Lab Course Machine Learning

Exercise Sheet 3

Prof. Dr. Dr. Lars Schmidt-Thieme
Jung Min Choi

1 Newton's Method

Newton's Method is an iterative optimization algorithm used to find the minimum or maximum of a function. It is particularly useful for convex functions. The method uses the second-order derivative (Hessian matrix) to adjust the step size, leading to faster convergence compared to first-order methods.

Algorithm Given a function $f(\theta)$, the update rule for Newton's Method is:

$$\theta^{(t+1)} = \theta^{(t)} - H^{-1}\nabla f(\theta),$$

where:

- $\nabla f(\theta)$ is the gradient of $f(\theta)$, representing the vector of partial derivatives:

$$\nabla f(\theta) = \begin{bmatrix} \frac{\partial f}{\partial \theta_1} \\ \frac{\partial f}{\partial \theta_2} \\ \vdots \\ \frac{\partial f}{\partial \theta_n} \end{bmatrix}.$$

- H is the Hessian matrix of $f(\theta)$, a square matrix of second-order partial derivatives:

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial \theta_1^2} & \frac{\partial^2 f}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 f}{\partial \theta_1 \partial \theta_n} \\ \frac{\partial^2 f}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 f}{\partial \theta_2^2} & \cdots & \frac{\partial^2 f}{\partial \theta_2 \partial \theta_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial \theta_n \partial \theta_1} & \frac{\partial^2 f}{\partial \theta_n \partial \theta_2} & \cdots & \frac{\partial^2 f}{\partial \theta_n^2} \end{bmatrix}.$$

The Hessian matrix encapsulates the curvature of the function, providing information about how the gradient changes. Using the Hessian, Newton's Method adjusts the step size and direction to move more efficiently towards the minimum or maximum.

Details of the Update Rule The update rule:

$$\theta^{(t+1)} = \theta^{(t)} - H^{-1}\nabla f(\theta),$$

can be interpreted as follows:

- $H^{-1}\nabla f(\theta)$ computes a scaled adjustment to the parameters θ, taking into account both the gradient and the curvature of the function.
- The inverse of the Hessian, H^{-1}, adjusts the magnitude and direction of the step based on the curvature. If the curvature is steep, smaller steps are taken to prevent overshooting.

Newton's Method assumes that H is invertible and that the function $f(\theta)$ is twice differentiable.

2 Gradient of MSE and Cross-Entropy Loss

A Gradient of MSE The Mean Squared Error (MSE) loss is defined as:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

where y_i is the true label and $\hat{y}_i$ is the predicted value, which can be expressed as $\hat{y}_i = X_i\theta$ for a linear model.
To compute the gradient of $L(\theta)$ with respect to θ, we follow these steps: 1. Expand the squared term:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - X_i\theta)^2.$$

2. Differentiate $L(\theta)$ with respect to θ:

$$\frac{\partial L(\theta)}{\partial \theta} = \frac{1}{n}\sum_{i=1}^{n} 2\,(y_i - X_i\theta)\,(-X_i).$$

3. Rewrite in matrix form:

$$\nabla L(\theta) = -\frac{2}{n}X^T(y - \hat{y}),$$

where X is the input feature matrix, y is the vector of true labels, and $\hat{y} = X\theta$ is the vector of predictions.

B Gradient of Cross-Entropy Loss The Cross-Entropy Loss for binary classification is defined as:

$$L(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right],$$

where $\hat{y}_i = \sigma(z_i)$ and $z_i = X_i\theta$, with $\sigma(z)$ representing the sigmoid function:

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

To compute the gradient of $L(\theta)$ with respect to θ, we proceed as follows:

1. **Binary Cross-Entropy Loss and Its Derivative:**
   The binary cross-entropy loss function is defined as:

   $$L(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right],$$

   where $\hat{y}_i$ is the predicted probability (output of the sigmoid function), and $y_i \in \{0, 1\}$ is the true label.
   Taking the derivative of $L(\theta)$ with respect to $\hat{y}_i$ gives:

   $$\frac{\partial L(\theta)}{\partial \hat{y}_i} = -\frac{1}{n}\left[\frac{y_i}{\hat{y}_i} - \frac{1 - y_i}{1 - \hat{y}_i}\right].$$

2. **Substituting the Sigmoid Function:**
   The sigmoid function is defined as:

   $$\sigma(z_i) = \frac{1}{1 + e^{-z_i}}, \quad \text{and its complement is} \quad 1 - \sigma(z_i) = \frac{e^{-z_i}}{1 + e^{-z_i}}.$$

   Substituting $\hat{y}_i = \sigma(z_i)$ into the derivative, we get:

   $$\frac{\partial L(\theta)}{\partial \hat{y}_i} = -\frac{1}{n}\left[\frac{y_i}{\sigma(z_i)} - \frac{1 - y_i}{1 - \sigma(z_i)}\right].$$

3. **Simplify Using Sigmoid Properties:**
   Using the sigmoid function properties:

   $$\frac{1}{\sigma(z_i)} = 1 + e^{-z_i}, \quad \text{and} \quad \frac{1}{1 - \sigma(z_i)} = 1 + e^{z_i},$$

   substitute these back into the derivative:

   $$\frac{\partial L(\theta)}{\partial \hat{y}_i} = -\frac{1}{n}\left[y_i(1 + e^{-z_i}) - (1 - y_i)(1 + e^{z_i})\right].$$

4. **Expand and Group Terms:**

   Expand the terms inside the brackets:

$$\frac{\partial L(\theta)}{\partial \hat{y}_i} = -\frac{1}{n}\left[y_i + y_i e^{-z_i} - (1-y_i) - (1-y_i)e^{z_i}\right].$$

   Distribute $(1-y_i)$:

$$\frac{\partial L(\theta)}{\partial \hat{y}_i} = -\frac{1}{n}\left[y_i + y_i e^{-z_i} - 1 + y_i - e^{z_i} + y_i e^{z_i}\right].$$

5. **Simplify Further:**

   Group terms involving y_i and constants:

$$\frac{\partial L(\theta)}{\partial \hat{y}_i} = -\frac{1}{n}\left[(y_i + y_i) + y_i e^{-z_i} + y_i e^{z_i} - 1 - e^{z_i}\right].$$

   Using the symmetry of the sigmoid function, terms involving e^{z_i} and e^{-z_i} cancel out. This results in:

$$\frac{\partial L(\theta)}{\partial \hat{y}_i} = \frac{1}{n}(\hat{y}_i - y_i).$$

3 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is an optimization algorithm where updates to the model parameters are performed using individual data points or small batches instead of the entire dataset. This allows for faster iterations, particularly for large datasets.

Algorithm Given a loss function $L(\theta)$, the update rule for SGD is:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla L_i(\theta),$$

where:

- η is the learning rate,
- $\nabla L_i(\theta)$ is the gradient of the loss function computed for a single data point i.

Example Consider a dataset with $n = 3$ samples, where the feature matrix X and the true labels y are given as:

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

Assume the model is a linear regression model where the predictions are:

$$\hat{y} = X\theta,$$

and the Mean Squared Error (MSE) loss is:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

Let the initial parameter vector $\theta^{(0)}$ be:

$$\theta^{(0)} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \end{bmatrix},$$

and the learning rate $\eta = 0.01$.

Step-by-Step SGD Iteration 1. **Select a Data Point**: Randomly select one data point from the dataset. For instance, pick the first data point:

$$x^{(1)} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}, \quad y^{(1)} = 1.$$

2. **Compute the Prediction**:

$$\hat{y}^{(1)} = x^{(1)}\theta^{(0)} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \end{bmatrix} = 1.4.$$

3. **Compute the Loss Gradient**: The gradient of the MSE loss for this data point is:

$$\nabla L_1(\theta) = -2\left(y^{(1)} - \hat{y}^{(1)}\right) x^{(1)T}.$$

Substitute the values:

$$\nabla L_1(\theta) = -2\,(1 - 1.4) \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 1.6 \\ 2.4 \end{bmatrix}.$$

4. **Update the Parameters**: Using the SGD update rule:

$$\theta^{(1)} = \theta^{(0)} - \eta \nabla L_1(\theta).$$

Substitute the values:

$$\theta^{(1)} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \end{bmatrix} - 0.01 \begin{bmatrix} 0.8 \\ 1.6 \\ 2.4 \end{bmatrix} = \begin{bmatrix} 0.092 \\ 0.184 \\ 0.276 \end{bmatrix}.$$

5. **Repeat**: Select the next data point (randomly or sequentially) and repeat the above steps until convergence or after a fixed number of iterations.

Final Remarks This simple example demonstrates how SGD operates on a single data point at each iteration. While the update here is performed for one sample, small batches of data (mini-batch SGD) are often used in practice to strike a balance between computational efficiency and convergence stability.

4 Akaike Information Criterion (AIC)

The **Akaike Information Criterion (AIC)** is a widely used metric for evaluating and comparing models in statistics and machine learning. It provides a way to balance how well a model fits the data (goodness of fit) against its complexity (number of parameters). AIC helps in selecting the model that is both accurate and not overly complex, avoiding overfitting.

A Formula for AIC

$$\text{AIC} = 2k - 2\ln\left(\hat{L}\right),$$

where:

- k: The number of parameters in the model.
- $\hat{L}$: The maximum value of the likelihood function for the model.

B Components of AIC

1. **Model Complexity ($2k$):**
   - The term $2k$ penalizes the model for having too many parameters.
   - Adding more parameters increases the flexibility of the model, but overly complex models are more likely to overfit the data.
   - This part ensures that simpler models are preferred when they fit the data similarly to more complex models.
2. **Goodness of Fit ($-2\ln\left(\hat{L}\right)$):**
   - The likelihood function ($\hat{L}$) measures how well the model fits the data.
   - A higher likelihood (better fit) reduces the AIC value, favoring the model.

C Interpreting AIC

- **Lower AIC values** indicate a model that achieves a better trade-off between goodness of fit and simplicity.
- When comparing multiple models, the model with the lowest AIC is usually considered the best choice for the data.
- AIC does not provide an absolute "score" for a single model but rather helps to compare multiple models relative to one another.

D Pseudo-Code for AIC

```
Inputs:
    k: Number of parameters in the model
    L: Maximum likelihood of the model

Steps:
    1. Compute the penalty term:
       Penalty = 2 * k
    2. Compute the goodness-of-fit term:
       GoodnessOfFit = -2 * log(L)
    3. Calculate the final AIC value:
       AIC = Penalty + GoodnessOfFit
Output:
    AIC
```

5 Backward Feature Selection

Backward Feature Selection is an iterative approach to select important features by starting with all features and gradually removing the least significant ones.

Steps

1. Fit a model with all features.
2. Compute the performance metric (e.g., AIC).
3. Remove the feature that has the least impact on performance.
4. Repeat until the desired number of features remains.

Advantages

- Reduces model complexity.
- Identifies the most relevant features.

Pseudo-Code for Backward Feature Selection

```
Inputs:
    X: Dataset with all features
    y: Target variable
    metric_function: Performance metric to evaluate the model (e.g., AIC)
    min_features: Minimum number of features to retain

Steps:
    1. Initialize selected_features as all features in X.
    2. While the number of selected_features > min_features:
        a. Fit the model using selected_features.
        b. Compute the performance metric for the model.
        c. For each feature in selected_features:
           i. Temporarily remove the feature.
           ii. Fit the model and compute the performance metric.
        d. Identify the feature whose removal results in the least increase
```

```
            in the performance metric (or the largest improvement).
        e. Remove that feature from selected_features.
    3. Return the final selected_features.

Output:
    selected_features
```