Syed Muhammad Hamza Shah

CSC 546

Professor Liang

**Homework 3**

**1. Classification and Cross-entropy loss**

(1) The formula for binary cross-entropy loss (K=2) is:

L($yn$, $\hat{y}n$) = -[$yn$ * log($\hat{y}n$) + (1 - $yn$) * log(1 - $\hat{y}n$)]

where $yn$ is the ground-truth class label (0 or 1), and $\hat{y}n$ is the output probability/confidence of the logistic regression classifier for class-1.

(2) The formula for cross-entropy loss (K⩾2) is:

L($yn$, $\hat{y}n$) = - ∑_{k=1}^{K} $ynk$ * log($\hat{y}nk$)

where $yn$ is the one-hot encoded ground-truth class label, $\hat{y}n$ is the output probability/confidence vector from the logistic regression classifier, and K is the number of classes.

(3) Given $yn$ = 1 and $\hat{y}n$ = 0.8, we can compute the binary cross-entropy loss as follows:

L($yn$, $\hat{y}n$) = -[$yn$ * log($\hat{y}n$) + (1 - $yn$) * log(1 - $\hat{y}n$)]

L(1, 0.8) = -(1 * log(0.8) + (1 - 1) * log(1 - 0.8))

L(1, 0.8) = -log(0.8)

L(1, 0.8) $\approx$ 0.223

Thus, the binary cross-entropy loss associated with the single data sample $xn$ is approximately 0.223.

(4) Given $yn$ = 2 and $\hat{y}n$ = [0.1, 0.2, 0.7]^T, we first one-hot-encode $yn$:

$yn$ = [0, 0, 1]^T

Now we can compute the cross-entropy loss as follows:

$L(yn, \hat{y}n) = - \sum_{k=1}^{K} ynk * \log(\hat{y}nk)$

$L([0, 0, 1]^T, [0.1, 0.2, 0.7]^T) = -(0 * \log(0.1) + 0 * \log(0.2) + 1 * \log(0.7))$

$L([0, 0, 1]^T, [0.1, 0.2, 0.7]^T) = -\log(0.7)$

$L([0, 0, 1]^T, [0.1, 0.2, 0.7]^T) \approx 0.357$

Thus, the cross-entropy loss associated with the single data sample $xn$ is approximately 0.357.

(5)

To show that the function $f(x) = -\log(1/(1+e^{-x}))$ is convex in $x$, we need to show that its second derivative with respect to $x$ is non-negative (i.e., $\partial^2 f/\partial x^2 \geqslant 0$).

function $f(x)$:

$f(x) = -\log(1/(1+e^{-x}))$

$f(x) = \log(1+e^x)$

first and second derivatives of $f(x)$ with respect to $x$:

First derivative $\partial f/\partial x$:

Using the chain rule, we get:

$\partial f/\partial x = (e^x) / (1+e^x)$

Second derivative $\partial^2 f/\partial x^2$:

using the chain rule and quotient rule, we get:

$\partial^2 f/\partial x^2 = (e^x * (1+e^x) - (e^x)^2) / (1+e^x)^2$

Simplifying the expression:

$\partial^2 f/\partial x^2 = (e^x - (e^x)^2) / (1+e^x)^2$

$\partial^2 f/\partial x^2 = e^x(1 - e^x) / (1+e^x)^2$

$\partial^2 f/\partial x^2 \geqslant 0$. Since $(1+e^x)^2$ is always positive, we only need to show that the numerator is non-negative:

$e^x(1 - e^x) \geqslant 0$

For $x > 0$, $e^x > 1$, and $(1 - e^x) < 0$, which makes the product negative. Considering the natural logarithm, the domain of $f(x)$ is $(0, \infty)$. So the function $f(x) = -\log(1/(1+e^{(-x)}))$ is convex in $x$.

**2. Regression**

(1) Mean Squared Error (MSE) Loss:

$MSE = (1/N) * \sum_{n=1}^{N} \sum_{i=1}^{2} (yni - \hat{y}ni)^2$

where N is the number of data samples, $yni$ is the i-th element of the ground-truth vector $yn$, and $\hat{y}ni$ is the i-th element of the predicted vector $\hat{y}n$.

(2) Mean Absolute Error (MAE) Loss:

$MAE = (1/N) * \sum_{n=1}^{N} \sum_{i=1}^{2} |yni - \hat{y}ni|$

(3) Mean Absolute Percentage Error (MAPE) Loss:

$MAPE = (100/N) * \sum_{n=1}^{N} \sum_{i=1}^{2} |(yni - \hat{y}ni) / yni|$

**3.**

(1) A decision tree for regression looks like a staircase because each leaf node represents a constant value for a specific region in the input space. The input space is divided into regions

based on splitting criteria, and the output value for each region is the average of the target values in that region. This results in a piecewise constant function that looks like a staircase.

(2) Building a deep tree such that every leaf-node of the tree is a pure node can lead to overfitting, which means the model would be too complex and would not generalize well to new, unseen data.

(3) The total number of training samples according to the above tree is 60, which is given at Node-0.

(4) The max-depth of the tree is 2.

(5) The pure nodes are Node-2 and Node-4. The entropy of Node:

For Node-0: [10, 20, 30]

p(1) = 10/60 = 1/6, p(2) = 20/60 = 1/3, p(3) = 30/60 = 1/2

Entropy(Node-0) = -[(1/6) * log2(1/6) + (1/3) * log2(1/3) + (1/2) * log2(1/2)] ≈ 1.459

(6) The number of leaf/terminal nodes in the tree is 3: Node-2, Node-3, and Node-4.

**4. Bagging and Random Forest**

(1) Bagging is an ensemble technique that reduces model variance by averaging the predictions of multiple base learners trained on different subsets of the training data. It works well for base learners with high variance and low bias, whose errors are not strongly correlated, and when there is sufficient data to create multiple subsets for training.

(2) The strategy is called "feature bagging" or "random subspace method," which limits the number of features considered for splitting at each node and reduces correlation between trees by forcing them to make decisions based on different sets of features.

**5. Boosting**

Boosting aims to reduce both bias and variance by sequentially training multiple base learners, with each learner correcting the errors of the previous one. Bagging mainly targets variance, while boosting targets both variance and bias. Random Forest is an example of bagging that reduces variance by averaging predictions of deep decision trees, while XGBoost is a boosting algorithm that reduces both bias and variance by iteratively correcting errors and combining shallow decision trees into a strong ensemble model.

**6. Stacking**

(1) Stacking and bagging are both ensemble methods that combine multiple models to improve predictive performance, but they differ in how they combine the models. Bagging involves training multiple models independently on random subsets of the data and averaging their predictions, while stacking trains a meta-model to combine the predictions of multiple base models.

(2) Stacking many polynomial models of the same degree may not be useful because they may all have similar biases and may not provide diverse enough predictions to improve performance. It may be better to stack models with different biases or to use a different ensemble method that allows for more diverse models.

(3) It can be useful to stack models of different types/structures because they may capture different aspects of the data or have different strengths and weaknesses. By combining these models, we can leverage their diverse predictions to improve overall performance. It is important to carefully select and train the base models to ensure they complement each other well and avoid overfitting.

**7. Overfitting and Underfitting**

Consider two scenarios in a classification task:

(1) the training accuracy is 100% and the testing accuracy is 50%

(2) the training accuracy is 80% and the testing accuracy is 70%

In the first scenario, where the training accuracy is 100% and the testing accuracy is 50%, overfitting is likely present. This is because the model has learned to perfectly fit the training data, including its noise and outliers, but fails to generalize to new, unseen data, resulting in poor performance on the testing set.

In the second scenario, where the training accuracy is 80% and the testing accuracy is 70%, overfitting is less likely to be present. This is because the model is not able to fit the training data perfectly, indicating that it has not learned to memorize the training data, which may allow it to better generalize to new data.

Consider two new scenarios in a classification task:

(1) the training accuracy is 80% and the testing accuracy is 70%

(2) the training accuracy is 50% and the testing accuracy is 50%

In which scenario is underfitting likely present?

In the first new scenario, where the training accuracy is 80% and the testing accuracy is 70%, underfitting is less likely to be present. This is because the model is able to achieve reasonable performance on the testing data, indicating that it has learned some of the underlying patterns in the data.

In the second new scenario, where the training accuracy is 50% and the testing accuracy is 50%, underfitting is more likely present. This is because the model is not able to learn even the most basic patterns in the data, resulting in poor performance on both the training and testing sets.

## 8. Training, Validation, and Testing for Classification and Regression

(1) Hyperparameters are settings of a model that are predetermined by the practitioner and control various aspects of the model's behavior, such as its complexity or regularization strength. Examples include the number of hidden layers in a neural network and the learning rate in gradient descent.

(2) A validation set is used to evaluate model performance and prevent overfitting during the model development process. It allows us to compare the performance of different models and hyperparameters and estimate a model's ability to generalize to unseen data without using the test set.

(3) Optimizing hyperparameters on the training set can lead to overfitting, so a validation set is used to ensure that the chosen hyperparameters and model generalize well to new data.

(4) The testing set should not be used to optimize hyperparameters because doing so can result in "leakage" of information from the test set into the model development process and lead to overly optimistic estimates of the model's performance on unseen data. The test set should only be used to provide an unbiased estimate of the model's performance after all model development and hyperparameter tuning processes have been completed.

**9. SVM**

(1) Maximizing the margin in SVM improves robustness against noise by finding a decision boundary with the largest distance to the closest data points. This reduces overfitting and improves generalization.

(2) Nonlinear SVM uses a kernel function to map data to a higher-dimensional feature space where the margin is maximized. This captures complex relationships between features and target variables.

(3) SVM can cause "out-of-memory" errors for large datasets because the quadratic programming problem required for training involves computing and storing the entire kernel matrix.

(4) A kernel function in SVM transforms data into a higher-dimensional feature space where it may become linearly separable, allowing for more complex decision boundaries and capturing nonlinear relationships.