CSE 5819 Assignment #4

by: Aayushi Verma (uef24001)

This assignment is organized as follows:

1 Part 1: ChatGPT Self-Learning (20pts)

You can start from the following prompts, but you need to create subsequent questions that attempt to understand the basic concepts of machine learning.

1.1 What is overfitting? What is underfitting?

Overfitting occurs when a machine learning model learns not only the underlying pattern in the training data but also the noise and irrelevant details, resulting in a model that performs well on the training set but poorly on unseen data. Overfitting happens when the model is too complex, often with too many parameters relative to the amount of training data, leading to low training error but high testing error.

Underfitting, on the other hand, occurs when a model is too simple to capture the underlying pattern in the data, leading to poor performance on both the training and testing sets. Underfitting typically happens when the model has insufficient complexity or when important features of the data are not considered, resulting in both high training and testing errors.

1.2 How does logistic regression handle overfitting?

Logistic regression can handle overfitting through the use of **regularization techniques**. Regularization adds a penalty to the loss function to discourage complex models that overfit the training data. Two common regularization techniques are:

- **L2 regularization** (also known as Ridge regression): Adds a penalty proportional to the square of the magnitude of the model coefficients. This helps to reduce the complexity of the model by shrinking the coefficients, effectively limiting the influence of less important features.
- **L1 regularization** (also known as Lasso regression): Adds a penalty proportional to the absolute value of the model coefficients. This can lead to sparse models by driving some coefficients to zero, effectively performing feature selection and reducing overfitting.

By tuning the regularization strength, logistic regression can achieve a balance between bias and variance, helping to prevent overfitting while maintaining good generalization to unseen data.

1.3 How does least squares handle overfitting?

Least squares regression, like logistic regression, can suffer from overfitting when the model is too complex relative to the data, leading to a good fit on the training data but poor generalization to unseen data. To handle overfitting in least squares regression, several techniques are employed:

- **Regularization**: Adding a regularization term to the least squares loss function can help reduce overfitting by penalizing large coefficients.
  - **Ridge regression** (L2 regularization): This modifies the least squares objective by adding a penalty proportional to the sum of the squared coefficients, which discourages overly complex models by shrinking the magnitude of the coefficients.
  - **Lasso regression** (L1 regularization): This adds a penalty proportional to the sum of the absolute values of the coefficients, which can drive some coefficients to zero, effectively simplifying the model by selecting a subset of the most important features.
- **Cross-validation**: Cross-validation techniques, such as k-fold cross-validation, can be used to tune the model complexity by selecting hyperparameters (e.g., the regularization strength) that prevent overfitting.
- **Feature selection**: By selecting only the most relevant features or using dimensionality reduction techniques (such as Principal Component Analysis, PCA), the model can be simplified, reducing the risk of overfitting.

By incorporating these techniques, least squares regression can achieve a balance between fitting the data and avoiding overfitting, improving its performance on unseen data.

1.4 What regularizers can be used to regularize machine learning algorithms?

Regularizers are techniques used to prevent overfitting in machine learning algorithms by adding a penalty to the loss function. Some common regularizers include:

- **L2 Regularization** (Ridge): Adds a penalty proportional to the sum of the squared values of the model's coefficients. This encourages smaller coefficient values, which reduces model complexity while retaining all features. It is commonly used in linear and logistic regression.
- **L1 Regularization** (Lasso): Adds a penalty proportional to the sum of the absolute values of the model's coefficients. This can result in some coefficients being set to zero, effectively performing feature selection, which simplifies the model.
- **Elastic Net**: Combines both L1 and L2 regularization, balancing the penalties of both Ridge and Lasso. This approach allows for both feature selection and coefficient shrinkage, making it useful when there are many correlated features.
- **Dropout** (for neural networks): Randomly drops a fraction of neurons during each training iteration, preventing co-adaptation of neurons and reducing overfitting in deep learning models.
- **Early Stopping**: Monitors the performance of the model on a validation set during training, stopping the training process when performance starts to degrade. This prevents the model from learning noise in the training data.
- **Data Augmentation**: Although not a direct regularization term, augmenting the training data with transformations (such as rotations or flips in image data) increases the diversity of the training set, reducing the chance of overfitting.
- **Weight Decay**: Equivalent to L2 regularization, weight decay explicitly penalizes large weights during training, commonly used in neural networks.
- **Norm-based Regularization**: Adds constraints on the norms of the weight vectors, such as imposing an upper bound on the L2 norm of the weights, which can help reduce model complexity.

These regularizers help machine learning models generalize better by controlling model complexity and reducing overfitting to the training data.

1.5 How to compute the gradient of the least squares loss?

The least squares loss function is commonly used in linear regression and is defined as:

$$L(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^\top \mathbf{w})^2$$

where:

- $\mathbf{x}_i \in \mathbb{R}^d$ is the feature vector for the i-th sample,
- $y_i \in \mathbb{R}$ is the observed target value for the i-th sample,
- $\mathbf{w} \in \mathbb{R}^d$ is the weight vector (the model's parameters),
- n is the number of samples in the dataset.

To compute the gradient of the least squares loss with respect to the weight vector $\mathbf{w}$, we follow these steps:

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = -\sum_{i=1}^{n}\mathbf{x}_i(y_i - \mathbf{x}_i^\top \mathbf{w})$$

Alternatively, in matrix form, the loss function can be written as:

$$L(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w})$$

where $\mathbf{X} \in \mathbb{R}^{n\times d}$ is the matrix of feature vectors and $\mathbf{y} \in \mathbb{R}^n$ is the vector of target values.
The gradient of the least squares loss with respect to $\mathbf{w}$ is:

$$\nabla_{\mathrm{w}} L(\mathbf{w}) = \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y})$$

This gradient provides the direction of the steepest ascent for the loss function, and in gradient descent, we use it to update the weights as follows:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathrm{w}} L(\mathbf{w})$$

where η is the learning rate.

Thus, the gradient of the least squares loss function is computed as $\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y})$, which is then used to iteratively update the weight vector in gradient-based optimization methods.

1.6 How to compute the gradient of the logistic loss?

The logistic loss (or log-loss) is commonly used in logistic regression for binary classification. Given a set of training samples $(\mathbf{x}_i, y_i)$ where $y_i \in \{0, 1\}$, the logistic loss for a linear model $\mathbf{w} \in \mathbb{R}^d$ is defined as:

$$L(\mathbf{w}) = -\sum_{i=1}^{n} [y_i \log(p(\mathbf{x}_i)) + (1 - y_i) \log(1 - p(\mathbf{x}_i))]$$

where $p(\mathbf{x}_i) = \sigma(\mathbf{x}_i^\top \mathbf{w})$ is the predicted probability of the positive class, and $\sigma(z)$ is the sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Gradient of the Logistic Loss

The gradient of the logistic loss with respect to the weight vector $\mathbf{w}$ can be computed as follows:

1. *Predicted probability*: The probability of the positive class is $p(\mathbf{x}_i) = \sigma(\mathbf{x}_i^\top \mathbf{w})$. 2. *Gradient of sigmoid*: The derivative of the sigmoid function $\sigma(z)$ with respect to z is $\sigma(z)(1 - \sigma(z))$.

The partial derivative of the logistic loss with respect to $\mathbf{w}$ is:

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = \sum_{i=1}^{n} \left[\sigma(\mathbf{x}_i^\top \mathbf{w}) - y_i\right] \mathbf{x}_i$$

In matrix form, where $\mathbf{X} \in \mathbb{R}^{n \times d}$ is the matrix of feature vectors and $\mathbf{y} \in \mathbb{R}^n$ is the vector of true labels, this gradient can be written as:

$$\nabla_{\mathrm{w}} L(\mathbf{w}) = \mathbf{X}^\top \left(\sigma(\mathbf{X}\mathbf{w}) - \mathbf{y}\right)$$

where $\sigma(\mathbf{X}\mathbf{w})$ is the vector of predicted probabilities for all samples, and $\mathbf{X}^\top$ is the transpose of the feature matrix.

Gradient Descent Update

Once the gradient is computed, it is used to update the weight vector in gradient descent as follows:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathrm{w}} L(\mathbf{w})$$

where η is the learning rate.

Thus, the gradient of the logistic loss is $\mathbf{X}^\top(\sigma(\mathbf{X}\mathbf{w}) - \mathbf{y})$, and this gradient is used to iteratively optimize the weight vector in logistic regression.

2 Problems

2.1 Question One

[Gradient Descent]

[20pts] You have a univariate function you wish to minimize, $f(w) = 5(w-11)^4$. Suppose you wish to perform gradient descent with constant step size $\alpha = 1/40$. Starting with $w_0 = 13$, perform 2 steps of gradient descent. What are $w_0, ..., w_2$?

2.2 Question Two

[Logistic Regression]

In this problem, we are going to assume the same notation setup in class. For logistic regression, we model the class probability by

$$P(y|\mathbf{x}_i) = \sigma(y(\mathbf{w}^T\mathbf{x}_i))$$

where we define the *logistic function* $\sigma : \mathbb{R} \to \mathbb{R}$ as

$$\sigma(s) = \frac{1}{1+e^{-s}}$$

(Note: We dropped the bias term b since we can always absorb the bias into $\mathbf{w}$ by representing $\mathbf{x}$ in homogeneous coordinates.)

1. [20 pts] Prove that the given σ is a properly defined probabilistic model. In other words, prove that $P(y = +1|\mathrm{x_i}) + P(y = -1|\mathrm{x_i}) = 1$

   (Hint: use the relationship between $\sigma(s)$ and $\sigma(-s)$.)

2.3 Question Three

2. [20 pts] Show that the gradient of the log likelihood function, namely, $\nabla_w \log P(\mathbf{y}|X, \mathbf{w})$ is

$$\sum_{i=1}^{n}(1 - \sigma(y_i(\mathbf{w}^T\mathbf{x}_i)))y_i\mathbf{x}_i$$

where y_i is a scalar, and x_i is a vector of descriptors, and $y = [y_1, y_2, \cdots, y_n]^T$ is a vector, and $X = [x_1, x_2, \cdots, x_n]$ is a matrix indicating all data.