1: Warm Up: Feature Expansion

The concept class C consisting of functions f_r is defined by a radius r as follows:

$$f_r(x_1, x_2) = \begin{cases} +1 & 4x_1^4 + 16x_2^4 \leq r; \\ -1 & \text{otherwise} \end{cases}$$

This hypothesis class is *not* linearly separable in $\mathbb{R}^2$. To make positive and negative examples linearly separable, the examples must be mapped to a new space using a function $\phi(x_1, x_2)$ defined as :

$$\phi(x_1, x_2) = \begin{bmatrix} x_1^4 \\ x_2^4 \end{bmatrix}$$

To prove that the positive and negative points are linearly separated in this new space, we can produce a hyperplane that splits them. That is, a weight vector $\mathbf{w}$ and a bias b are found such that $\mathbf{w}^T \phi(x_1, x_2) \geq b$ if, and only if, $f_r(x_1, x_2) = +1$.

$$\mathbf{w} = \begin{bmatrix} -4 \\ -16 \end{bmatrix} \; and \; b = -r$$

2: Mistake Bound Model of Learning

(1) Each function f_r in a concept class **C** is defined by a radius r, where $1 \leq r \leq 80$. This gives the functions $f_1, f_2, ..., f_{79}, f_{80}$ in **C**. For a concept class of size 80.

$$\boxed{|C| = 80}$$

(2) Given an input point (x_1^t, x_2^t) along with it's label y^t, we can use the following expression to check whether the current hypothesis f_r has made a mistake.

$$\boxed{sgn((x_1^t)^2 + (x_2^t)^2 - r^2 - 1) = sgn(y^t)}$$

If both sides of the expression have the same sign, we know we have made a mistake. The intuition is $x_1^2 + x_2^2 - r^2$ will be negative in the case that r^2 is greater than $x_1^2 + x_2^2$ (an incorrect label of -1 is also negative). But $x_1^2 + x_2^2 - r^2$ will be positive when r^2 is less than $x_1^2 + x_2^2$ (an incorrect label of +1 is also positive).

There is an edge case where $x_1^2 + x_2^2 = r^2$. The incorrect labeling is -1, but $sgn(x_1^2 + x_2^2 - r^2) \neq sign(-1)$ in this case. To account for this, one is subtracted from the left side of the equation.

(3) When there is an error, the radius r must be updated. If there is a mistake when $y^t = +1$ then r will be increased by one. Otherwise, if $y^t = -1$ then r will be decreased by one. The radius r is bounded by $1 \leq r \leq 80$ and the modifications to r must obey this.

$$\boxed{y^t = +1 : increase \; r \; by \; one.}$$

$$\boxed{y^t = -1 : decrease \; r \; by \; one.}$$

(4) The mistake-driven algorithm will determine the correct function in 40 mistakes or less.

Algorithm 1 Mistake-Driven Learning

```
1: procedure LEARN FUNCTION(INPUTS)
2:     current_radius = 40
3:     not_all_inputs_passed = True
4:     while not_all_inputs_passed do
5:         not_all_inputs_passed = False
6:         for input in inputs do
7:             if sign(input.x_1^2 + input.x_2^2 − current_radius^2 − 1) == sign(input.y) then
8:                 if input.y == −1 then
9:                     current_radius = current_radius - 1
10:                else
11:                    current_radius = current_radius + 1
12:                end if
13:                not_all_inputs_passed = True
14:                break
15:            end if
16:        end for
17:    end while
18: end procedure
```

The algorithm begins by guessing the function is defined by $r = 40$. For $-80 \leq x_1, x_2 \leq 80$ all possible input points are tested on the current hypothesis using the equality defined in part two.

$$sgn((x_1^t)^2 + (x_2^t)^2 - r^2 - 1) = sgn(y^t)$$

Once a mistake is detected, the hypothesis function is updated and we break out of looping through all input points. If the mistake occurs on a positive example the radius of the hypothesis function is increased by 1. If the mistake take occurs on a negative example the radius is decreased by 1.

This is repeated until all possible input points are tested on the hypothesis function and no mistakes are made. This means we have found the target function. At most there will be 40 mistakes made, which is the case when the target function is defined by $r = 80$.

We know this mistake bound is correct because the radius of the hypothesis function will either move towards 1 or 80. This is a result of there being no noise in the data so it is separable by the hypothesis class. We are zeroing in on the true function as we drop potential functions and the hypothesis space shrinks.

Maximum number of mistakes is 40.

(5a) The original concept class will be a set of functions defined by $1 \leq r \leq 80$. Each time a mistake is made at least half of the functions in the concept class will be discarded. The remaining functions will be a sequence of integers in ascending order with no gaps (ex. 1, 2, 3, 4). To store this list with only two integers, we can store the bounds of the number sequence (ex. [1, 4]).

Keep track of the bounds of the radiuses that define the remaining functions.

(5b) For each function $f_r \in C_i$ (where C_i is the set of remaining hypothesis functions), make a prediction using $f(x_1^t, x_2^t)$. Take the majority label of the predictions (either +1 or -1) and compare it to y^t. If the labels don't match, the functions that made the majority prediction have made a mistake.

(5c) The halving algorithm will remove at least half of the remaining hypothesis functions each time it makes a mistake.

Algorithm 2 The Halving Algorithm

```
1: procedure LEARN FUNCTION(INPUTS, C)
2:     while |C| > 1 do
3:         positive_labels = [ ]
4:         negative_labels = [ ]
5:         for input in inputs do
6:             for function in C do
7:                 if function(x1, x2) == 1 then
8:                     positive_labels.append(function)
9:                 else
10:                    negative_labels.append(function)
11:                end if
12:            end for
13:            if positive_labels is the majority and y^t = −1 then
14:                C_{i+1} = C_i − positive_labels
15:            end if
16:            if negative_labels is the majority and y^t = 1 then
17:                C_{i+1} = C_i − negative_labels
18:            end if
19:        end for
20:    end while
21:    return C
22: end procedure
```

At the beginning of the algorithm, all functions f_r (defined by $1 \leq r \leq 80$) are considered as a hypothesis for the target function. The algorithm is given examples labeled by the target function. For each example, a prediction is made by each function $f_r \in C_i$ where C_i is the set of remaining functions. The majority prediction is compared to y^t. If the prediction doesn't agree with y^t then $C_{i+1} = C_i - C_m$ (where C_m is the set of functions that made the majority prediction). In an actual implementation, the edge case of positive predictions and negative predictions being the same count would need to be considered.

The process continues until $|C| = 1$ which will be the target function. The mistake upper bound is given by:

$$log_2(|C|) = log_2(80) = \ 6.32$$

An intuitive explanation for this bound is the hypothesis class is pruned of at least half of it's functions each time a mistake is made. In the worst case of removing exactly half each mistake, it will take six mistakes to find the target function.

There will be at most 6 mistakes.

3.1: The Perceptron Algorithm and its Variants

(1) Running the simple Perceptron algorithm on the data from *table*2, with a learning rate $r = 0.5$, produces the following weight vector.

$$\mathbf{w} = \begin{bmatrix} 0 \\ 0.5 \\ 0 \\ -0.5 \\ 1 \end{bmatrix}$$

With one pass of the Perceptron algorithm, four mistakes are made on the `table2` dataset.

Four mistakes made.

(2) Before training a binary classifier using the classic Perceptron algorithm, 6-fold cross validation is ran on `a5a.train` to determine a good learning rate r.

Learning Rate	Accuracy
1	77.72%
0.1	78.94%
0.25	79.6%
0.5	78.66%
0.75	77.53%
0.001	68.04%
0.0001	55.44%

Table 1: Classification accuracy for various learning rates.

From this cross validation, the best choice for the learning rate r is 0.25. The Perceptron is trained on `a5a.train` using this best hyperparameter.

Updates made during training: 1,405/6,414 or 21.91%

Using the trained classic Perceptron, I test the classification accuracy of both the `a5a.train` and `a5a.test` data sets.

Algorithm	Data Set	Accuracy
classic Perceptron	a5a.train	5,190/6,414 or 80.92%
classic Perceptron	a5a.test	21,080/26,147 or 80.62%

Table 2: Accuracy of classic Perceptron with a learning rate of 0.25.

This process is repeated using the margin Perceptron. For the margin Perceptron both a learning rate r and margin μ are determined with cross validation.

Margin	Accuracy
0	78.66%
1	78.28%
1.5	79.97%
2	80.82%
2.5	81.99%
3	82.1%
4	82.43%
5	82.4%

Table 3: Classification accuracy for various margins using a learning rate of 0.5.

Learning Rate	Accuracy
1	80.64%
0.1	83.58%
0.25	83.02%
0.5	82.43%
0.75	80.62%
0.001	75.6%
0.0001	66.84%

Table 4: Classification accuracy for various learning rates using a margin of 4.

From the cross-validation, for the margin Perceptron the best margin is $\mu = 4$ and the best learning rate is $r = 0.1$. The margin Perceptron is trained on `a5a.train` using these best hyperparameters.

Updates made during training: 2,386/6,414 or 37.2%

Using the trained margin Perceptron, I test the classification accuracy of both the `a5a.train` and `a5a.test` data sets.

Algorithm	Data Set	Accuracy
margin Perceptron	a5a.train	5,420/6,414 or 84.5%
margin Perceptron	a5a.test	22,037/26,147 or 84.28%

Table 5: Accuracy of the margin Perceptron with a learning rate of 0.1 and a margin of 4.

(3) Using the best learning rate for classical Perceptron from question 2, the Perceptron is trained on `a5a.train` with an additional hyperparameter: the number of epochs. Two Perceptrons are trained, one with 3 epochs and another with 5 epochs.

Updates made during training with 3 epochs: 4,061/19,242 or 21.11%

Updates made during training with 5 epochs: 6,692/32,070 or 20.87%

Using the two trained classic Perceptrons, I test the classification accuracy of both the `a5a.train` and `a5a.test` data sets.

Algorithm	Data Set	Epoch Count	Accuracy
classic Perceptron	a5a.train	3	5,073/6,414 or 79.09%
classic Perceptron	a5a.test	3	20,584/26,147 or 78.72%
classic Perceptron	a5a.train	5	4,439/6,414 or 69.21%
classic Perceptron	a5a.test	5	17,751/26,147 or 67.89%

Table 6: Accuracy of the classic Perceptron with a learning rate of 0.25.

Using the best learning rate and margin for the margin Perceptron, the margin Perceptron is trained on `a5a.train` with both an epoch of 3 and 5.

Updates made during training with 3 epochs: 6,681/19,242 or 34.72%

Updates made during training with 5 epochs: 10,932/32,070 or 34.09%

Using the two trained margin Perceptrons, I test the classification accuracy of both the `a5a.train` and `a5a.test` data sets.

Algorithm	Data Set	Epoch Count	Accuracy
margin Perceptron	a5a.train	3	5,421/6,414 or 84.52%
margin Perceptron	a5a.test	3	22,007/26,147 or 84.17%
margin Perceptron	a5a.train	5	5,386/6,414 or 83.97%
margin Perceptron	a5a.test	5	21,816/26,147 or 83.44%

Table 7: Accuracy of the margin Perceptron with a learning rate of 0.1 and a margin of 4.

(4) To determine the best margin μ for the aggressive Perceptron, 6-fold cross validation is ran on `a5a.train`.

Margin	Accuracy
0	71.52%
1	76.89%
1.5	77.11%
2	77.21%
2.5	77.19%
3	77.19%
4	77.22%
5	77.35%

Table 8: Classification accuracy for various margins using a learning rate of 0.5.

From the cross-validation, the best margin is $\mu = 5$ for aggressive Perceptron with margin. Four aggressive Perceptrons are trained: each with 3 epochs with shuffling, 3 epochs without shuffling, 5 epochs with shuffling, and 5 epochs without shuffling.

Updates made during training with 3 epochs and shuffling: 8,021/19,242 or 41.68%

Updates made during training with 5 epochs and shuffling: 13,273/32,070 or 41.39%

Updates made during training with 3 epochs and no shuffling: 8,152/19,242 or 42.37%

Updates made during training with 5 epochs and no shuffling: 13,459/32,070 or 41.97%

Using the four trained aggressive Perceptrons, I test the classification accuracy of both `a5a.train` and `a5a.test` data sets.

Algorithm	Data Set	Epoch Count	Shuffling	Accuracy
aggressive Perceptron	a5a.train	3	yes	5,313/6,414 82.83%
aggressive Perceptron	a5a.test	3	yes	21,676/26,147 82.9%
aggressive Perceptron	a5a.train	3	no	5,200/6,414 81.07%
aggressive Perceptron	a5a.test	3	no	20,969/26,147 80.2%
aggressive Perceptron	a5a.train	5	yes	4,592/6,414 71.59%
aggressive Perceptron	a5a.test	5	yes	18,376/26,147 70.28%
aggressive Perceptron	a5a.train	5	no	5,195/6,414 80.99%
aggressive Perceptron	a5a.test	5	no	20,964/26,147 80.18%

Table 9: Accuracy of the aggressive Perceptron with a margin of 5.

Notes on experiments: The weight vector and bias used by all the Perceptrons in the experiments and cross-validations were initialized to a random number in the range [-1, 1]. The random number generator was seeded to get consistent results. The exception being experiment 1, where the weight vector and bias were initialized to 0.

The cross validation was done with `testHyperParameters.py` and the four experiments are

conducted with `experiment1.py, experiment2.py, experiment3.py, and experiment4.py`. The shell script `run.sh` with run all four of the experiments. Additionally, `experiments.trace` and `hyperparameters.trace` contain the output.