Foundations of Artificial Intelligence: Homework 3

b11902038 鄭博允

May 10, 2025

Problem 1

Let $z_n = y_n \mathbf{w}^T x_n$, then the gradient is

- $\nabla \text{err}() = 0$, if $z_n \geq 1$
- $\nabla \text{err}() = -2(1 - z_n) y_n x_n$, if $z_n < 1$

Therefore,

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{1}{N} \sum_{n: y_n \mathbf{w}^T x_n < 1} -2(1 - y_n \mathbf{w}^T x_n) y_n x_n$$

Problem 2

1. substitute the X:

$$X^TX = (V\Sigma^TU^T)(U\Sigma V^T) = V\Sigma^T\Sigma V^T$$

$$X^Ty = (V\Sigma^TU^T)y$$

$$(X^TX)^{-1}X^Ty = (V\Sigma^T\Sigma V^T)^{-1}(V\Sigma^TU^T)y$$

2. $V^{-1} = V^T$ since V is orthogonal, then we have

$$(V\Sigma^T\Sigma V^T)^{-1}(V\Sigma^TU^T)y = V(\Sigma^T\Sigma)^{-1}V^T(V\Sigma^TU^T)y = V(\Sigma^T\Sigma)^{-1}\Sigma^TU^Ty$$

3. Now we can define $\Gamma = (\Sigma^T\Sigma)^{-1}\Sigma^T$, and get

$$(X^TX)^{-1}X^Ty = V\Gamma U^Ty$$

Problem 3

The pdf of Gaussian is:

$$p_u(x) = \frac{1}{(2\pi)^{d/2}} \exp(-\frac{1}{2}||x-u||^2)$$

Therefore the likelihood is:

$$L(u) = \Pi_{n=1}^{N} p_u(x_n) = \frac{1}{(2\pi)^{Nd/2}} \exp(-\frac{1}{2}\sum_{n=1}^{N} ||x_n-u||^2)$$

Take the log-likelihood:

$$\log L(u) = -\frac{Nd}{2}\log(2\pi) - \frac{1}{2}\sum_{n=1}^{N} ||x_n-u||^2$$

Since the first term is constant, maximizing $\log L(u)$ is equivalent to minimizing:

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

To minimize $f(u)$, we set it's gradient to zero:

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

And we get

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

Problem 4

Find a perceptron

The transformed data:

$$\begin{aligned}
\mathbf{z}_1 &= (1,1,1,1,1,1), & y_1 &= -1\\
\mathbf{z}_2 &= (1,-1,1,1,-1,1), & y_2 &= 1\\
\mathbf{z}_3 &= (1,-1,-1,1,1,1), & y_3 &= -1\\
\mathbf{z}_4 &= (1,1,-1,1,-1,1), & y_4 &= 1
\end{aligned}$$

From the data we can see that $y = \text{sign}(-x_1x_2)$.
We can easily set $\tilde{\mathbf{w}} = [0,0,0,0,-1,0]$, that make $\tilde{\mathbf{w}}^T\mathbf{z} = -x_1x_2$.

Plot the boundary

The boundary is at point that make $\tilde{\mathbf{w}}^T\Phi_2(\mathbf{x}) = -x_1x_2 = 0$, which happened when $x_1 = 0$ or $x_2 = 0$.

Therefore we have the boundary being two perpendicular lines crossing at the origin: $x_1 = 0$ and $x_2 = 0$

Problem 5

Let original weight be: $W = \sum_{n=1}^{N} w_n^t$
Then the total weight on misclassified is: $W_M = \varepsilon_t \cdot W$
and the total weight on correctly classified is: $W_C = (1-\varepsilon_t) \cdot W$

After update, the total weight become:

$$W = W_M \cdot d_t + W_C / d_t = \varepsilon_t \cdot W \cdot d_t + (1-\varepsilon_t) \cdot W / d_t$$

Therefore the new error rate become:

$$\begin{aligned}
\varepsilon_{t+1} &= \frac{\varepsilon_t \cdot W \cdot d_t}{\varepsilon_t \cdot W \cdot d_t + (1-\varepsilon_t) \cdot W / d_t} \\
&= \frac{\varepsilon_t \cdot d_t}{\varepsilon_t \cdot d_t + (1-\varepsilon_t)/d_t} \\
&= \frac{\varepsilon_t \cdot \sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}}}{\varepsilon_t \cdot \sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}} + (1-\varepsilon_t) \cdot \sqrt{\frac{\varepsilon_t}{1-\varepsilon_t}}} \\
&= \frac{\sqrt{\varepsilon_t(1-\varepsilon_t)}}{2\sqrt{\varepsilon_t(1-\varepsilon_t)}} \\
&= 0.5
\end{aligned}$$

hw3 Programming Part Report

Homework 3 (Programming Part Report)
b11902038 資工三 鄭博允

Implementation

LinearModel

We implement gradient descent for both linear and logistic regression.

```python
for _ in range(self.iterations):
    grad = self._compute_gradients(X, y)
    self.weights -= self.learning_rate * grad
```

However, we use different methods to compute weight and gradient.

linear regression

- the shape of `weights` is `(n_features,)`
- we use MSE to compute gradient:

  $$\nabla E_{\text{in}}(\mathbf{w}_t) = \frac{1}{N}\mathrm{X}^T(X\mathbf{w} - y)$$

  ```python
  return (X.T @ (X @ self.weights - y)) / len(y)
  ```

For logistic regression

- the shape of `weights` is `(n_features, n_classes)`.
- we use Cross-Entropy to compute gradient:

  $$\nabla E_{\text{in}}(\mathbf{w}_t) = \frac{1}{N}X^T(\sigma(X\mathbf{w}) - y)$$

where σ stands for softmax function.

```
y_pred = self._softmax(X @ self.weights)
return (X.T @ (y_pred - y)) / len(y)
```

- we also apply one-hot encoding on `y` to make sure it fit the shape of `pred_y` computed by `weights`.

DecisionTree

We implement the following algorithm to build a tree:

1. if exceed max depth or the dataset is pure, return a leaf
2. find branching criteria: `_find_best_split(X, y)`, where we traverse all possible thresholds and find splits with best purity.
3. split the dataset using branching criteria

   ```
   mask = X[:, feature] <= threshold
   left_X, left_y = X[mask], y[mask]
   right_X, right_y = X[~mask], y[~mask]
   ```

4. build subtrees by the splitted datasets

   ```
   left_child = self._build_tree(left_X, left_y, depth + 1)
   right_child = self._build_tree(right_X, right_y, depth + 1)
   ```

For step 2, we use Gini Index and MSE as different approach to calculate purity score.

- Gini Index:

$$\text{Gini}(D) = 1 - \sum_{k=1}^{K} p_k^2$$

  where K stands for the number of class: `np.unique(y)` and p stands for the probability of one class appear in the whole dataset: `np.sum(y == c) / len(y)`.

```
gini = 1 - np.sum([(np.sum(y == c) / len(y)) ** 2 for c in
np.unique(y)]) if len(y) > 0 else 0
```

- MSE:

$$\mathrm{MSE}(D) = \frac{1}{|D|} \sum_{i=1}^{|D|} (y_i - \bar{y})^2$$

```
mse = np.mean((y - np.mean(y)) ** 2) if len(y) > 0 else 0
```

RandomForest

Here's the algorithm for RandomForest:

1. init N `DecisionTree()` which we build previously
2. fit each tree with different datasets, generated by randomly select n samples in original dataset (size=n)

```
np.random.choice(n, n, replace=True)
```

3. to predict, return most frequent class in N trees for classification and average prediction value in N trees for regression

Comparision: models

Model	Accuracy (classification)	MSE (regression)
LinearModel	0.6667	43.4139
DecisionTree	0.8889	28.3416
RandomForest	**0.9333**	**24.3225**

For classification, the random forest model performs the best, followed by the decision tree, and then the linear model. This is because the Iris dataset is not linearly separable, and the nonlinear models (decision tree and random forest) can capture the complex relationships between the features. Random forests generally outperform single decision trees due to the ensemble method reducing overfitting.

For regression, the random forest also performs the best, followed by the decision tree, and then the linear model. The Boston housing dataset may also have non-linear relationships, which the tree-based models can capture better than a linear model. Again, the ensemble nature of random forests gives it an edge.

Comparision: preprocess techniques

Technique	Accuracy (classification)	MSE (regression)
Normalization	0.6667	43.4139
Standardization	0.8889	21.9085

We use linear model for this task. In this case, standardization seems to improve the performance of the linear model for both classification and regression. This suggests that the features in the Iris and Boston housing datasets may have different scales or may benefit from having a Gaussian-like distribution. Standardization often helps gradient descent converge faster, which is used in the linear model's training.

Comparision: hyperparameter (linear model)

learning_rate	n_iterations	Accuracy (classification)	MSE (regression)
0.01	1000	0.6667	43.4139
0.1	1000	0.9111	57.5492
0.01	100	0.6222	88.8652
0.1	100	0.6667	43.3975

- The best classification accuracy is achieved with a learning rate of 0.1 and 1000 iterations. The lowest MSE for regression with 100 iterations is very close to the MSE with 1000 iterations and learning rate 0.01.
- **Learning Rate:** A higher learning rate can lead to faster convergence but may also cause the algorithm to overshoot the optimal solution, resulting in higher MSE (as seen with lr=0.1, iterations=1000). A lower learning rate may lead to slower convergence but is more likely to find the optimal solution.
- **Number of Iterations:** More iterations generally lead to better convergence, but there is a point of diminishing returns. With too few iterations (e.g., iterations=100), the model may not converge, resulting in poor performance.

Comparision: hyperparameter (random tree)

n_estimators	max_depth	Accuracy (classification)	MSE (regression)
100	5	0.9111	23.9231
100	10	0.9333	23.7611
200	5	0.9333	24.0250
200	10	0.9333	23.0885

- **Number of Trees (n_estimators):** Increasing the number of trees generally improves performance by reducing overfitting and improving generalization. However, there is a point of diminishing returns, and increasing the number of trees also increases the computational cost. In this specific output, increasing from 100 to 200 trees does not significantly change the classification accuracy, but does improve the regression MSE in one case.
- **Maximum Depth (max_depth):** The maximum depth of the individual trees controls the complexity of the model. A larger maximum depth allows the trees to capture more complex relationships in the data, but it also increases the risk of overfitting. A smaller maximum depth can prevent overfitting but may lead to underfitting if the trees are not complex enough to capture the underlying patterns in the data. In this output, increasing the max_depth from 5 to 10 improves both classification and regression performance in most cases.

Strengths / Weakness Analysis

- **Linear Model (Linear/Logistic Regression):**
  - Strengths: Simple, interpretable, computationally efficient, works well for linearly separable data (for classification) and data with linear relationships (for regression).
  - Weaknesses: Cannot capture nonlinear relationships, performs poorly on complex datasets.
  - Scenarios: Use for problems where the relationship between the features and the target variable is linear, or when interpretability and computational efficiency are important.
- **Nonlinear Model (Decision Tree):**
  - Strengths: Can capture nonlinear relationships, relatively easy to understand, can handle both categorical and numerical data.
  - Weaknesses: Prone to overfitting, can be sensitive to small changes in the data, can be unstable.
  - Scenarios: Use for problems where the relationship between the features and the target variable is nonlinear, or when interpretability is important.
- **Random Forest Model:**
  - Strengths: Can capture nonlinear relationships, robust to outliers, less prone to overfitting than single decision trees, high accuracy.
  - Weaknesses: Less interpretable than single decision trees or linear models, computationally more expensive than linear models.
  - Scenarios: Use for complex problems where high accuracy and robustness are important, and interpretability is less of a concern.