FAI HW3

B10902024 林宸宇

May 5, 2024

1 Hand-written Part

1.1 Problem 1

If $yw^Tx \geq 1$, $err(w^Tx, y) = 0$.

If $yw^Tx < 1$, $err(w^Tx, y) = (1 - yw^Tx)^2$.

Therefore,

$$\frac{\partial err(w^Tx, y)}{\partial w} = \begin{cases} 0, & \text{if } yw^Tx \geq 1 \\ -2yx(1 - yw^Tx), & \text{otherwise} \end{cases}$$

The above equation can be rewritten as

$$\frac{\partial err(w^Tx, y)}{\partial w} = -2yx(max(1 - yw^Tx, 0))$$

Finally,

$$\nabla E_{in}(w) = \frac{-2}{N} \sum_{n=1}^{N} y_n x_n (max(1 - y_n w^T x_n, 0))$$

1.2 Problem 2

By definition,

$$p_u(x_n) = \frac{1}{\sqrt{(2\pi)^d}} e^{\frac{-1}{2}(x_n - u)^T (x_n - u)}$$

We can then rewrite the estimation of u,

$$\begin{aligned} u^* &= argmax_{u \in R^d} \prod_{n=1}^{N} p_u(x_n) \\ &= argmax_{u \in R^d} \sum_{n=1}^{N} \log p_u(x_n) \\ &= argmax_{u \in R^d} \sum_{n=1}^{N} -(x_n - u)^T (x_n - u) \\ &= argmin_{u \in R^d} \sum_{n=1}^{N} (x_n - u)^T (x_n - u) \end{aligned} \tag{1}$$

Then we can take the derivatives to find u^*,

$$-2\sum_{n=1}^{N}(x_n - u^*) = 0$$

$$Nu^* = \sum_{n=1}^{N} x_n$$

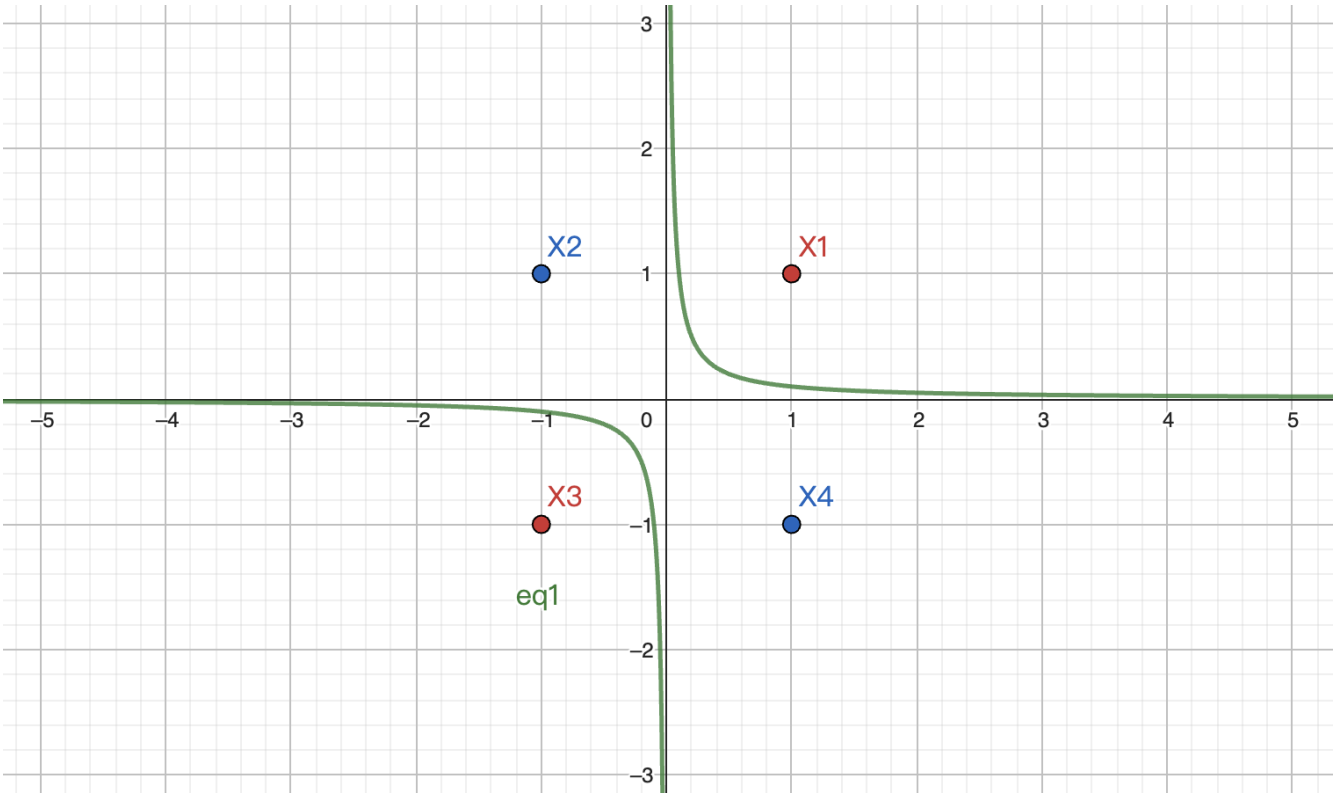
$$u^* = \frac{1}{N}\sum_{n=1}^{N} x_n$$

1.3 Problem 3

We can easily set $\tilde{w} = (0.1, 0, 0, 0, -1, 0)$ as the perceptron. It only considers the x_1x_2 term, and it can separate those 4 data correctly. The classification boundary

$$\tilde{w}\Phi_2(x) = 0.1 - x_1x_2 = 0$$

Below is the classification boundary graph



1.4 Problem 4

By the definition of error rate, we can calculate that

$$1 - \epsilon_t = \frac{\sum_{n=1}^{N} w_n^t(1 - \delta(g_t(x_n), y_n))}{\sum_{n=1}^{N} w_n^t}$$

We can also rewrite the equations

$$\sum_{n=1}^{N} w_n^t(\delta(g_t(x_n), y_n)) = \epsilon * \sum_{n=1}^{N} w_n^t$$

$$\sum_{n=1}^{N} w_n^t(1 - \delta(g_t(x_n), y_n)) = (1 - \epsilon) * \sum_{n=1}^{N} w_n^t$$

After updating the new weights, the numerator of error rate becomes

$$\begin{aligned}
\sum_{n=1}^{N} w_n^{t+1}(\delta(g_t(x_n), y_n)) &= d_t * \sum_{n=1}^{N} w_n^t(\delta(g_t(x_n), y_n)) \\
&= \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} * \sum_{n=1}^{N} w_n^t(\delta(g_t(x_n), y_n)) \\
&= \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} * \epsilon_t * \sum_{n=1}^{N} w_n^t \\
&= \sqrt{(1-\epsilon_t) * \epsilon_t} * \sum_{n=1}^{N} w_n^t
\end{aligned} \tag{2}$$

The denominator of error rate becomes

$$\begin{aligned}
\sum_{n=1}^{N} w_n^{t+1} &= \sum_{n=1}^{N} w_n^{t+1} * \delta(g_t(x_n), y_n) + \sum_{n=1}^{N} w_n^{t+1} * (1 - \delta(g_t(x_n), y_n)) \\
&= \sqrt{(1-\epsilon_t) * \epsilon_t} * \sum_{n=1}^{N} w_n^t + \sum_{n=1}^{N} w_n^{t+1} * (1 - \delta(g_t(x_n), y_n)) \\
&= \sqrt{(1-\epsilon_t) * \epsilon_t} * \sum_{n=1}^{N} w_n^t + \frac{\sum_{n=1}^{N} w_n^t(1 - \delta(g_t(x_n), y_n))}{d_t} \\
&= \sqrt{(1-\epsilon_t) * \epsilon_t} * \sum_{n=1}^{N} w_n^t + \sqrt{\frac{\epsilon_t}{1-\epsilon_t}} * (1-\epsilon) * \sum_{n=1}^{N} w_n^t \\
&= 2\sqrt{(1-\epsilon_t) * \epsilon_t} * \sum_{n=1}^{N} w_n^t
\end{aligned} \tag{3}$$

Thus, the error rate after updating with new weights equals

$$\begin{aligned}
\frac{\sum_{n=1}^{N} w_n^{t+1}(\delta(g_t(x_n), y_n))}{\sum_{n=1}^{N} w_n^{t+1}} &= \frac{\sqrt{(1-\epsilon_t) * \epsilon_t} * \sum_{n=1}^{N} w_n^t}{2\sqrt{(1-\epsilon_t) * \epsilon_t} * \sum_{n=1}^{N} w_n^t} \\
&= 0.5
\end{aligned} \tag{4}$$

2 Programming Part

1. Source code (Python)

2. A detailed report answering the following questions:

Model	Linear	Decision Tree	Random Forest
Classification Accuracy	0.667	0.889	0.933
Mean Square Error	43.414	34.443	28.569

(a) Result

In terms of performance, random forest outperforms the other two model for both tasks. For the classification task, values of some features varies greatly, so well designed decision tree can split the data better than liner model. Instead of relying on a single decision tree, random forest in this case combines 100 trees with sampling results in a more robust model.

As for the regression task, some features might contain dependencies or other relations, which the linear model can't model. Thus, the more complicated nonlinear random forest has the better performance.

(b) Result

Model	Logistic Regression	
Techniques	Normalization	Standardization
Classification Accuracy	0.667	0.889
Mean Square Error	43.414	21.908

Normalization is used when the features have different scales and we want to scale them to a similar range, such as 0 to 1. Standardization is useful when the data is deviate and we want to transform them to have a mean of 0 and a standard deviation of 1.

There's no absolute preference between normalization and standardization. The selection between them depend on the context of datasets. In the given tasks, with the existence of outliers and some features having much higher/lower values, standardization might be a better choice.

(c) Learning rate (iterations = 1000)

Model	Logistic Regression			
Learning rate	0.01	0.02	0.1	0.5
Classification Accuracy	0.667	0.756	0.911	0.956
Mean Square Error	43.414	39.705	57.549	63.598

Iterations (learning rate = 0.01)

Model	Logistic Regression				
Iterations	100	1000	2000	10000	500000
Classification Accuracy	0.622	0.667	0.756	0.911	0.978
Mean Square Error	88.999	43.414	39.705	57.536	63.363

The universally best combination of learning rate and number of iterations doesn't exist, so the below discussion only focus on our dataset. For the classification task, increasing the learning rate or number of iterations seems to yield better result. However, setting too large learning rate might lead to overflow while setting too many iterations might be time consuming. Thus, despite increasing learning rate and number of iterations seems better, we should still choose them cautiously.

For the regression task, there's a sweet spot around learning rate = 0.02 + iterations = 1000. With lower learning rate or smaller number of iterations, the model will underfit and leads to higher MSE. On the other hand, with higher learning rate or larger number o iterations, the model might overfit and result in higher MSE as well.

For both tasks, increasing learning rate seems to have same effect as increasing number of iterations. The overall configurations can sort of be viewed as the product of learning rate and number of iterations.

(d) Sample Rate (number of trees = 100, maximum depth = 5)

Model	Random Forest			
Sample Rate	0.3	0.5	0.7	0.9
Classification Accuracy	0.911	0.933	0.911	0.889
Mean Square Error	28.805	28.569	29.893	28.972

Number of trees (sample rate = 0.5, maximum depth = 5)

Model	Random Forest			
Number of Trees	10	30	100	500
Classification Accuracy	0.911	0.956	0.933	0.933
Mean Square Error	30.859	28.821	28.569	28.846

Maximum Depth (sample rate = 0.5, number of trees = 100)

Model	Random Forest			
Maximum Depth	1	3	5	10
Classification Accuracy	0.6	0.956	0.933	0.933
Mean Square Error	47.465	30.720	28.569	28.551

When the number of trees is large enough, the sample rate seems doesn't affect a lot to the performance. Just make sure the sample is within a reasonable range (very low sample rate makes all decision trees inaccurate while very high sample rate makes random forest equivalent to a single decision tree).

In theory, the larger number of trees yield a more robust model since it increases the model complexity and generalization ability. Larger maximum depth increase the model complexity as well, but might lead to overfitting if not carefully designed. In my implementation, I tried several combinations with my predefined features. I found that number of trees over 30 is good enough. Keep increasing the number of trees won't increase the accuracy or lower the MSE. As for maximum depth, depth more than 3 can yield decent result. To sum up, considering the performance and time efficiency, I'll select sample rate = 0.5, number of trees = 30 and maximum depth = 5 for our tasks.

(e) Linear model has the lowest model complexity. It's easy to implement and time efficient, but it has lower interpretability and can't handle complex dataset. On the other hand, decision tree is much more complex and more difficult to implement. However, when dealing with large amount of data or the features within data contains nonlinear relations, decision tree has better generalization ability. Random forest inherits the pros of decision trees with reduced susceptibility to overfitting. Despite it seems perfect, its training process is time-consuming.

Considering all the above factors, I'll choose to use linear model if the data is simple and have a significant linear separable pattern. For those cases, linear model will yield a good enough solution, and we can expect the testing accuracy is close to training accuracy. Conversely, if the data has multiple features, and those features are correlated or not linear separable, I'll choose random forest for better performance.