Yen Jui Chieh
B09902136

Homework #3

CSIE3005 Foundations of Artificial Intelligence

May 19, 2023

1 Hand-written Part

Problem 1

The gradient of the squared hinge error is

$$\nabla_{\mathbf{w}} \operatorname{err}(\mathbf{w}^T\mathbf{x}, y) = -2\mathbf{x}y \max(1 - y\mathbf{w}^T\mathbf{x}, 0).$$

Therefore, the gradient of the in-sample error is

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \nabla_{\mathbf{w}} \operatorname{err}(\mathbf{w}^T\mathbf{x}_n, y_n) = \frac{-2}{N}\sum_{n=1}^{N} \mathbf{x}_n y_n \max(1 - y_n\mathbf{w}^T\mathbf{x}_n, 0).$$

Problem 2

The probability density function of $\mathcal{N}(\mathbf{u}, \mathbf{I})$ is

$$p_{\mathbf{u}}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{u})^T(\mathbf{x}-\mathbf{u})\right).$$

The maximum likelihood estimate of $\mathbf{u}$ is then

$$\mathbf{u}^* = \operatorname*{arg\,max}_{\mathbf{u}\in\mathbb{R}^d} \prod_{n=1}^{N} p_{\mathbf{u}}(\mathbf{x}_n) = \operatorname*{arg\,max}_{\mathbf{u}\in\mathbb{R}^d} \sum_{n=1}^{N} \log p_{\mathbf{u}}(\mathbf{x}_n) = \operatorname*{arg\,min}_{\mathbf{u}\in\mathbb{R}^d} \sum_{n=1}^{N} (\mathbf{x}_n-\mathbf{u})^T(\mathbf{x}_n-\mathbf{u}) = \operatorname*{arg\,min}_{\mathbf{u}\in\mathbb{R}^d} \sum_{n=1}^{N} f(\mathbf{u}),$$

where $f(\mathbf{u}) := \sum_{n=1}^{N}(\mathbf{x}_n-\mathbf{u})^T(\mathbf{x}_n-\mathbf{u})$. To find $\mathbf{u}^*$, we equate the gradient of f computed at $\mathbf{u}^*$ to zero:

$$\nabla_{\mathbf{u}} f(\mathbf{u}^*) = -2\sum_{n=1}^{N}(\mathbf{x}_n - \mathbf{u}^*) = 0. \qquad (\star)$$

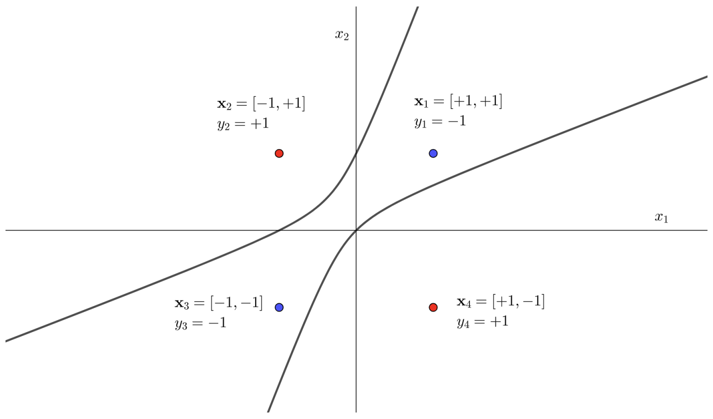
The solution to $(\star)$ is $\mathbf{u}^* = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n$.

Problem 3

Let $\tilde{\mathbf{w}} = (0, 1, -1, 1, -3, 1)$ be the perceptron. It can be easily shown that $\tilde{\mathbf{w}}$ correctly separates the data. The corresponding classification boundary is

$$\tilde{\mathbf{w}}^T\Phi_2(\mathbf{x}) = x_1 - x_2 + x_1^2 - 3x_1x_2 + x_2^2 = 0.$$

Below is the plot of this boundary, which looks like a quadratic curve. We see that $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$ are classified perfectly.



Problem 4

Denote by $[N]$ the set of integers from 1 to N. We define

$$S_t := \{ n \in [N] \colon g_t(\mathbf{x}_n) \neq y_n \}$$

and $S'_t := [N]/S_t$. Since

$$\frac{\sum_{n=1}^{N} w_n^{t+1} \delta(g_t(\mathbf{x}_n), y_n)}{\sum_{n=1}^{N} w_n^{t+1}} = \frac{\sum_{n \in S_t} w_n^{t+1}}{\sum_{n \in S_t} w_n^{t+1} + \sum_{n \in S'_t} w_n^{t+1}} ,$$

to show that the fraction above equals 0.5, it suffices to show that

$$\sum_{n \in S_t} w_n^{t+1} = \sum_{n \in S'_t} w_n^{t+1} . \qquad (\bullet)$$

Note that the error rate ε_t can be rewritten as

$$\varepsilon_t = \frac{\sum_{n \in S_t} w_n^t}{\sum_{n \in S_t} w_n^t + \sum_{n \in S'_t} w_n^t} ,$$

and thus

$$d_t = \sqrt{\frac{1}{\varepsilon_t} - 1} = \sqrt{\frac{\sum_{n \in S'_t} w_n^t}{\sum_{n \in S_t} w_n^t}} .$$

Therefore,

$$\sum_{n \in S_t} w_n^{t+1} = \sum_{n \in S_t} w_n^t \cdot d_t = \sqrt{\left(\sum_{n \in S_t} w_n^t \right) \left(\sum_{n \in S'_t} w_n^t \right)} = \left(\sum_{n \in S'_t} w_n^t \right) \sqrt{\frac{\sum_{n \in S_t} w_n^t}{\sum_{n \in S'_t} w_n^t}} = \sum_{n \in S'_t} w_n^t / d_t = \sum_{n \in S'_t} w_n^{t+1} .$$

2 Programming Part

1. See `hw3.py` in `b09902136.zip`.
2. (a) Below is the performance of our implementation.
    - Logistic regression accuracy: 0.6666666666666666.
    - Decision tree classifier accuracy: 0.8888888888888888.
    - Random forest classifier accuracy: 0.9333333333333333.
    - Linear regression MSE: 43.413924633863004.
    - Decision tree regressor MSE: 34.24809957115192.
    - Random forest regressor MSE: 28.859538082029687.

    We can see that in both classification and regression tasks, decision tree and random forest algorithms have better performances. In the case of classification, I assume that it's because samples of different classes may have some features that are far greater/smaller than those of other classes. For example, a certain type of flowers can have far longer petals than other types of flowers. Decision trees with good splitting method can clearly utilize such characteristics better than logistic regression. In the case of regression, I'm convinced it's because decision trees also support nonlinear solutions, while linear regression only supports linear solutions. Therefore, when a good linear solution doesn't exist for the given samples, decision trees tend to have a better performance.

    (b) Below is the performance of logistic regression with normalized and standardized datasets.
    - With normalization, logistic regression achieves 0.6666666666666666 accuracy.
    - With standardization, logistic regression achieves 0.8888888888888888 accuracy.

    We standardize datasets to make sure each feature contributes to the analysis equally and avoid creating a bias. We normalize datasets to make the features have the same scales, which is especially useful when our algorithms don't make assumptions about the distribution of datasets. I don't think either technique has any absolute advantage or disadvantage over the other. Instead, I suppose the performances of normalization and standardization depend on the given datasets.

    (c) Logistic regression model achieves 0.6666666666666666 accuracy after 1000 iterations with learning rate 0.01. Meanwhile, with learning rate 0.3 it achieves 0.9555555555555556 accuracy after the same number of iterations.

    I suggest that the higher the learning rate is, the better the solution it yields. However, when the learning rate is too large, gradient descent can increase rather than decrease the training error. For example, out logistic regression model achieves a horrendous accuracy of 0.2222222222222222 after 1000 iterations when the learning rate is 10000. Therefore, if the number of iterations is fixed, with learning rate being too low or too high, the performance may be worse. On the other hand, with a decent learning rate, logistic regression model yields better solution after more iterations, but apparently consumes more time.

    (d) The more trees the random forest model has, or the greater the maximum depth of its trees is, the model's complexity and generalization ability increase, while its potential for overfitting decreases. I didn't spend too much time tuning the hyperparameters of my random forest model; I focused more on the correctness of my implementation. I kept 100 decision trees in the model, each of which has maximum depth 5. Its performances as a classifier and regressor have been shown in (a).

    (e) In my opinion, decision tree and random forest models have far better interpretability and performance in classification tasks, but much more complexity in their implementation. Therefore, I would prefer these two nonlinear models over logistic regression models, especially in multinomial classification, since it's usually difficult to learn a weight matrix that can correctly separate different classes of data.

    When it comes to regression tasks, I think all the three models have similar interpretability, but the linear regression has slightly worse performance than the other two, despite having the least complexity. It's not like one can always find a great linear solution for every dataset, and nonlinear models such as decision trees and random forests can handle well more sorts of datasets. Random forests have the best performance, since it's based on decision tree models and it reduces the potential for overfitting. I would prefer random forest models for regression tasks.

3 References

3.1 Hand-written Part

1. https://hal.science/hal-00708243v1/file/appendixMNBCI.pdf
2. https://stats.stackexchange.com/questions/351549/maximum-likelihood-estimators-multivariate-gaussian

3-4. None.

3.2 Programming Part

None.