1 Warm up: Linear Classifiers and Boolean Functions

1. It is linearly separable. The linear threshold unit could be:

$$y = \begin{cases} 1 & if \quad x_1 + x_3 - x_2 \geq 0 \\ 0 & otherwise \end{cases} \tag{1}$$

2. It is linearly separable. The linear threshold unit could be:

$$y = \begin{cases} 1 & if \quad x_1 + x_3 + 2x_2 \geq 2 \\ 0 & otherwise \end{cases} \tag{2}$$

3. It is not linearly separable.

4. It is not linearly separable.

5. It is linearly separable. The linear threshold unit could be:

$$y = \begin{cases} 1 & if \quad -x_1 + x_2 - x_3 \geq 1 \\ 0 & otherwise \end{cases} \tag{3}$$

2 Mistake Bound Model of Learning

1. (a) There are only 80 possible values for l. And each l corresponds to a unique function in the concept class. So we can se that

$$|C| = 80 \tag{4}$$

   (b) Define a function

$$g(x_1, x_2) = f_l(x_1, x_2) - y^t \tag{5}$$

   if $g()$ equals zero, no mistake is made here. Otherwise, it makes a mistake.

   (c) i. If $g(x_1, x_2)$ is greater than zero, showing that function f think it is positive but the true label isn't. In this case, the range of f is bigger than we want. So remove all the functions with length no smaller than l.

   ii. If $g(x_1, x_2)$ is smaller than zero, showing that function f thinks that it is negative but the true label is positive. In this case, the range of l is smaller than what we want. So remove all the functions whose length is no bigger than l in the concept class.

Algorithm 1 Mistake-driven Learning Algorithm

(d)
Initialize $C_0 = C$.

for sample (x_1^i, x_2^i) **do**
    Find function f_k whose length k is the middle number among the length of all the functions in current concept class C_i.
    Check if the function f_k made a mistake.

    if f_k doesn't make a mistake **then**
        Assign $C_{i+1} = C_i$
        Continue to the next sample
    else
        Calculate $g = f_k(x_1^i, x_2^i) - y^i$
        if $g > 0$ **then**
            Remove all the functions in C_i with length $\geq k$.
        else
            Remove all the functions in C_i with length $\leq k$.
        end if
    end if
    if $|C_i| = 1$ **then**
        Break the loop.
    end if
    Assign $C_{i+1} = C_i$
end for
return the only remaining function.

2. *Proof.* For the Halving algorithm, the algorithm will stop when only M functions in the concept class. Because the remaining M functions are all perfect experts since they will never be removed from the concept class. Suppose the algorithm made k mistakes before it stops. Then we have:

$$M \times 2^k = N \tag{6}$$

So we know that $k = \log \frac{N}{M}$. It means that the mistakes the algorithm will make is no bigger than $\log \frac{N}{M}$. That is, the mistake bound of it is $O(\log \frac{N}{M})$. □

3 The perceptron Algorithm and its Variants

1. The whole program is realized using $c++$. To realize it, I divide the program into several parts. The main idea is that, I want to encapsulate the procedure inside different classes to simplify the overall complexity of the code. Also, by doing this, the way to reuse the classes are convinient. The first part is called *url*. It handles all the examples. It stores the label and the feature vector of the example and will return corresponding value during query. The second part is called *urls*. This class is used to handle all of the examples. It has two(three when using the dev data) vectors corresponding to the training set and the test data set separately. In each training time, it will fed the algorithm the examples and return required values. The third part is called *perceptron*. This class is used to calculate

the training and test(also dev process in dev dataset) datasets. By fedding it the examples, it will train the perceptron. And by giving the examples, it can return the testing accuracy. That is to say, I encapsulate all the training and testing process inside this class. The final part is the *main c++ file*. It first initialize the perceptron and the urls. Then it reads the required hyperparameters. And it will also control the epoch of the training and output the accuracy.

url This is class is used to inclose all the information of an example. By defining this, I can store the feature and the label inside this class and call the corresponding functions to get the value when needed.

urls This class is used to handle all the examples. It will store the training and the testing examples in two vectors. Everytime the training process starts, just call its function and it can fed the data one by one. By doing this, I can using the whole data when needed and don't need to define any global variables to remember the data.

perceptron This class imitate the process of the perceptron. It contains the weight vector and the bias. These values are randomly initialized at the beginning. After that, everytime just give the dataset to it and specify which process to do, it can finish the process itself and then return the answer. By doing this, the processes are only occured inside this class thus when doing little changes to the perceptron, it is very convinient. It contains the training process, which will do the update of the weight and bias. It also contains the test process, which will return the accuracy of the testset.

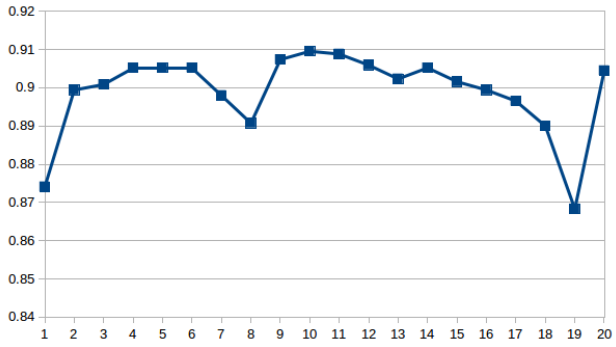
main program The main program mainly used to combine the parts above into a program. It first initialize one *urls* and one *perceptron*. And then it will call the *urls* to read the data. After that, it controls the epoch, and in each epoch, giving the examples to the *perceptron* and require the needed value. At the end of the program, return the accuracy of the testset by calling the *perceptron* to do that.

2. For the test and development set, the accuracy must $\geq 50\%$. And more specificly, suppose there are total N examples and there are M examples with the most frequent label. Then the accuracy would be $N/M \times 100\%$.

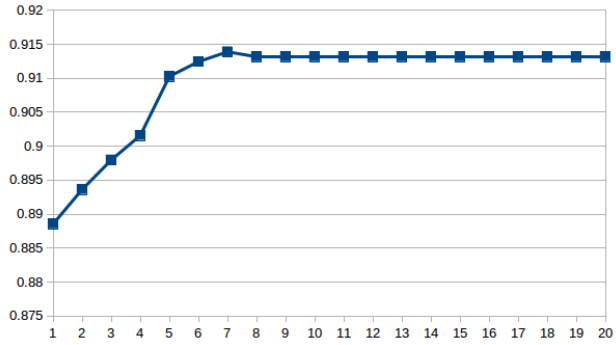
3.

Algorithm	Bst hypr-pam	CV accuracy	# Updates	Dev accuracy	Test accuracy
Basic Perceptron	0.1	0.914603	13101	0.904486	0.912446
Dynamic Perceptron	1	0.92908	10993	0.913169	0.922576
Margin Perceptron	0.1 , 1	0.932699	18571	0.92547	0.929088
Average Perceptron	1	0.930889	25198	0.928365	0.931259
Aggresive Perceptron	0.1	0.916659	27365	0.895803	0.900145

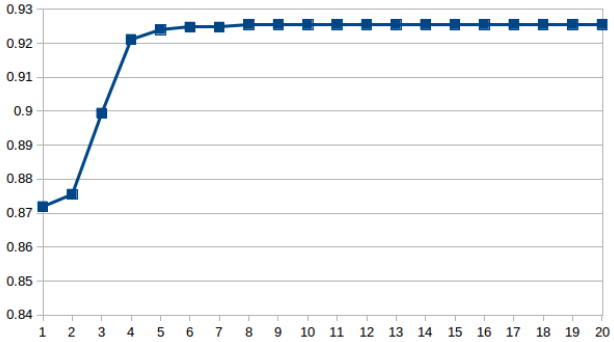
Simple perceptron



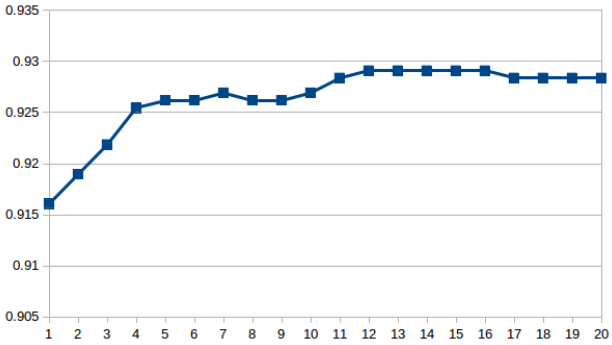
Dynamic perceptron



Margin perceptron



Average perceptron



Aggressive perceptron