COMP 6321 Machine Learning

Assignment 1 Answers

Name : Parsa Kamalipour , *StudentID* : 40310734

Exercise 1: Perceptron (7 pts)

NOTE This is a theoretical question, and you are not required to code anything.

1. (1.5 points) Consider the dataset shown in Table 1 where $\mathrm{x} = (x_1, x_2) \in \{0,1\} \times \{0,1\}, y \in \{-,+\}$. Note that this dataset corresponds to the boolean function "AND" over the 2-bit binary input. Suppose we are training a Perceptron to learn on this dataset and we initialize the weights and the bias, $w^{(0)} = 0$ and $w_0^{(0)} = 0$. Please i) answer if this dataset is learnable by a Perceptron, and ii) if so, write down the weights update procedure for each iteration; if not, explain why.

2. (1.5 points) Extending AND to any boolean functions over a 2-bit binary input, where we have $2^{2^2} = 16$ possible distinct boolean functions in total, among which how many of them can be learnable by a Perceptron? Please also write down the truth table(s) of the boolean functions that are **not learnable**, if there are any. [Hint AND is one of the 16 boolean functions, with the output -, -, -, +. Similarly, a constant function represents two of the 16 boolean functions, with the output -, -, -, - or +, +, +, +.]

3. (4 points) *Modified Perceptron Algorithm* Recall the Perceptron algorithm we learned in class: for each "mistake" we update the weights by setting $w \leftarrow w + y_i \mathrm{x}_i, w_0 \leftarrow w_0 + y_i$. Now we would like to modify this algorithm by considering a constant $c > 1$ and making modifications to the update rule $w \leftarrow w + cy_i \mathrm{x}_i, w_0 \leftarrow w_0 + cy_i$. Let's call this algorithm "modified Perceptron". Please **prove or refute** that the modified Perceptron algorithm converges after the same number of mistakes as made by the Perceptron algorithm. [Hint: c.f. the convergence theorem in lecture 2]

x_1	x_2	y
0	0	-
0	1	-
1	0	-
1	1	+

Table 1: The Truth Table for AND Function.

Answers to Exercise 1

1) Based on the info given in this assignment, we know that we have a dataset that basically simulates the AND function for two bits, we have 2 inputs noted as $x = (x_1, x_2)$ which can only hold the value of either zero or one, we also know that our output $y \in -1, +1$ can only hold positive or negative sign, which means either -1 or +1.

a) Yes! the dataset and AND function is learnable by a perceptron because we can basically divide the results in two. For example, in our scenario every combination of x_1, x_2 except 1,1 will results in negative sign! this clearly shows that we can separate the results with just one non-curved hyperplane.

b) We know that initially our weights is set as zero $w = [0, 0]$ and also our bias is set as zero $w_0 = 0$ and our dataset is shown in table 1.
Now we have:

$$x_1, x_2 \in 0, 1$$

$$y \in -1, +1$$

$$w^{(0)} = [0, 0]\ , \quad w_0^{(0)} = 0$$

And this is our Perceptron Learning algorithm [1] showed below this question. Now based on our algorithm we have:
First Pass Through:

- Iteration 1:
  Input: $[0, 0], y = -1$
  Prediction: $\hat{y} = 0$
  Update weights: $w = [0, 0], w_0 = -1$
- Iteration 2:
  Input: $[0, 1], y = -1$
  Prediction: $\hat{y} = -1$ **(CORRECT!)**
  Update weights: **(NO UPDATE!)**
- Iteration 3:
  Input: $[1, 0], y = -1$
  Prediction: $\hat{y} = -1$ **(CORRECT!)**
  Update weights: **(NO UPDATE!)**
- Iteration 4:
  Input: $[1, 1], y = +1$
  Prediction: $\hat{y} = -1$ **(INCORRECT!)**
  Update weights: $w = [1, 1], w_0 = 0$

Second Pass Through:

- Iteration 1:
  Input: $[0, 0], y = -1$
  Prediction: $\hat{y} = 0$ **(INCORRECT!)**
  Update weights: $w = [1, 1], w_0 = -1$
- Iteration 2:
  Input: $[0, 1], y = -1$
  Prediction: $\hat{y} = 0$ **(INCORRECT!)**
  Update weights: $w = [1, 0], w_0 = -2$
- Iteration 3:
  Input: $[1, 0], y = -1$
  Prediction: $\hat{y} = -1$ **(CORRECT!)**
  Update weights: **(NO UPDATE!)**
- Iteration 4:
  Input: $[1, 1], y = +1$
  Prediction: $\hat{y} = -1$ **(INCORRECT!)**
  Update weights: $w = [2, 1], w_0 = -1$

Third Pass Through:

- Iteration 1:
  Input: $[0, 0], y = -1$
  Prediction: $\hat{y} = -1$ **(CORRECT!)**
  Update weights: **(NO UPDATE!)**
- Iteration 2:
  Input: $[0, 1], y = -1$
  Prediction: $\hat{y} = -1$ **(CORRECT!)**
  Update weights: **(NO UPDATE!)**
- Iteration 3:
  Input: $[1, 0], y = -1$
  Prediction: $\hat{y} = -1$ **(CORRECT!)**
  Update weights: **(NO UPDATE!)**
- Iteration 4:
  Input: $[1, 1], y = +1$
  Prediction: $\hat{y} = +1$ **(CORRECT!)**

Update weights: **(NO UPDATE!)**

At this point we will know that we have reached the convergence and we can stop because our model learned the AND function successfully!

2) Let's expand our function from AND one to every combination of 2bit boolean one!
So basically we have 16 different functions to test and lets begin:

a) ***Constant Function #1:***
All outputs are -1 no what the input is.
This one is **learnable** because if you look at the outputs regarding the inputs you can see that a hyperplane can divide the points (in this case all points will be in one side).

b) ***Constant Function #2:***
All outputs are +1 no matter what the input is.
This one is also **learnable** due to it being linearly separable.

c) ***AND Function:***
Output is +1 when only both inputs are 1.
This one is also **learnable** due to it being linearly separable.

d) ***OR Function:***
Output is +1 when at least one of the inputs are 1.
This one is also **learnable** due to it being linearly separable.

e) ***NAND Function:***
Output is -1 when only both inputs are 1.
This one is also **learnable** due to it being linearly separable.

f) ***NOR Function:***
Output is +1 when only both inputs are 0.
This one is also **learnable** due to it being linearly separable.

g) ***XOR Function:***
Output is +1 when exactly one of inputs is 1.
This one is **NOT learnable** because no single line can separate the two positive outputs from the two negative ones.

h) ***XNOR Function:***
Output is +1 when both of the inputs equals each other.
This one is **NOT learnable** because no single line can separate the two positive outputs from the two negative ones.

i) ***Un-named Function #1:*** For input $x_1, x_2 = (0,0), (0,1), (1,0), (1,1)$ it has the following outputs: $y = -1, +1, -1, -1$ which makes it linearly separable since its data can be divided by one hyperplane.

j) ***Un-named Function #2:*** For input $x_1, x_2 = (0,0), (0,1), (1,0), (1,1)$ it has the following outputs: $y = +1, -1, -1, -1$ which makes it linearly separable since its data can be divided by one hyperplane.

k) ***Un-named Function #3:*** For input $x_1, x_2 = (0,0), (0,1), (1,0), (1,1)$ it has the following outputs: $y = +1, +1, +1, -1$ which makes it linearly separable since its data can be divided by one hyperplane.

l) ***Un-named Function #4:*** For input $x_1, x_2 = (0,0), (0,1), (1,0), (1,1)$ it has the following outputs: $y = +1, +1, -1, +1$ which makes it linearly separable since its data can be divided by one hyperplane.

m) ***Un-named Function #5:*** For input $x_1, x_2 = (0,0), (0,1), (1,0), (1,1)$ it has the following outputs: $y = -1, +1, +1, +1$ which makes it linearly separable since its data can be divided by one hyperplane.

n) ***Un-named Function #6:*** For input $x_1, x_2 = (0,0), (0,1), (1,0), (1,1)$ it has the following outputs: $y = +1, -1, +1, +1$ which makes it linearly separable since its data can be divided by one hyperplane.

o) ***Un-named Function #7:*** For input $x_1, x_2 = (0,0), (0,1), (1,0), (1,1)$ it has the following outputs: $y = -1, +1, -1, +1$ which makes it linearly separable since its data can be divided by one hyperplane.

p) ***Un-named Function #8:*** For input $x_1, x_2 = (0,0), (0,1), (1,0), (1,1)$ it has the following outputs:

$y = -1, -1, +1, +1$ which makes it linearly separable since its data can be divided by one hyperplane

3) Now lets solve the Modified perceptron algorithm question:
We know that:
a) there exist w^* such that $y_i \frac{w^* x_i}{||w^*||} \geq \gamma > 0$ for all i.
b) $||x_i|| \leq R$
And the update formula was like:

$$w = w + y_i x_i$$
$$w_0 = w_0 + y_i$$

Then:
The perceptron will make AT MOST $(R/\gamma)^2$ mistakes.

From all that we can get that:

$$\gamma = \min_i (y_i \frac{w^* x_i}{||w^*||})$$
$$R = \max_i ||x_i||$$

For our modified perceptron, our updates will be like:

$$w \leftarrow w + c y_i x_i$$
$$w_0 \leftarrow w_0 + c y_i$$

For all $c > 1$.

Now we will assume these:

$$w^* = [Vector\ of\ weights]$$
$$w^k : weight\ for\ function\ F\ after\ k\ mistakes$$
$$K\ is\ final\ iteration$$

And we will prove that that the modified Perceptron algorithm converges after the same number of mistakes as made by the Perceptron algorithm in this way:

$$\frac{w^* w^k}{||w^*||} = \frac{w^{k-1} w^* + c y_i x_i w^*}{||w^*||} = \frac{w^{k-1} w^*}{||w^*||} + c \frac{y_i x_i w^*}{||w^*||}$$

and we know that $\frac{y_i x_i w^*}{||w^*||} \geq \gamma$ thus:

$$\rightarrow \frac{w^* w^k}{||w^*||} \geq \frac{w^{k-1} w^*}{||w^*||} + c\gamma K$$

After that we follow with this:

$$||w^k||^2 = ||w^{k-1} + c y_i x_i||^2 = ||w^{k-1}||^2 + 2 c y_i x_i w^{k-1} + ||c x_i y_i||^2$$

Which $2 c y_i x_i w^{k-1}$ is a negative number and $||c x_i y_i||^2$ equals to $c^2 ||x_i||^2$ therefore:

$$\rightarrow ||w^k||^2 \leq ||w^{k-1}||^2 + c^2 R^2 K$$

Based on our last two findings we can get:

$$\frac{1}{||w^k||^2} \geq \frac{1}{c^2R^2K}$$
$$\rightarrow \frac{1}{||w^k||} \geq \frac{1}{cR\sqrt{K}}$$

We plug this in and find out that:

$$Cos(w^*, w^k) \geq \frac{Kc\gamma}{\sqrt{K}Rc} \geq \sqrt{K}\frac{\gamma}{R}$$

As you can see in the privous step our two "c" got canceled together and it shows that no longer "c" is effecting our equations which basically means **our modified perceptron doesn't change convergence**, we can continue to prove it further like this:

$$\rightarrow \frac{R}{\gamma} \geq \sqrt{K} \rightarrow K \leq (\frac{R}{\gamma})^2$$

And just like this we have proved that adding a "c" won't change the convergence.

Algorithm 1 Perceptron Learning Algorithm

```
procedure LEARNING(w, w_0, x, y, hyperparameter)
    Initialize weights w and bias w_0
    for t = 1 to hyperparameter do
        for i = 0 to N do
            if y_i(w · x_i + w_0) ≤ 0 then
                w_new = w_old + y_i x_i
                w_0^new = w_0^old + y_i
    Return w, w_0
```

Exercise 2: Logistic Regression (9 pts)

In this exercise you will implement the logistic regression algorithm and evaluate it on the dataset provided with this assignment. The training dataset is divided into five different csv files. You need to combine this into a single training dataset. Do not use any machine learning library, but feel free to use libraries for linear algebra and feel free to verify your results with existing machine learning libraries.

1. (2 pts) Let $\Pr(C_1|x) = \sigma(\mathbf{w}^T\mathbf{x} + w_0)$ and $\Pr(C_2|\mathbf{x}) = 1 - \sigma(\mathbf{w}^T\mathbf{x} + w_0)$. Learn the parameters $\mathbf{w}$ and w_0 using the gradient descent algorithm. Use the maximum number of epochs to be 100 for GD. You can also use any appropriate convergence criteria to stop the GD loop before 100 epochs. Use a step size of 0.1 (or 0.01 if it is converging poorly) for GD.
2. (0.25 pts) After the training process, create the following plots:
   (a) test error vs the number of epochs
   (b) training error vs the number of epochs
   (c) test loss vs the number of epochs
   (d) training loss vs the number of epochs
   (e) Print the parameters $\mathbf{w}, w_0$ found for logistic regression.
3. (5 pts) Now let's add a regularization term $0.5\lambda||\mathbf{w}||_\mathbf{2}^\mathbf{2}$ to the loss function of your logistic regression algorithm. Your new loss function will be $\mathcal{L}_{new} = \mathcal{L}_{old} + 0.5\lambda||\mathbf{w}||_\mathbf{2}^\mathbf{2}$, where $\mathcal{L}_{old}$ is the loss function of logistic regression we learned in lecture 3. Choose two values of $\lambda = \{0.5, 1\}$ and train this algorithm on the given dataset with these two values of λ.

4. (0.25 pts) After the training process, create the plots mentioned in step 2, for this updated logistic regression algorithm. You should have two sets of plots since you used two different values of *lambda* to train the model.

5. (1.5 pts) Compared the results for step 2 with the results for the updated logistic regression with two values of *lambda*. Which of the tree algorithms performs the best? Why? Compare the values of $\mathbf{w}, w_0$ for all three cases. How is λ affecting these parameter values, and the overall error?

6. (Extra Credit: 1.5 pts) Use 5-fold cross-validation on the training dataset to find the best value for λ. Report the best λ, and also draw a plot that shows the cross-validation error of logistic regression as λ varies. Note that the training dataset is already divided into 5 different csv files, to ease the cross-validation process. Report the test error for the best λ. Is this better than the results in steps 2, and 5? Why or why not?

Answers to Exercise 2

Please look at the ParsaKamalipour_COMP6321_A1.ipynb file to see my code for this question.