COMP4211
Machine Learning

Notes

Insturctor: Dit Yan, Yeung

Fall 2025

Edited by

ThunderDora

The Hong Kong University of Science and Technology

Last Edited: ***August 18, 2025***

Contents

1 Machine Learning

1.1 What is Machine Learning?

★ Definition

Machine Learning is the science of **making computer artifacts improve their performance** with respect to a certain performance criterion using example data without requring humans to explicitly program the rules for improvement.

Machine Learning problems can be categorized into three main types:

- **Supervised Learning**: Algorithm that learns from a labeled dataset, where the input data corresponds to the accurate output.
- **Unsupervised Learning**: Model that is developed using an unlabeled dataset, in which the input data lacks associated output labels.
- **Reinforcement Learning**: Algorithm that acquires knowledge through engagement with an environment, obtaining feedback as rewards or penalties depending on its actions.

1.2 What is Supervised Learning?

The fundamental steps of supervised learning is outlined below:

1. Consider a training set $S = \{(\mathbf{x}^{(l)}, \mathbf{y}^{(l)})\}_{l=1}^{N}$ consisting of N labeled pairs of inputs and outputs.
2. Identify a function $f(\cdot)$ with the training set S so that $f(\mathbf{x}^{(l)}) \approx \mathbf{y}^{(l)}$ holds for all $l = 1, \ldots, N$, and that $f(\mathbf{x})$ for new examples $\mathbf{x}$ also results in $f(\mathbf{x}) \approx \mathbf{y}$ from the same distribution.
3. In the testing stage, examples without labels containing only the input $\mathbf{x}$ are given, and the model determines the output $\mathbf{y}$ by applying the learned function $f(\mathbf{x})$.

Supervised learning is generally applied to address two kinds of issues:

- **Classification**: The output $\mathbf{y}$ is a **categorical** value, like a label class.
- **Regression**: The output $\mathbf{y}$ is a **continuous** quantity, like a real number.

1.3 What is Unsupervised Learning?

The basic steps of unsupervised learning are detailed below:

1. Take a training set $S = \{\mathbf{x}^{(l)}\}_{l=1}^{N}$ that contains N inputs without labels.
2. Determine a function $f(\cdot)$ using the training set S such that $f(\mathbf{x}^{(l)})$ represents the essential pattern of the data

Unsupervised learning is typically utilized to tackle four types of problems:

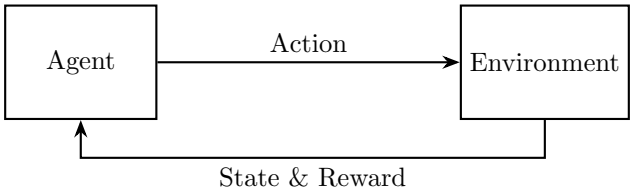
- **Clustering**: Method that organizes the data into clusters by their similarities.
- **Dimensionality Reduction**: Algorithm that minimizes the number of attributes while maintaining crucial information.
- **Anomaly Detection**: Algorithm that detects atypical patterns that deviate from anticipated behavior.
- **Density Estimation**: Method that calculates the likelihood distribution of the data.

1.4 What is Reinforcement Learning?

Reinforcement learning is a form of machine learning in which an agent learns to make choices by performing actions in an environment to optimize total rewards. The essential elements of reinforcement learning are:

- **Agent**: Entity making decisions or learning by engaging with the environment.
- **Environment**: Outside system that the agent engages with.
- **State**: Depiction of the existing circumstances of the environment.
- **Action**: Decision made by the agent that influences the condition of the environment.
- **Reward**: Feedback signal obtained by the agent following an action, showing how effective or ineffective that action was.

The reinforcement learning process entails the agent noticing the existing state of the environment, choosing an action guided by a policy, obtaining a reward, and refining its policy to enhance future actions.



2 Linear Regression

2.1 What is Regression?

★ Definition

Regression is a statistical modeling approach used to estimate the **relationship between a dependent variable and one or several independent variables that may contain errors**. The aim of regression is to identify the most suitable line or curve that represents the connection between the variables.

The fundamental concept of regression can be illustrated through the following steps:

1. Consider a training set $S = \{(\mathbf{x}^{(l)}, \mathbf{y}^{(l)})\}_{l=1}^{N}$ which comprises N pairs of inputs and corresponding outputs, with $\mathbf{x}^{(l)}$ representing the input and $\mathbf{y}^{(l)}$ denoting the output.
2. The input $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ represents a vector in d dimensions, with d denoting the count of features or attributes.
3. **Regression Function** $f(\cdot; \mathbf{w})$ employs S to ensure the predicted outcome $f(\mathbf{x}^{(l)}; \mathbf{w})$ is as near as feasible to the true output $\mathbf{y}^{(l)}$ for every $l = 1, \ldots, N$, with $\mathbf{w}$ representing the parameter vector of the regression function.
4. When the output $\mathbf{y}$ consists of a multivariate vector, it represents a **multi-output regression** issue. For a univariate result, it constitutes a **single-output regression** issue.

2.2 Linear Regression Function

If the regression function $f(\mathbf{x}; \mathbf{w})$ is **linear**, it can be expressed as:

$$f(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_d x_d = \begin{bmatrix} w_0 & w_1 & w_2 & \ldots & w_d \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} = \mathbf{w}^T \tilde{\mathbf{x}} = \tilde{\mathbf{x}}^T \mathbf{w} \tag{2.1}$$

- Weight w_0 represents the **bias** term, which is a fixed value that acts as an adjustment for the regression function.
- Learning problem involves identifying the optimal parameter vector $\mathbf{w} = (w_0, w_1, \ldots, w_d)$ that reduces the discrepancy between the predicted output $f(\mathbf{x}^{(l)}; \mathbf{w})$ and the actual output $\mathbf{y}^{(l)}$ for every $l = 1, \ldots, N$.

2.3 Squared Loss Function

To determine the parameter vector $\mathbf{w}$ of the linear regression function $f(\mathbf{x}; \mathbf{w})$, it is essential to establish a loss function $L(\mathbf{w}; S)$ that measures the discrepancy between the predicted output and the true output. The loss function most frequently utilized for linear regression is the **squared loss function**, defined as:

$$L(\mathbf{w}; S) = \sum_{l=1}^{N} \left(f(\mathbf{x}^{(l)}; \mathbf{w}) - \mathbf{y}^{(l)} \right)^2 = \sum_{l=1}^{N} \left(w_0 + w_1 x_1^{(l)} + w_2 x_2^{(l)} + \ldots + w_d x_d^{(l)} - \mathbf{y}^{(l)} \right)^2 \tag{2.2}$$

The **mean squared error** (MSE) can be defined as the average of the squared loss function across all N training samples:

$$\text{MSE}(\mathbf{w}; S) = \frac{1}{N} L(\mathbf{w}; S) = \frac{1}{N} \sum_{l=1}^{N} \left(f(\mathbf{x}^{(l)}; \mathbf{w}) - \mathbf{y}^{(l)} \right)^2 \tag{2.3}$$

The squared loss function has two scenarios: $d = 1$ and $d > 1$, with d representing the count of features in the input $\mathbf{x}$.

2.3.1 Special Case: Single-Output Regression ($d = 1$)

Considering the squared loss function for single-output regression, the loss function can be represented as:

$$\boxed{L(w_0, w_1; S) = \sum_{l=1}^{N} \left(w_0 + w_1 x^{(l)} - y^{(l)}\right)^2} \tag{2.4}$$

The distinctive optimal solution $\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$ can be obtained by minimizing the loss function $L(w_0, w_1; S)$ through the least squares technique.

To find the optimal solution, one can calculate the partial derivatives of the loss function concerning w_0 and w_1, equate them to zero, and resolve the resulting equations:

$$\frac{\partial L}{\partial w_0} = 2\sum_{l=1}^{N} \left(w_0 + w_1 x^{(l)} - y^{(l)}\right) \cdot 1 = 0 \implies \sum_{l=1}^{N} \left(w_0 + w_1 x^{(l)}\right) = \sum_{l=1}^{N} y^{(l)} \implies N w_0 + w_1 \sum_{l=1}^{N} x^{(l)} = \sum_{l=1}^{N} y^{(l)}$$

$$\frac{\partial L}{\partial w_1} = 2\sum_{l=1}^{N} \left(w_0 + w_1 x^{(l)} - y^{(l)}\right) x^{(l)} = 0 \implies w_0 \sum_{l=1}^{N} x^{(l)} + w_1 \sum_{l=1}^{N} x^{(l)^2} = \sum_{l=1}^{N} y^{(l)} x^{(l)}$$

We have a system of linear equations consisting of two equations and two unknown variables, which can be represented in matrix notation as:

$$\boxed{\mathbf{A}\mathbf{w} = \begin{bmatrix} N & \sum_{l=1}^{N} x^{(l)} \\ \sum_{l=1}^{N} x^{(l)} & \sum_{l=1}^{N} x^{(l)^2} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} \sum_{l=1}^{N} y^{(l)} \\ \sum_{l=1}^{N} y^{(l)} x^{(l)} \end{bmatrix} = \mathbf{b}} \tag{2.5}$$

Assuming the matrix $\mathbf{A}$ is invertible, the optimal solution can be found by multiplying both sides of the equation by $\mathbf{A}^{-1}$:

$$\boxed{\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}} \tag{2.6}$$

2.3.2 General Case: Multi-Output Regression ($d > 1$)

In multi-output regression, two methods can be used to identify the best solution.

The initial strategy is to utilize the **least squares method**. Initially, we represent the input and output sections of the N examples as:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_d^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)} & x_2^{(N)} & \dots & x_d^{(N)} \end{bmatrix} \in \mathbb{R}^{N \times (d+1)} \quad \text{and} \quad \mathbf{Y} = \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \\ \vdots \\ y_d^{(N)} \end{bmatrix} \in \mathbb{R}^{N \times d}$$

We obtain a system of $d + 1$ linear equations involving $d + 1$ unknowns, which can be represented in matrix form as:

$$\boxed{\mathbf{A}\mathbf{w} = (\mathbf{X}^T\mathbf{X})\mathbf{w} = \mathbf{X}^T\mathbf{y} = \mathbf{b}} \tag{2.7}$$

Assuming the matrix $\mathbf{A} = \mathbf{X}^T\mathbf{X}$ is invertible, one can determine the optimal solution by multiplying both sides of the equation by $\mathbf{A}^{-1}$:

$$\boxed{\mathbf{w} = \mathbf{A}^{-1}\mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}} \tag{2.8}$$

An alternative method is to differentiate using **Multivariable Calculus**. Initially, we can represent $\mathbf{Xw} - \mathbf{Y}$ as:

$$\mathbf{Xw} - \mathbf{Y} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_d^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)} & x_2^{(N)} & \dots & x_d^{(N)} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix} = \begin{bmatrix} w_0 + w_1 x_1^{(1)} + w_2 x_2^{(1)} + \ldots + w_d x_d^{(1)} - y^{(1)} \\ w_0 + w_1 x_1^{(2)} + w_2 x_2^{(2)} + \ldots + w_d x_d^{(2)} - y^{(2)} \\ \vdots \\ w_0 + w_1 x_1^{(N)} + w_2 x_2^{(N)} + \ldots + w_d x_d^{(N)} - y^{(N)} \end{bmatrix}$$

The squared loss function can be represented as the **squared L-2 norm of the vector** $\mathbf{Xw} - \mathbf{y}$:

$$\boxed{L(\mathbf{w}; S) = \|\mathbf{Xw} - \mathbf{y}\|^2} \tag{2.9}$$

Then we can express the squared loss function as:

$$L(\mathbf{w}; S) = (\mathbf{Xw} - \mathbf{y})^T (\mathbf{Xw} - \mathbf{y}) = \mathbf{w}^T \mathbf{X}^T \mathbf{Xw} - 2\mathbf{y}^T \mathbf{Xw} + \mathbf{y}^T \mathbf{y}$$

To reduce the squared loss function, we can calculate the gradient of $L(\mathbf{w}; S)$ concerning $\mathbf{w}$ and equate it to zero:

$$\begin{aligned} \nabla_{\mathbf{w}} L(\mathbf{w}; S) = 2\mathbf{X}^T \mathbf{Xw} - 2\mathbf{X}^T \mathbf{y} = 0 &\implies \mathbf{X}^T \mathbf{Xw} = \mathbf{X}^T \mathbf{y} \\ &\implies \mathbf{w} = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{y} \end{aligned}$$

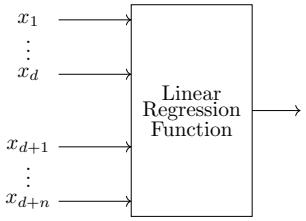
To find the optimal solution $\mathbf{w}$, we must compute the inverse of the matrix $\mathbf{X}^T\mathbf{X} \in \mathbb{R}^{(d+1)\times(d+1)}$. We can apply the **LeGall algorithm**, recognized as the quickest identified method for matrix multiplication, exhibiting a **time complexity of** $O(n^{2.3728596})$ for an $n \times n$ matrix, in contrast to the straightforward $O(n^3)$ approach used in Cholesky decomposition, LU decomposition, or Gaussian elimination.

2.4 Nonlinear Extension of Linear Regression

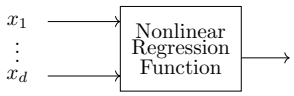
Nonlinear regression functions are crucial because more complex issues cannot be addressed by linear regression. To tackle this, we can modify the linear regression function $f(\mathbf{x}; \mathbf{w})$ into a nonlinear format by adding nonlinear transformations of the input features. Three distinct methods exist for nonlinear expansions to accomplish this:

1. **Feature Engineering**: Intentionally generate additional input dimensions (nonlinearly related to the original input) and utilize linear regression on the newly created input dimensions. For instance, if the original input is $\mathbf{x} = (x_1, x_2)$, we can generate additional features like x_1^2, x_2^2, and $x_1 x_2$ to create an updated input $\tilde{\mathbf{x}} = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$.

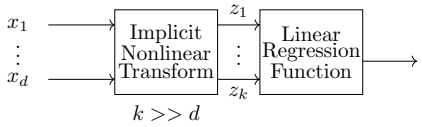
   The key benefit is that linear regression remains applicable, and the weights in the model are **highly interpretable** (the greater the weight's magnitude, the more significant the feature). The primary drawback is that it necessitates domain expertise to develop new features and can be costly in terms of computation if the feature count is high.



2. **Explicit Mapping**: Apply an explicitly defined **nonlinear regression function** $f(\mathbf{x}; \mathbf{w})$ that is nonlinear in the input $\mathbf{x}$, such as polynomial regression, radial basis function (RBF) regression, or neural networks. The model learns the parameters $\mathbf{w}$ of the nonlinear function directly from the training data.



3. **Implicit Mapping**: Implement an **implicitly defined nonlinear transformation** on the original input, followed by utilizing a linear regression function on the modified input. This method employs kernel techniques or neural networks with latent layers to automatically acquire valuable feature representations without their explicit specification. For instance, kernel techniques such as Support Vector Machines (SVM) implicitly transform the input into a higher-dimensional space via kernel functions.



2.5 Polynomial Regression

Polynomial Regression introduces higher-order polynomial terms as the additional input features in the linear regression function. The polynomial regression function can be expressed as:

$$f(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x + \ldots + w_m x^m = \begin{bmatrix} w_0 & w_1 & \ldots & w_m \end{bmatrix} \begin{bmatrix} 1 \\ x \\ \vdots \\ x^m \end{bmatrix} = \mathbf{w}^T \tilde{\mathbf{x}} = \tilde{\mathbf{x}}^T \mathbf{w} \tag{2.10}$$

- The parameter vector $\mathbf{w} = (w_0, w_1, \ldots, w_m)$ consists of $m + 1$ parameters, where m is the degree of the polynomial.
- The input $\tilde{\mathbf{x}} = (1, x, x^2, \ldots, x^m)$ represents the polynomial features derived from the original input x.

Although $f(\mathbf{x}; \mathbf{w})$ is nonlinear with respect to the input x, it stays linear concerning the parameters $\mathbf{w}$. Consequently, we can utilize the same least squares technique to determine the ideal parameter vector $\mathbf{w}$ by minimizing the squared loss function.

Additionally, we can employ **broader transformations** of the initial input dimensions to develop polynomial features. For example, feature engineering can specify the features tailored to the application that are used on the input data, including interaction terms or transformations specific to the domain. Using body weight and body height as an illustration, the body mass index (BMI) will be the supplementary characteristic specified. The least squares method for linear regression remains applicable to determine the optimal parameter vector $\mathbf{w}$ for acquiring the **closed form solution**.

2.6 Regularization

Regularization is a technique that alters the original loss function by incorporating one or more penalty terms known as **regularizers** that discourage large parameter values to avoid overfitting due to excessively high parameter magnitudes.

Overfitting arises when the **training dataset $S = \{(\mathbf{x}^{(l)}, \mathbf{y}^{(l)})\}_{l=1}^{N}$ is limited in size relative to the number of parameters in the linear regression model $f(\mathbf{x}; \mathbf{w})$**. In this scenario, the model could discover significant values in certain parameters, as an expanded search space enables the model to align more closely with the training data, which may result in inadequate generalization on new data.

Regularization helps tackle this issue by constraining the magnitude of the parameters, promoting simpler models that are less likely to overfit the training data. The **loss function with regularization** employing **L_2 regularization / Tikhonov regularization** can be represented as:

$$\boxed{L_\lambda(\mathbf{w}; S) = L(\mathbf{w}; S) + \lambda\|\mathbf{w}\|^2 = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \lambda\|\mathbf{w}\|^2 = \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{y}^T\mathbf{y} + \lambda\mathbf{w}^T\mathbf{w}} \tag{2.11}$$

- The parameter $\lambda > 0$ serves as a **regularization hyperparameter** that regulates the intensity of the regularization, established during the validation phase.
  A greater λ produces more intense regularization, whereas a lesser λ causes weaker regularization.

- The notation $\|\mathbf{w}\|^2$ signifies the squared L2 norm of the parameter vector $\mathbf{w}$, imposing a penalty on large parameter values.

In practice, the bias w_0 is frequently left out of the regularization term since it merely acts as an **offset** and does not add to the model's complexity. The loss function with regularization can be optimized through techniques like gradient descent or by utilizing **closed-form solutions**.

To find the closed-form solution with L_2 regularization, we can differentiate the loss function $L_\lambda(\mathbf{w}; S)$ with respect to $\mathbf{w}$ and set it to zero:

$$2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y} + 2\lambda\mathbf{w} = 0 \implies (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\mathbf{w} = \mathbf{X}^T\mathbf{y} \quad \text{where } \mathbf{I} \text{ is the identity matrix}$$

The closed-form solution for the least squares estimate of the parameter vector $\mathbf{w}$ with L_2 regularization can be defined as:

$$\boxed{\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}} \quad \text{where } \mathbf{X}^T\mathbf{X} + \lambda\mathbf{I} \text{ is invertible for } \lambda > 0 \tag{2.12}$$

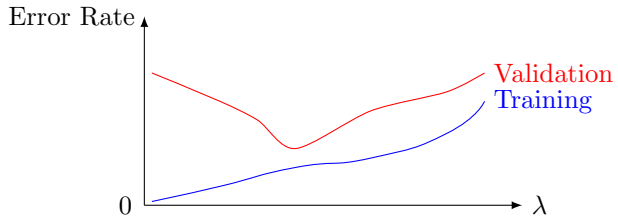
This regularized method is referred to as **Ridge Regression**. It minimizes overfitting and enhances the model's ability to generalize to new data by imposing penalties on large parameter values. Crucially, when the regularization parameter λ is equal to zero, ridge Regression reduces to standard linear regression, since no penalty is enforced.

2.6.1 Choice of Regularization Parameter λ

The regularization coefficient λ is an essential hyperparameter that dictates the intensity of the regularization imposed on the model. The selection of λ can greatly influence the performance of the model and is usually established via a validation process. A frequent method for determining the best value of λ is to employ **cross-validation**.

Cross-validation involves **splitting the training dataset into multiple subsets (folds) and training the model on one subset** while evaluating it on the remaining data. Subsequently, the trained model is **evaluated on a separate validation set** to resemble the test set. The procedure is carried out for different λ values, and the model's effectiveness is assessed utilizing a specific metric (e.g., mean squared error, accuracy). The best performance on the validation set is achieved by choosing the optimal regularization parameter as the value of λ with the smallest validation error.

The standard training and validation error curves for various λ values are illustrated below.



The training error usually rises with an increase in λ, whereas the validation error first drops and then rises, suggesting that **a low λ causes overfitting whereas a high λ leads to underfitting**.

2.6.2 Other Regularization Techniques

In addition to L_2 regularization, other approaches can also be used to prevent overfitting in machine learning models. A general framework for these methods is provided by the L_p norm, which generalizes both the L_1 and L_2 norms. By varying the value of p, a family of regularization strategies can be defined:

$$\|\mathbf{v}\|_p = \left(\sum_{i=1}^{n} |v_i|^p\right)^{1/p} \quad \text{for } p \geq 1 \tag{2.13}$$

Among these, **L_1 regularization**, known as **LASSO (Least Absolute Shrinkage and Selection Operator)**, is especially popular. LASSO adds a penalty term based on the L_1 norm of the parameter vector $\mathbf{w}$ to the loss function, which encourages sparsity by driving many coefficients to exactly zero:

$$L_\lambda(\mathbf{w}; S) = L(\mathbf{w}; S) + \lambda\|\mathbf{w}\|_1 = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \lambda \sum_{i=0}^{d} |w_i| \tag{2.14}$$

While L_1 regularization leads to a convex optimization problem, it does not admit a closed-form solution. Instead, iterative optimization algorithms such as coordinate descent or subgradient descent are typically used to find the optimal parameter vector $\mathbf{w}$.

2.7 Common Performance Metrics for Linear Regression

To evaluate the performance of linear regression models, several metrics are commonly used. These metrics help assess how well the model predicts the target variable and can guide improvements in model design. Two of the most widely used metrics are **Mean Squared Error (MSE)** and **R-squared Scores**.

2.7.1 Mean Squared Error (MSE)

Once a regression model has been trained, it is crucial to quantitatively evaluate how accurately the model's predictions align with the real target values. A commonly utilized metric for this aim is the **Mean Squared Error (MSE)**. MSE computes the **mean of the squared deviations between the predicted values and the actual target values for all samples**. It is described as:

$$\text{MSE} = \frac{1}{N}\sum_{l=1}^{N}\left(f(\mathbf{x}^{(l)}; \mathbf{w}) - \mathbf{y}^{(l)}\right)^2 = \frac{1}{N}L(\mathbf{w}; S) \tag{2.15}$$

- N is the total number of samples in the dataset, $f(\mathbf{x}^{(l)}; \mathbf{w})$ is the predicted value for the l-th sample, and $\mathbf{y}^{(l)}$ is the corresponding true value.

MSE can be calculated on the training set, validation set, or test set to assess model performance and generalization.

Nonetheless, since MSE is represented in the squared units of the target variable, interpreting it directly can be challenging. To tackle this, the **Root Mean Squared Error (RMSE)** is commonly employed, as it calculates the square root of the MSE to revert the error to the initial scale of the target variable:

$$\boxed{\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{N}\sum_{l=1}^{N}\left(f(\mathbf{x}^{(l)};\mathbf{w}) - \mathbf{y}^{(l)}\right)^2}} \tag{2.16}$$

Consequently, RMSE is easier to understand and gives a clear indication of the usual size of prediction errors in the same units as the target variable.

2.7.2 R-squared (R^2) Score

Though MSE and RMSE offer an absolute assessment of prediction error, having a relative metric that shows how effectively the model accounts for the variability in the target variable against a basic baseline is often beneficial. The **R-squared (R^2) value**, often referred to as the **coefficient of determination**, fulfills this role.

R^2 measures the fraction of the overall variability in the target variable that the model's predictions explain. It is characterized as:

$$\boxed{R^2 = 1 - \frac{\sum_{l=1}^{N}\left(f(\mathbf{x}^{(l)};\mathbf{w}) - \mathbf{y}^{(l)}\right)^2}{\sum_{l=1}^{N}\left(\mathbf{y}^{(l)} - \bar{\mathbf{y}}\right)^2} = 1 - \frac{\text{MSE}}{\text{VAR}(\mathbf{y})}} \quad \text{where } \bar{\mathbf{y}} = \frac{1}{N}\sum_{l=1}^{N}\mathbf{y}^{(l)} \text{ is the mean of the target variable} \tag{2.17}$$

- The numerator is the residual sum of squares (RSS), indicating the variance that remains unexplained (the discrepancy between predicted and actual values).
- The denominator is the total sum of squares (TSS), which indicates the overall variance in the target variable concerning its average.

R^2 varies between 0 and 1, with **1 indicating that the model completely accounts for all variability** in the target variable, and **0 signifying no explanation** at all (similar to predicting the average consistently). Negative R^2 values may arise when the model is less effective than merely predicting the average, signifying inadequate model fit.

3 Logistic Regression

3.1 What is Logistic Regression?

★ Definition

Logistic Regression is a statistical method used for **classification** tasks. Unlike linear regression, which predicts continuous values, logistic regression maps linear combinations of input features to probabilities of class membership using the logistic function (sigmoid function) for binary classification and the softmax function for multiclass classification.

Logistic regression addresses the fundamental difference between regression and classification problems. While linear regression outputs continuous values that can range from negative infinity to positive infinity, classification requires outputs that represent probabilities (bounded between 0 and 1) or discrete class assignments.

The key distinction between linear regression and logistic regression lies in their transformation functions:

- **Linear Regression**: Uses the identity function to directly output continuous values for regression problems.
- **Logistic Regression**: Uses transformation functions to convert linear combinations into probability distributions for classification problems:
  - **Logistic function (sigmoid)** for binary classification
  - **Softmax function** for multiclass classification

3.2 Types of Classification Problems

Before diving into logistic regression specifics, it is essential to understand the different types of classification problems:

3.2.1 Binary Classification

Binary classification involves distinguishing between exactly two classes. The training set is represented as $S = \{(\mathbf{x}^{(l)}, y^{(l)})\}_{l=1}^{N}$ where $y^{(l)} \in \{0, 1\}$. Common examples include:

- Email spam detection: {spam, not spam}
- Medical diagnosis: {disease, no disease}
- Face detection: {face, non-face}
- Image classification: {dog, cat}

3.2.2 Multiclass Classification

Multiclass classification involves $K \geq 3$ classes where each input belongs to exactly one class. The training set is $S = \{(\mathbf{x}^{(l)}, \mathbf{y}^{(l)})\}_{l=1}^{N}$ where $\mathbf{y}^{(l)} \in \{0, 1\}^K$ using **one-hot encoding**. In one-hot encoding, $y_j^{(l)} = 1$ if and only if $\mathbf{x}^{(l)} \in C_j$, and all other components are 0.

Examples include:

- Animal classification: {dog, cat, cow}

- Digit recognition: {0, 1, 2, ..., 9}
- Email folder assignment in Outlook

For binary classification ($K = 2$), the second output becomes redundant since $y_2^{(l)} = 1 - y_1^{(l)}$.

3.2.3 Multilabel Classification

Multilabel classification differs from multiclass classification in that each input can belong to multiple classes simultaneously. Unlike multiclass classification where exactly one output is 1, in multilabel classification each output can independently be 0 or 1.

Multiclass	**Multilabel**
One and only one output is 1	Each output can be 0 or 1
Email folders (Outlook)	Email labels (Gmail)
Single class assignment	Multiple tags possible

Examples of multilabel classification include email tagging systems where an email can have multiple labels like "travel" and "insurance" simultaneously. Multilabel classification can be approached as multiple independent binary classification problems.

3.3 Binary Classification with Logistic Regression

3.3.1 The Sigmoid Function

The foundation of binary logistic regression is the **sigmoid function** (also called the logistic function), which maps any real number to a value between 0 and 1:

$$\boxed{\sigma(z) = \frac{1}{1+e^{-z}} \quad \forall z \in \mathbb{R}, \sigma(z) \in (0,1)} \tag{3.1}$$

The sigmoid function has several important properties:

- **S-shaped curve**: Smooth transition from 0 to 1
- **Differentiable**: Enables gradient-based optimization
- **Probabilistic interpretation**: Output can be interpreted as probability
- **Symmetric**: $\sigma(-z) = 1 - \sigma(z)$

3.3.2 Generalized Logistic Function

The sigmoid function can be generalized with a scaling parameter $a > 0$:

$$\boxed{\sigma_a(z) = \frac{1}{1+e^{-az}} \quad (a > 0)} \tag{3.2}$$

Special cases include:

- $a = 1$: Standard sigmoid $\sigma(z)$
- $a \to \infty$: Approaches step function
- $a \to 0$: Approaches constant function

The parameter a controls the steepness of the transition, providing a smooth, differentiable approximation of the step function with "switch" behavior where the output approaches 1 when the input is large and positive.

3.3.3 Logistic Regression Model

The binary logistic regression model combines a linear function with the sigmoid transformation:

$$\boxed{f(\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^T \tilde{\mathbf{x}}) = \frac{1}{1+e^{-\mathbf{w}^T \tilde{\mathbf{x}}}}} \tag{3.3}$$

where $\tilde{\mathbf{x}} = [1, x_1, x_2, \ldots, x_d]^T$ includes the bias term, and $\mathbf{w} = [w_0, w_1, w_2, \ldots, w_d]^T$ are the model parameters.

This model serves as the foundation for feedforward neural networks and can be visualized as a simple network with input nodes, weighted connections, and a sigmoid activation function.

3.3.4 Probabilistic Interpretation

The logistic regression model provides a probabilistic framework for classification:

$$\boxed{P(y=1|\mathbf{x}) = f(\mathbf{x};\mathbf{w}) = \sigma(\mathbf{w}^T\tilde{\mathbf{x}})} \tag{3.4}$$

Consequently, $P(y=0|\mathbf{x}) = 1 - f(\mathbf{x};\mathbf{w})$.

The optimization goal is to adjust the parameters $\mathbf{w}$ such that:

- For $\mathbf{x}^{(l)} \in C_1$ (where $y^{(l)} = 1$): Make $f(\mathbf{x}^{(l)};\mathbf{w}) \to 1$
- For $\mathbf{x}^{(l)} \in C_0$ (where $y^{(l)} = 0$): Make $f(\mathbf{x}^{(l)};\mathbf{w}) \to 0$

This probabilistic interpretation assumes that the output y follows a **Bernoulli distribution** parameterized by the logistic regression output.

3.4 Loss Function and Optimization

3.4.1 Maximum Likelihood Estimation

Given the probabilistic interpretation, we can derive the likelihood function for the training dataset. For a single example, the probability is:

$$P(y^{(l)}|\mathbf{x}^{(l)}) = f(\mathbf{x}^{(l)};\mathbf{w})^{y^{(l)}}(1 - f(\mathbf{x}^{(l)};\mathbf{w}))^{1-y^{(l)}} \tag{3.5}$$

For the entire dataset, assuming independence, the likelihood is:

$$\boxed{\mathcal{L}(\mathbf{w}) = \prod_{l=1}^{N} f(\mathbf{x}^{(l)};\mathbf{w})^{y^{(l)}}(1 - f(\mathbf{x}^{(l)};\mathbf{w}))^{1-y^{(l)}}} \tag{3.6}$$

3.4.2 Cross-Entropy Loss Function

To facilitate optimization, we work with the negative log-likelihood, which gives us the **cross-entropy loss function**:

$$\boxed{L(\mathbf{w};S) = -\log \mathcal{L}(\mathbf{w}) = -\sum_{l=1}^{N}\left[y^{(l)}\log f(\mathbf{x}^{(l)};\mathbf{w}) + (1-y^{(l)})\log(1 - f(\mathbf{x}^{(l)};\mathbf{w}))\right]} \tag{3.7}$$

The advantages of using the log-likelihood include:

- **Numerical stability**: Avoids numerical issues with very small probabilities
- **Convexity**: The cross-entropy loss is convex in the parameters
- **Computational efficiency**: Easier to differentiate and optimize

Remarks
The cross-entropy loss function is convex, meaning it has no local minima (though the global minimum may not be unique). This is a significant advantage over non-convex loss functions as it guarantees that gradient-based optimization methods will converge to a global optimum.

3.4.3 Gradient Descent Optimization

Since the cross-entropy loss has no closed-form solution, we must use iterative optimization algorithms. **Gradient descent** is the most commonly used approach:

$$\boxed{\mathbf{w}^{[t+1]} = \mathbf{w}^{[t]} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}^{[t]}; S)} \tag{3.8}$$

where η is the **learning rate** and $\nabla_{\mathbf{w}} L(\mathbf{w}^{[t]}; S)$ is the gradient of the loss function.

Each iteration of gradient descent involves two steps:

1. **Compute gradient**: Calculate $\nabla_{\mathbf{w}} L(\mathbf{w}^{[t]}; S)$
2. **Update weights**: Move in the direction opposite to the gradient

3.4.4 Derivative of the Sigmoid Function

A key computational advantage of the sigmoid function is that its derivative can be expressed in terms of the function itself:

Equation Derivation
Starting with $\sigma(z) = \frac{1}{1+e^{-z}}$, we can compute:

$$\sigma'(z) = \frac{d}{dz}\left(\frac{1}{1+e^{-z}}\right) \tag{3.9}$$

$$= \frac{0 \cdot (1+e^{-z}) - 1 \cdot (-e^{-z})}{(1+e^{-z})^2} \tag{3.10}$$

$$= \frac{e^{-z}}{(1+e^{-z})^2} \tag{3.11}$$

$$= \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}} \tag{3.12}$$

$$= \sigma(z) \cdot (1 - \sigma(z)) \tag{3.13}$$

$$\boxed{\sigma'(z) = \sigma(z)(1-\sigma(z))} \tag{3.14}$$

This property is crucial for efficient gradient computation in neural networks and allows us to compute the derivative using only the function output.

3.4.5 Gradient Computation

To compute the gradient of the cross-entropy loss, we need the partial derivatives with respect to each weight w_i:

Equation Derivation
For the logistic regression model $f(\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^T \tilde{\mathbf{x}})$:

$$\frac{\partial f}{\partial w_i} = \frac{\partial \sigma(\mathbf{w}^T \tilde{\mathbf{x}})}{\partial w_i} \tag{3.15}$$

$$= \sigma'(\mathbf{w}^T \tilde{\mathbf{x}}) \cdot \frac{\partial (\mathbf{w}^T \tilde{\mathbf{x}})}{\partial w_i} \tag{3.16}$$

$$= \sigma(\mathbf{w}^T \tilde{\mathbf{x}})(1 - \sigma(\mathbf{w}^T \tilde{\mathbf{x}})) \cdot x_i \tag{3.17}$$

$$= f(\mathbf{x}; \mathbf{w})(1 - f(\mathbf{x}; \mathbf{w}))x_i \tag{3.18}$$

The gradient of the cross-entropy loss with respect to w_i is:

$$\boxed{\frac{\partial L}{\partial w_i} = -\sum_{l=1}^{N} \left(y^{(l)} - f(\mathbf{x}^{(l)}; \mathbf{w}) \right) x_i^{(l)} = -\sum_{l=1}^{N} \delta^{(l)} x_i^{(l)}} \tag{3.19}$$

where $\delta^{(l)} = y^{(l)} - f(\mathbf{x}^{(l)}; \mathbf{w})$ represents the prediction error for the l-th example.

3.4.6 Weight Update Rule

The batch gradient descent update rule for logistic regression becomes:

$$\boxed{\Delta w_i = \eta \sum_{l=1}^{N} \left(y^{(l)} - f(\mathbf{x}^{(l)}; \mathbf{w}) \right) x_i^{(l)}} \tag{3.20}$$

In vector form:

$$\boxed{\Delta \mathbf{w} = \eta \sum_{l=1}^{N} \left(y^{(l)} - f(\mathbf{x}^{(l)}; \mathbf{w}) \right) \tilde{\mathbf{x}}^{(l)}} \tag{3.21}$$

The interpretation is intuitive: the weight change is proportional to the input magnitude times the prediction error, summed over all training examples. Larger errors and larger input values lead to larger weight updates.

3.5 Multiclass Classification with Softmax

3.5.1 Extension to Multiple Classes

For multiclass classification with $K \geq 3$ classes, we extend the binary logistic regression approach using the **softmax function**. The model now requires a weight matrix $\mathbf{W} \in \mathbb{R}^{K \times (d+1)}$ where each row $\mathbf{w}_k$ corresponds to class C_k.

3.5.2 The Softmax Function

The softmax function generalizes the sigmoid function to multiple classes:

$$\boxed{\text{softmax}(\mathbf{z})_j = r_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \in (0, 1)} \tag{3.22}$$

where $\mathbf{z} = [z_1, z_2, \ldots, z_K]^T$ are the linear combinations for each class.

Key properties of the softmax function include:

- **Probability distribution**: $\sum_{j=1}^{K} r_j = 1$
- **"Rich gets richer"**: Amplifies the largest inputs while suppressing smaller ones
- **Differentiable**: Provides a smooth alternative to the argmax function
- **Generalization**: Reduces to sigmoid for $K = 2$

3.5.3 Multiclass Logistic Regression Model

The multiclass logistic regression model is defined as:

$$\boxed{f(\mathbf{x}; \mathbf{W}) = \text{softmax}(\mathbf{W}\tilde{\mathbf{x}}) = \mathbf{r}} \tag{3.23}$$

where $\mathbf{r} = [r_1, r_2, \ldots, r_K]^T$ represents the probability distribution over the K classes.

The optimization goal is to adjust the weight matrix $\mathbf{W}$ such that:

- For $\mathbf{x}^{(l)} \in C_k$: Make $r_k^{(l)} \to 1$
- For all $j \neq k$: Make $r_j^{(l)} \to 0$

3.5.4 Softmax Derivative

The derivative of the softmax function has an elegant form:

$$\boxed{\frac{\partial r_j}{\partial z_k} = r_j(\delta_{jk} - r_k)} \tag{3.24}$$

where δ_{jk} is the Kronecker delta ($\delta_{jk} = 1$ if $j = k$, and 0 otherwise).

3.5.5 Multiclass Cross-Entropy Loss

The likelihood for multiclass classification is:

$$\boxed{\mathcal{L}(\mathbf{W}) = \prod_{l=1}^{N}\prod_{j=1}^{K}(r_j^{(l)})^{y_j^{(l)}}} \tag{3.25}$$

The corresponding cross-entropy loss function is:

$$\boxed{L(\mathbf{W}; S) = -\sum_{l=1}^{N}\sum_{j=1}^{K} y_j^{(l)} \log r_j^{(l)}} \tag{3.26}$$

This loss function maintains the convex property but requires iterative optimization as no closed-form solution exists.

3.5.6 Multiclass Gradient Computation

The gradient of the multiclass cross-entropy loss with respect to weight w_{ki} is:

$$\boxed{\frac{\partial L}{\partial w_{ki}} = -\sum_{l=1}^{N}(y_k^{(l)} - r_k^{(l)})x_i^{(l)}} \tag{3.27}$$

The batch update rule becomes:

$$\boxed{\Delta w_{ki} = \eta\sum_{l=1}^{N}(y_k^{(l)} - r_k^{(l)})x_i^{(l)}} \tag{3.28}$$

In vector form for each class k:

$$\boxed{\Delta \mathbf{w}_k = \eta\sum_{l=1}^{N}(y_k^{(l)} - r_k^{(l)})\tilde{\mathbf{x}}^{(l)}} \tag{3.29}$$

3.6 Regularization in Logistic Regression

Similar to linear regression, logistic regression can benefit from regularization to prevent overfitting, especially when the number of features is large relative to the number of training examples.

3.6.1 L_2 Regularization

For binary logistic regression, L_2 regularization adds a penalty term to the cross-entropy loss:

$$\boxed{L_{\text{reg}}(\mathbf{w}; S) = L(\mathbf{w}; S) + \lambda \|\mathbf{w}\|^2} \tag{3.30}$$

For multiclass logistic regression, we use the Frobenius norm of the weight matrix:

$$\boxed{L_{\text{reg}}(\mathbf{W}; S) = L(\mathbf{W}; S) + \lambda \|\mathbf{W}\|_F^2} \tag{3.31}$$

where $\|\mathbf{W}\|_F^2 = \sum_{k=1}^{K} \sum_{i=1}^{d+1} w_{ki}^2$ is the Frobenius norm.

Practical considerations for regularization include:

- $\lambda > 0$: Regularization strength parameter
- Typically exclude bias terms from regularization
- Prevents overfitting by controlling model complexity
- Requires tuning via cross-validation

3.7 Performance Metrics for Classification

Evaluating classification models requires different metrics than regression. While accuracy is the most intuitive metric, it can be misleading for imbalanced datasets or when different types of errors have different costs.

3.7.1 Confusion Matrix

The **confusion matrix** provides a comprehensive view of classification performance for binary classification:

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

From the confusion matrix, we can derive several important metrics:

$$\boxed{\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}} \tag{3.32}$$

3.7.2 Precision, Recall, and F1 Score

Precision measures the exactness of positive predictions:

$$\boxed{P = \frac{\text{TP}}{\text{TP} + \text{FP}}} \tag{3.33}$$

Recall (also called sensitivity) measures the completeness of positive predictions:

$$\boxed{R = \frac{\text{TP}}{\text{TP} + \text{FN}}} \tag{3.34}$$

F1 Score provides a balanced measure by computing the harmonic mean of precision and recall:

$$\boxed{F1 = 2 \cdot \frac{P \cdot R}{P + R} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}} \tag{3.35}$$

The F1 score is particularly useful for imbalanced datasets where accuracy can be misleading.

Example
Consider a binary classification problem with the following confusion matrix:

- TP = 5, TN = 90, FP = 0, FN = 5
- Total examples = 100

Computing the metrics:

$$\text{Accuracy} = \frac{95}{100} = 0.95 \tag{3.36}$$

$$\text{Precision} = \frac{5}{5} = 1.0 \tag{3.37}$$

$$\text{Recall} = \frac{5}{10} = 0.5 \tag{3.38}$$

$$\text{F1} = 2 \cdot \frac{1.0 \cdot 0.5}{1.0 + 0.5} = \frac{2}{3} \approx 0.667 \tag{3.39}$$

While the accuracy appears high at 95%, the low recall (50%) indicates that the model misses half of the positive cases. The F1 score of 0.667 provides a more balanced assessment of performance.

Remarks
For imbalanced datasets, accuracy can be misleading. A model that always predicts the majority class will achieve high accuracy but provide no useful information about the minority class. Precision, recall, and F1 score provide more meaningful insights into model performance across different classes.