Linear Regression and Gradient Descent

CS145: Introduction to Data Mining

Spring 2024

Contents

1 Introduction

Linear regression is a fundamental supervised learning technique used for predicting a continuous target variable based on one or more input features. In this document, we will discuss the linear regression model, its closed-form solution, and optimization techniques such as batch gradient descent, mini-batch stochastic gradient descent, and the impact of learning rate. We will also cover preprocessing techniques like z-score normalization and polynomial feature augmentation.

2 Linear Regression Model

Given a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, where $\boldsymbol{x}_i \in \mathbb{R}^D$ is the input feature vector and $y_i \in \mathbb{R}$ is the corresponding target value, linear regression aims to learn a linear function $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$ that best approximates the relationship between the input features and the target variable.

The parameters of the linear regression model are the weight vector $\boldsymbol{w} \in \mathbb{R}^D$ and the bias term $b \in \mathbb{R}$. The goal is to find the optimal values of these parameters that minimize the difference between the predicted values $f(\boldsymbol{x}_i)$ and the true target values y_i.

2.1 Objective Function: Mean Squared Error (MSE)

The most common objective function used in linear regression is the Mean Squared Error (MSE), which measures the average squared difference between the predicted and true target values:

$$\mathrm{MSE}(\boldsymbol{w}, b) = \frac{1}{N}\sum_{i=1}^N (f(\boldsymbol{x}_i) - y_i)^2 = \frac{1}{N}\sum_{i=1}^N (\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)^2 \tag{1}$$

The goal is to find the optimal weight vector $\boldsymbol{w}^*$ and bias term b^* that minimize the MSE:

$$(\boldsymbol{w}^*, b^*) = \operatorname*{argmin}_{\boldsymbol{w}, b} \mathrm{MSE}(\boldsymbol{w}, b) \tag{2}$$

2.2 Closed-Form Solution

Linear regression has a closed-form solution that can be obtained by setting the gradient of the MSE with respect to the parameters to zero and solving the resulting system of linear equations.

To simplify the notation, we can augment the input feature vectors with an additional dimension of ones to account for the bias term: $\boldsymbol{x}_i \leftarrow [\boldsymbol{x}_i; 1]$. This allows us to represent the linear function as $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$, where $\boldsymbol{w} \in \mathbb{R}^{D+1}$ now includes the bias term.

Let $\boldsymbol{X} \in \mathbb{R}^{N \times (D+1)}$ be the matrix of augmented input feature vectors, and $\boldsymbol{y} \in \mathbb{R}^N$ be the vector of target values. The closed-form solution for the optimal weight vector $\boldsymbol{w}^*$ is given by:

$$\boldsymbol{w}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \tag{3}$$

This solution is derived by setting the gradient of the MSE with respect to $\boldsymbol{w}$ to zero:

$$\begin{align} \nabla_{\boldsymbol{w}} \mathrm{MSE}(\boldsymbol{w}) = \frac{2}{N}\sum_{i=1}^N (\boldsymbol{w}^\top \boldsymbol{x}_i - y_i)\boldsymbol{x}_i = \frac{2}{N}\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) = 0 \tag{4}\\ \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{w} = \boldsymbol{X}^\top \boldsymbol{y} \tag{5}\\ \boldsymbol{w}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \tag{6} \end{align}$$

The closed-form solution provides the optimal weight vector in a single step, without the need for iterative optimization. However, it requires computing the inverse of the matrix $\boldsymbol{X}^\top \boldsymbol{X}$, which can be computationally expensive for large datasets or high-dimensional feature spaces.

3 Gradient Descent Optimization

Gradient descent is an iterative optimization algorithm used to find the minimum of a differentiable objective function, such as the MSE in linear regression. It updates the parameters in the opposite direction of the gradient of the objective function, with a step size determined by the learning rate.

3.1 Batch Gradient Descent

Batch gradient descent computes the gradient of the MSE with respect to the parameters using the entire training dataset in each iteration. The update rules for the weight vector $\boldsymbol{w}$ and bias term b are:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \alpha \nabla_{\boldsymbol{w}} \text{MSE}(\boldsymbol{w}^{(t)}, b^{(t)}) \tag{7}$$

$$b^{(t+1)} = b^{(t)} - \alpha \frac{\partial}{\partial b} \text{MSE}(\boldsymbol{w}^{(t)}, b^{(t)}) \tag{8}$$

where α is the learning rate, and t denotes the iteration number.

The gradients of the MSE with respect to $\boldsymbol{w}$ and b are computed as:

$$\nabla_{\boldsymbol{w}} \text{MSE}(\boldsymbol{w}, b) = \frac{2}{N} \sum_{i=1}^{N} (\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_i + b - y_i) \boldsymbol{x}_i \tag{9}$$

$$\frac{\partial}{\partial b} \text{MSE}(\boldsymbol{w}, b) = \frac{2}{N} \sum_{i=1}^{N} (\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_i + b - y_i) \tag{10}$$

Batch gradient descent iteratively updates the parameters until convergence or a maximum number of iterations is reached. It guarantees convergence to the global minimum for convex objective functions like the MSE, but it can be computationally expensive for large datasets as it requires computing the gradients over the entire dataset in each iteration.

3.2 Mini-Batch Stochastic Gradient Descent

Mini-batch stochastic gradient descent (SGD) is a variant of gradient descent that computes the gradients using a randomly selected subset (mini-batch) of the training dataset in each iteration. The update rules for the parameters are the same as in batch gradient descent, but the gradients are computed using only the mini-batch:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \alpha \nabla_{\boldsymbol{w}} \text{MSE}(\boldsymbol{w}^{(t)}, b^{(t)}; \mathcal{B}_t) \tag{11}$$

$$b^{(t+1)} = b^{(t)} - \alpha \frac{\partial}{\partial b} \text{MSE}(\boldsymbol{w}^{(t)}, b^{(t)}; \mathcal{B}_t) \tag{12}$$

where $\mathcal{B}_t$ is the mini-batch of size m randomly sampled from the training dataset at iteration t.

The gradients of the MSE with respect to $\boldsymbol{w}$ and b using the mini-batch are computed as:

$$\nabla_{\boldsymbol{w}} \text{MSE}(\boldsymbol{w}, b; \mathcal{B}_t) = \frac{2}{m} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{B}_t} (\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_i + b - y_i) \boldsymbol{x}_i \tag{13}$$

$$\frac{\partial}{\partial b} \text{MSE}(\boldsymbol{w}, b; \mathcal{B}_t) = \frac{2}{m} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{B}_t} (\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_i + b - y_i) \tag{14}$$

Mini-batch SGD provides a balance between the computational efficiency of stochastic gradient descent (which uses a single example in each iteration) and the stability of batch gradient descent. It allows for faster convergence and can escape local minima more easily compared to batch gradient descent. The choice of the mini-batch size m is a hyperparameter that needs to be tuned.

3.3 Impact of Learning Rate

The learning rate α is a critical hyperparameter in gradient descent optimization. It determines the step size at which the parameters are updated in each iteration. The choice of the learning rate can significantly impact the convergence speed and the quality of the solution:

- If the learning rate is too small, the optimization process will converge slowly, requiring many iterations to reach the minimum.
- If the learning rate is too large, the optimization process may overshoot the minimum, leading to oscillations or divergence.

Choosing an appropriate learning rate is essential for efficient and stable convergence. Common strategies for setting the learning rate include:

- Fixed learning rate: Using a constant learning rate throughout the optimization process.
- Learning rate schedules: Gradually decreasing the learning rate over time, e.g., using a decaying function such as exponential decay or step decay.
- Adaptive learning rates: Automatically adjusting the learning rate based on the optimization progress, e.g., using algorithms like AdaGrad, RMSprop, or Adam.

It is often recommended to experiment with different learning rates and observe the convergence behavior to find a suitable value for the problem at hand.

4 Preprocessing Techniques

4.1 Z-score Normalization

Z-score normalization, also known as standardization, is a preprocessing technique that transforms the input features to have zero mean and unit variance. It is commonly used to scale the features to a similar range and improve the convergence of gradient-based optimization algorithms.

Given a feature x, the z-score normalization is computed as:

$$x' = \frac{x - \mu}{\sigma} \tag{15}$$

where μ is the mean of the feature and σ is its standard deviation.

To apply z-score normalization to a dataset, the mean and standard deviation of each feature are computed using the training data:

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij} \tag{16}$$

$$\sigma_j = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_{ij} - \mu_j)^2} \tag{17}$$

where x_{ij} is the j-th feature of the i-th example, and N is the number of training examples.

The normalized features are then obtained by subtracting the mean and dividing by the standard deviation:

$$x'_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j} \tag{18}$$

Z-score normalization ensures that each feature has zero mean and unit variance, which can help improve the convergence speed and stability of gradient descent optimization.

4.2 Polynomial Feature Augmentation

Polynomial feature augmentation is a technique used to introduce non-linearity into the linear regression model by creating new features based on polynomial combinations of the original features. This allows the model to capture more complex relationships between the input features and the target variable.

Given a set of input features $\boldsymbol{x} = [x_1, x_2, \dots, x_D]$, polynomial feature augmentation creates new features by taking polynomial combinations of the original features up to a specified degree d. For example, with $D = 2$ and $d = 2$, the augmented feature vector would be:

$$\boldsymbol{x}' = [1, x_1, x_2, x_1^2, x_1 x_2, x_2^2] \tag{19}$$

The augmented feature vector includes the original features, their squared terms, and their interaction terms. The bias term is represented by the constant feature 1.

Polynomial feature augmentation allows the linear regression model to capture non-linear relationships between the input features and the target variable. However, it also increases the dimensionality of the feature space, which can lead to increased computational complexity and the risk of overfitting. Regularization techniques, such as L1 or L2 regularization, can be used to mitigate overfitting when using polynomial features.

It is important to note that polynomial feature augmentation can be computationally expensive, especially for high-dimensional input spaces or high degrees of polynomials. In practice, the degree of the polynomial should be chosen carefully based on the complexity of the problem and the available computational resources.

5 Regularization

Regularization is a technique used to prevent overfitting in linear regression models by adding a penalty term to the objective function. Overfitting occurs when the model learns to fit the noise in the training data, leading to poor generalization performance on unseen data. Regularization helps to constrain the model's complexity and encourages simpler and more generalizable solutions.

5.1 L2 Regularization (Ridge Regression)

L2 regularization, also known as Ridge regression, adds a penalty term to the MSE objective function based on the L2 norm of the weight vector:

$$\text{MSE}_{\text{L2}}(\boldsymbol{w}, b) = \frac{1}{N} \sum_{i=1}^{N} (\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)^2 + \lambda \|\boldsymbol{w}\|_2^2 \tag{20}$$

where $\lambda > 0$ is the regularization parameter that controls the strength of the regularization, and $\|\boldsymbol{w}\|_2^2 = \sum_{j=1}^{D} w_j^2$ is the L2 norm of the weight vector.

The L2 regularization term encourages the model to have small weight values, effectively shrinking the weights towards zero. This helps to reduce the impact of less important features and prevents the model from overfitting to noise in the training data.

The closed-form solution for Ridge regression is given by:

$$\boldsymbol{w}^* = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \tag{21}$$

where $\boldsymbol{I}$ is the identity matrix.

When using gradient descent optimization, the update rule for the weight vector becomes:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \alpha \left(\nabla_{\boldsymbol{w}} \text{MSE}(\boldsymbol{w}^{(t)}, b^{(t)}) + 2\lambda \boldsymbol{w}^{(t)} \right) \tag{22}$$

5.2 L1 Regularization (Lasso Regression)

L1 regularization, also known as Lasso (Least Absolute Shrinkage and Selection Operator) regression, adds a penalty term to the MSE objective function based on the L1 norm of the weight vector:

$$\mathrm{MSE}_{\mathrm{L1}}(\boldsymbol{w}, b) = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}_i + b - y_i)^2 + \lambda\|\boldsymbol{w}\|_1 \tag{23}$$

where $\|\boldsymbol{w}\|_1 = \sum_{j=1}^{D} |w_j|$ is the L1 norm of the weight vector.

L1 regularization has the property of promoting sparsity in the weight vector, meaning that it encourages some of the weights to be exactly zero. This feature selection property is useful when dealing with high-dimensional input spaces, as it automatically identifies and discards irrelevant or redundant features.

Unlike Ridge regression, Lasso regression does not have a closed-form solution due to the non-differentiability of the L1 norm at zero. However, it can be solved using iterative optimization techniques such as coordinate descent or proximal gradient methods.

The choice between L1 and L2 regularization depends on the specific problem and the desired properties of the model. L1 regularization is preferred when feature selection is important, while L2 regularization is often used when all features are considered relevant and the goal is to prevent overfitting.

6 Model Evaluation and Selection

Evaluating the performance of a linear regression model and selecting the best model configuration are crucial steps in the machine learning pipeline. Here, we discuss some common techniques for model evaluation and selection.

6.1 Train-Test Split

The train-test split is a basic technique for evaluating the performance of a machine learning model. It involves splitting the available data into two subsets: a training set used to train the model and a test set used to evaluate its performance on unseen data.

A common split ratio is 80% for the training set and 20% for the test set, although this can vary depending on the size of the dataset and the specific problem. The model is trained on the training set, and its performance is evaluated on the test set using metrics such as mean squared error (MSE), root mean squared error (RMSE), or mean absolute error (MAE).

It is important to ensure that the test set is representative of the overall data distribution and is not used during the model training process to avoid overfitting and obtain an unbiased estimate of the model's generalization performance.

6.2 Cross-Validation

Cross-validation is a more robust technique for evaluating the performance of a machine learning model and selecting the best model configuration. It involves splitting the data into multiple subsets, training and evaluating the model on different combinations of these subsets, and averaging the results to obtain a more reliable estimate of the model's performance.

The most common cross-validation technique is k-fold cross-validation, where the data is divided into k equally sized folds. The model is trained and evaluated k times, each time using a different fold as the test set and the remaining $(k-1)$ folds as the training set. The performance metrics are then averaged across all k iterations to obtain the final estimate.

Cross-validation helps to reduce the impact of random variations in the data split and provides a more accurate assessment of the model's performance. It is particularly useful when the amount of available data is limited, as it allows for more efficient use of the data for both training and evaluation.

6.3 Hyperparameter Tuning

Hyperparameter tuning is the process of selecting the best combination of hyperparameters for a machine learning model. Hyperparameters are parameters that are not learned from the data but are set before the training process begins, such as the learning rate, regularization parameter, or the degree of polynomial features.

The goal of hyperparameter tuning is to find the hyperparameter configuration that maximizes the model's performance on unseen data. This is typically done using a combination of cross-validation and grid search or random search.

In grid search, a predefined set of hyperparameter values is specified, and the model is trained and evaluated for all possible combinations of these values using cross-validation. The hyperparameter configuration that yields the best average performance across the cross-validation folds is selected as the optimal configuration.

Random search is an alternative to grid search, where hyperparameter values are sampled from specified distributions rather than exhaustively searching through a predefined set. This can be more efficient than grid search, especially when the number of hyperparameters is large, and the optimal values are likely to be within certain ranges.

Hyperparameter tuning is an essential step in building high-performance machine learning models, as the choice of hyperparameters can significantly impact the model's performance and generalization ability.

7 Challenges and Limitations

While linear regression is a powerful and widely used technique, it is important to be aware of its challenges and limitations:

1. **Linearity Assumption**: Linear regression assumes a linear relationship between the input features and the target variable. If the true relationship is non-linear, linear regression may not capture the underlying patterns accurately, leading to suboptimal performance. In such cases, feature transformations or more advanced non-linear models may be required.

2. **Outliers**: Linear regression is sensitive to outliers, which are data points that significantly deviate from the general trend. Outliers can have a disproportionate impact on the estimated model parameters, leading to biased or unstable results. Robust regression techniques, such as Huber regression or RANSAC (Random Sample Consensus), can be used to mitigate the effect of outliers.

3. **Multicollinearity**: Multicollinearity refers to high correlations among the input features. When features are highly correlated, the estimated model parameters can become unstable and difficult to interpret. Multicollinearity can also lead to increased variance in the parameter estimates and reduced model performance. Techniques like principal component analysis (PCA) or regularization can be used to address multicollinearity.

4. **High-Dimensional Data**: Linear regression can struggle with high-dimensional data, where the number of input features is much larger than the number of observations. In such cases, the model may overfit the training data and generalize poorly to unseen data. Regularization techniques, such as L1 (Lasso) or L2 (Ridge) regularization, can help mitigate overfitting by constraining the model complexity.

5. **Heteroscedasticity**: Linear regression assumes that the noise variance is constant across all observations (homoscedasticity). However, in some cases, the noise variance may vary with the input features or the

predicted values, leading to heteroscedasticity. Heteroscedasticity can affect the efficiency and validity of the model's inferences. Techniques like weighted least squares or generalized least squares can be used to account for heteroscedasticity.

6. **Autocorrelation**: Linear regression assumes that the observations are independent of each other. However, in time series or spatial data, observations may be correlated with their past or neighboring values, violating the independence assumption. Autocorrelation can lead to biased parameter estimates and incorrect inferences. Techniques like autoregressive models or generalized least squares with appropriate covariance structures can be used to handle autocorrelation.

7. **Causality and Interpretability**: Linear regression provides a measure of association between the input features and the target variable, but it does not necessarily imply causality. The estimated model parameters should be interpreted with caution, as they may be influenced by confounding factors or spurious correlations. Domain knowledge and careful experimental design are essential for making causal claims based on linear regression results.

8. **Model Selection and Validation**: Choosing the appropriate set of input features and validating the model's performance are crucial steps in linear regression. Model selection techniques, such as stepwise regression or regularization, can help identify the most relevant features. Cross-validation and holdout validation should be used to assess the model's generalization performance and avoid overfitting.

Addressing these challenges and limitations requires careful data preprocessing, feature engineering, model selection, and validation. It is important to critically evaluate the assumptions of linear regression and consider alternative models or techniques when the assumptions are violated or the data exhibits complex patterns.

Despite its limitations, linear regression remains a valuable tool in the data scientist's toolkit due to its simplicity, interpretability, and effectiveness in many real-world applications. Understanding its strengths and weaknesses allows for informed decision-making and the appropriate application of linear regression in the context of the given problem and dataset.

8 Discussions

Linear regression is a fundamental technique in machine learning for predicting continuous target variables based on input features. In this document, we covered the linear regression model, its closed-form solution, and optimization techniques such as batch gradient descent and mini-batch stochastic gradient descent. We discussed the impact of the learning rate on convergence speed and stability, as well as preprocessing techniques like z-score normalization and polynomial feature augmentation.

We also explored regularization techniques, such as L1 and L2 regularization, which help prevent overfitting and promote simpler and more generalizable models. Finally, we discussed model evaluation and selection techniques, including train-test split, cross-validation, and hyperparameter tuning, which are essential for assessing the model's performance and selecting the best configuration.

Understanding these concepts and techniques is crucial for effectively applying linear regression to real-world problems and building robust and accurate predictive models. By mastering linear regression, you lay a solid foundation for exploring more advanced machine learning algorithms and techniques in the future.

A Derivation of the Closed-Form Solution

The closed-form solution for linear regression can be derived by setting the gradient of the MSE with respect to the weight vector $\boldsymbol{w}$ to zero and solving the resulting system of linear equations.

Recall that the MSE objective function is given by:

$$\mathrm{MSE}(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{w}^\top \boldsymbol{x}_i - y_i)^2 \tag{24}$$

where $\boldsymbol{x}_i \in \mathbb{R}^{D+1}$ is the augmented input feature vector (including the bias term) and $y_i \in \mathbb{R}$ is the corresponding target value.

The gradient of the MSE with respect to $\boldsymbol{w}$ is given by:

$$\begin{aligned} \nabla_{\boldsymbol{w}}\mathrm{MSE}(\boldsymbol{w}) &= \frac{2}{N}\sum_{i=1}^{N}(\boldsymbol{w}^\top \boldsymbol{x}_i - y_i)\boldsymbol{x}_i & (25)\\ &= \frac{2}{N}\left(\sum_{i=1}^{N}\boldsymbol{x}_i\boldsymbol{x}_i^\top \boldsymbol{w} - \sum_{i=1}^{N} y_i\boldsymbol{x}_i\right) & (26)\\ &= \frac{2}{N}(\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^\top\boldsymbol{y}) & (27)\end{aligned}$$

where $\boldsymbol{X} \in \mathbb{R}^{N\times(D+1)}$ is the matrix of augmented input feature vectors, and $\boldsymbol{y} \in \mathbb{R}^N$ is the vector of target values.

Setting the gradient to zero:

$$\begin{aligned} \nabla_{\boldsymbol{w}}\mathrm{MSE}(\boldsymbol{w}) = \frac{2}{N}(\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^\top\boldsymbol{y}) = 0 & \quad (28)\\ \boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{w} = \boldsymbol{X}^\top\boldsymbol{y} & \quad (29)\end{aligned}$$

Assuming that $\boldsymbol{X}^\top\boldsymbol{X}$ is invertible (which is true if the input features are linearly independent), we can multiply both sides by $(\boldsymbol{X}^\top\boldsymbol{X})^{-1}$:

$$\begin{aligned} (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{w} &= (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} & (30)\\ \boldsymbol{w}^* &= (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} & (31)\end{aligned}$$

This is the closed-form solution for the optimal weight vector $\boldsymbol{w}^*$ that minimizes the MSE objective function. It provides a direct way to compute the optimal weights given the input features and target values, without the need for iterative optimization.

However, it is important to note that the closed-form solution requires the computation of the inverse of $\boldsymbol{X}^\top\boldsymbol{X}$, which can be computationally expensive for large datasets or high-dimensional feature spaces. In such cases, iterative optimization techniques like gradient descent are often preferred.

B Gradient Descent Update Rules

The gradient descent update rules for linear regression can be derived by taking the gradient of the MSE objective function with respect to the weight vector $\boldsymbol{w}$ and the bias term b.

Recall that the MSE objective function with the bias term is given by:

$$\mathrm{MSE}(\boldsymbol{w}, b) = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)^2 \tag{32}$$

where $\boldsymbol{x}_i \in \mathbb{R}^D$ is the input feature vector, $y_i \in \mathbb{R}$ is the corresponding target value, $\boldsymbol{w} \in \mathbb{R}^D$ is the weight vector, and $b \in \mathbb{R}$ is the bias term.

The gradients of the MSE with respect to $\boldsymbol{w}$ and b are given by:

$$\nabla_{\boldsymbol{w}} \text{MSE}(\boldsymbol{w}, b) = \frac{2}{N} \sum_{i=1}^{N} (\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_i + b - y_i) \boldsymbol{x}_i \tag{33}$$

$$\frac{\partial}{\partial b} \text{MSE}(\boldsymbol{w}, b) = \frac{2}{N} \sum_{i=1}^{N} (\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_i + b - y_i) \tag{34}$$

In batch gradient descent, the update rules for $\boldsymbol{w}$ and b are given by:

$$\begin{aligned} \boldsymbol{w}^{(t+1)} &= \boldsymbol{w}^{(t)} - \alpha \nabla_{\boldsymbol{w}} \text{MSE}(\boldsymbol{w}^{(t)}, b^{(t)}) && (35) \\ &= \boldsymbol{w}^{(t)} - \frac{2\alpha}{N} \sum_{i=1}^{N} (\boldsymbol{w}^{(t)T} \boldsymbol{x}_i + b^{(t)} - y_i) \boldsymbol{x}_i && (36) \\ b^{(t+1)} &= b^{(t)} - \alpha \frac{\partial}{\partial b} \text{MSE}(\boldsymbol{w}^{(t)}, b^{(t)}) && (37) \\ &= b^{(t)} - \frac{2\alpha}{N} \sum_{i=1}^{N} (\boldsymbol{w}^{(t)T} \boldsymbol{x}_i + b^{(t)} - y_i) && (38) \end{aligned}$$

where α is the learning rate and t denotes the iteration number.

In mini-batch stochastic gradient descent, the update rules are similar, but the gradients are computed using a randomly sampled mini-batch $\mathcal{B}_t$ of size m at each iteration:

$$\begin{aligned} \boldsymbol{w}^{(t+1)} &= \boldsymbol{w}^{(t)} - \frac{2\alpha}{m} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{B}_t} (\boldsymbol{w}^{(t)T} \boldsymbol{x}_i + b^{(t)} - y_i) \boldsymbol{x}_i && (39) \\ b^{(t+1)} &= b^{(t)} - \frac{2\alpha}{m} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{B}_t} (\boldsymbol{w}^{(t)T} \boldsymbol{x}_i + b^{(t)} - y_i) && (40) \end{aligned}$$

These update rules are derived by taking a step in the negative direction of the gradients, scaled by the learning rate α. The learning rate determines the size of the steps taken in each iteration and plays a crucial role in the convergence speed and stability of the optimization process.

The gradients used in the update rules provide the direction and magnitude of the steepest ascent of the MSE objective function at the current parameter values. By iteratively updating the parameters in the opposite direction of the gradients, gradient descent aims to find the values of $\boldsymbol{w}$ and b that minimize the MSE and yield the best-fitting linear regression model.

C Probabilistic Interpretation of Linear Regression

Linear regression can also be interpreted from a probabilistic perspective, which provides insights into the assumptions and uncertainties associated with the model.

In the probabilistic interpretation, we assume that the target variable y is generated from a linear combination of the input features $\boldsymbol{x}$ with additive Gaussian noise:

$$y = \boldsymbol{w}^\mathsf{T} \boldsymbol{x} + b + \epsilon \tag{41}$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is the Gaussian noise term with zero mean and constant variance σ^2.

Under this assumption, the likelihood of observing a target value y_i given the input features $\boldsymbol{x}_i$ and the model parameters $\boldsymbol{w}$ and b is given by:

$$p(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}, b, \sigma^2) = \mathcal{N}(y_i \mid \boldsymbol{w}^\mathsf{T} \boldsymbol{x}_i + b, \sigma^2) \tag{42}$$

Assuming that the observations are independent and identically distributed (i.i.d.), the likelihood of the entire dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ is the product of the individual likelihoods:

$$p(\mathcal{D} \mid \boldsymbol{w}, b, \sigma^2) = \prod_{i=1}^{N} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}, b, \sigma^2) \tag{43}$$

The goal in the probabilistic interpretation is to find the model parameters $\boldsymbol{w}$, b, and σ^2 that maximize the likelihood of the observed data, which is equivalent to minimizing the negative log-likelihood:

$$\begin{align}
\underset{\boldsymbol{w},b,\sigma^2}{\operatorname{argmax}}\, p(\mathcal{D} \mid \boldsymbol{w}, b, \sigma^2) &\equiv \underset{\boldsymbol{w},b,\sigma^2}{\operatorname{argmin}} -\log p(\mathcal{D} \mid \boldsymbol{w}, b, \sigma^2) \tag{44}\\
&= \underset{\boldsymbol{w},b,\sigma^2}{\operatorname{argmin}} \sum_{i=1}^{N} \left(\frac{(y_i - \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_i - b)^2}{2\sigma^2} + \frac{1}{2}\log(2\pi\sigma^2) \right) \tag{45}
\end{align}$$

Minimizing the negative log-likelihood with respect to $\boldsymbol{w}$ and b while keeping σ^2 fixed is equivalent to minimizing the MSE objective function. The optimal value of σ^2 can be estimated using the residuals after finding the optimal $\boldsymbol{w}$ and b:

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_i - b)^2 \tag{46}$$

The probabilistic interpretation allows for the computation of predictive distributions for new input features $\boldsymbol{x}_*$. The predictive distribution for a new target value y_* is given by:

$$p(y_* \mid \boldsymbol{x}_*, \mathcal{D}) = \mathcal{N}(y_* \mid \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_* + b, \sigma^2) \tag{47}$$

where $\boldsymbol{w}$ and b are the optimal model parameters learned from the training data, and σ^2 is the estimated noise variance.

The predictive distribution provides not only a point estimate for the predicted target value but also a measure of uncertainty through the variance σ^2. This allows for the computation of confidence intervals and the assessment of the model's predictive uncertainty.

The probabilistic interpretation of linear regression highlights the assumptions of the model, such as the linearity of the relationship between the input features and the target variable, the independence and identical distribution of the observations, and the Gaussian noise assumption. It also provides a framework for incorporating prior knowledge about the model parameters through Bayesian linear regression, which can further enhance the model's performance and interpretability.