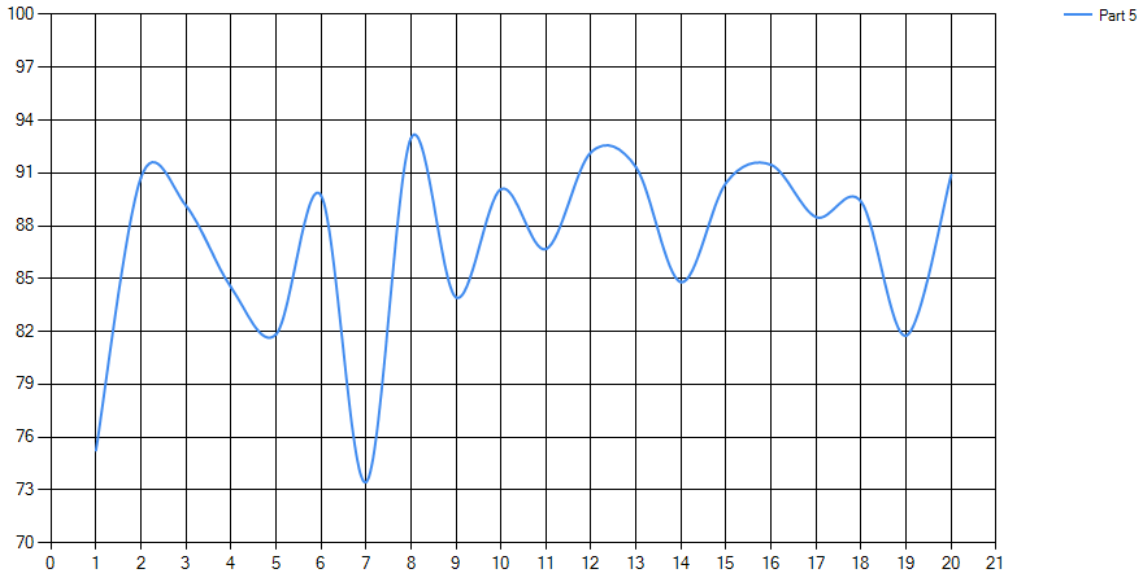
f. Graph:



Part 5

a. The best hyper-parameter is: **0.01 (Margin)**

b. The cross validation accuracy for the best hyper-parameter is: **91.846%**
c. Total number of updates for the best weight and bias: **7826**
d. Development Set accuracy: **92.981%**
e. Test Set accuracy: **93.849%**
f. Graph:



3.4 Experiment Report (Description of Each Part)

1. **Simple Perceptron**
   For this part, I created three separate Data objects that stored all the information needed to determine the best accuracy with the given Learning Rate. I made the five files for the 5-fold cross-validation as parameters for the Data constructor. I also made the number of epochs and the random seed as a parameter. The three separate Data objects each has an accuracy field that is calculated in the 5-fold. I then determine which is the largest accuracy and used the learning rate associated with it. I then created another Data object that has less parameters in its constructor, and I pass in the best learning rate. I then determine the best weight and bias from each of the epochs. I use that weight and bias and create another Data object to get the total accuracy for the test set.
2. **Perceptron with Dynamic Learning Rate**
   For this part, I did the same thing as the part 1, but added an additional parameter in the Data constructor (the constructor to determine the best hyper-parameter) that was a Boolean that said whether or not the learning rate was dynamic. I then used that Boolean in the Perceptron class to modify the learning rate to the correct rate. I initialized t to 0 and incremented for each example. I also kept incrementing through each epoch as the assignment states. I originally incremented for each error that happened and got better accuracy doing it that way than this way, but we were required to increment t it for every example.
3. **Margin Perceptron**
   For this part, I did the same thing as part 2, but added another parameter for margin in the constructor to determine the best hyper-parameter. I then passed in the margin into the

Perceptron object and used that value in the comparison to determine the guess of the label. To make it so the previous experiments still worked I passed in 0 for their margin always. To determine the best hyper-parameter combinations, I had to create nine separate Data objects for all the combinations. I then did the same as in part 1 in determining the best hyper-parameters and calculating the accuracy with the changes I specified in this section.

4. **Averaged Perceptron**

   For this part, I did the same thing as part 1, but I added another parameter in the Data constructor that was a Boolean to say whether or not the data was an averaged perceptron. Of course the dynamic Boolean was set to false and the margin set to 0. In the Data class, I then initialized an average weight and bias if the Average Boolean was true. I then passed in that average WeightBias object into the Perceptron object. At the end of the loop in the algorithm I updated the WeightBias average for each line in the data file. Once that was completed the Perceptron object in the Data class stored the WeightBias. I then set the accuracy of the first fold based off of the average WeightBias. I then got the accuracy for each hyper-parameter and chose the best and did the same thing as discussed in this part to determine the best WeightBias to determine the accuracy.

5. **Aggressive Perceptron with Margin**

   For this section, I did it similar to part 3, except I created an additional parameter to the Data constructor named Aggressive that was another Boolean. The Averaged and Margin Booleans were obviously set to false. As in part 1, I created three Data objects to determine the best hyper-parameter which was the Margin. When it came to setting the margin and comparing each guess for the label, I did it the same way in this part as in part 3. When it came to determining which learning rate to use, I had passed in the Aggressive Boolean into the Perceptron class, then if there was an error I set the learning rate to the correct value (using the formula given). I used that learning rate to modify the weight and bias.