Theorem[section] Algorithm[section]

CS 412 — Introduction to Machine Learning (UIC) February 16, 2025

Lecture 2

Instructor: Aadirupa Saha *Scribe(s): Harsh Shelke*

[This draft is not fully proofread. Please email any typos/errors to the instructor or directly edit the latex file.]

Overview

In this scribe for CS 412 — Introduction to Machine Learning, I've summarized key concepts from Lecture 2 by Aadirupa Saha, covering essential prediction tasks like classification, multi-class classification, and regression. This scribe also explores empirical risk minimization, different hypothesis classes, and the fundamentals of linear regression with regularization.

1 Types of Prediction Tasks

Machine learning involves various prediction tasks that differ in their output space and the nature of the prediction being made. Understanding these different tasks is fundamental to selecting appropriate models and evaluation metrics. Let's explore the main categories:

1.1 Classification

Classification is one of the most common prediction tasks in machine learning, where the goal is to categorize input data into predefined classes or categories.

1.1.1 Binary Classification

In binary classification, we assign inputs to one of two possible classes. This is widely used in applications like spam detection, medical diagnosis, and sentiment analysis.
The function f maps input data X to one of two discrete classes, usually represented as 0 or 1. This is the standard binary classification problem.

1.1.2 Generalized Classification

In practice, many classification algorithms don't directly output hard class labels but instead produce a continuous value representing the probability or confidence of class membership.

$$f : X \to [0, 1]$$

In this generalized form, the function outputs a value in the continuous interval $[0, 1]$, representing the probability of belonging to a certain class rather than a hard label.
This probabilistic output is particularly useful when we need to:

- Adjust decision thresholds based on the cost of different types of errors

- Rank instances by their likelihood of belonging to a class
- Incorporate uncertainty into downstream decision-making processes

1.2 m-Multiclass Classification

When dealing with more than two classes, we enter the domain of multiclass classification. Examples include digit recognition, object classification in images, and language identification.

$$f : X \to \{1, 2, \ldots, m\}$$

This represents multiclass classification, where the output is one of m possible classes.
In a more generalized sense, similar to binary classification, we can model the output as a probability distribution over all possible classes:

Probabilistic Multiclass Classification: The function maps the input to a probability distribution over m classes:

$$f : X \to \Delta_m$$

Where Δ_m is the set of all probability distributions over m classes. Each $p \in \Delta_m$ is a vector of probabilities:

$$p = (p_1, p_2, \ldots, p_m)$$

where:

$$p_i \geq 0 \text{ for all } i, \quad \sum_{i=1}^{m} p_i = 1$$

his probabilistic approach allows us to:

- Quantify uncertainty across multiple classes
- Implement more sophisticated decision rules
- Combine predictions with other information sources

1.3 Regression

Unlike classification, regression tasks involve predicting continuous numerical values. This is essential for applications like price prediction, demand forecasting, and physical measurements.

$$f : X \to \mathbb{R}$$

In a regression task, the function maps input data X to a real-valued number $\mathbb{R}$.
Regression differs fundamentally from classification in that:

- The output space is continuous rather than discrete
- The prediction errors are measured in terms of distance (e.g., absolute or squared difference)
- The goal is to approximate a continuous function rather than decision boundaries

Common applications include predicting house prices, temperature forecasting, and estimating physical quantities based on sensor measurements.

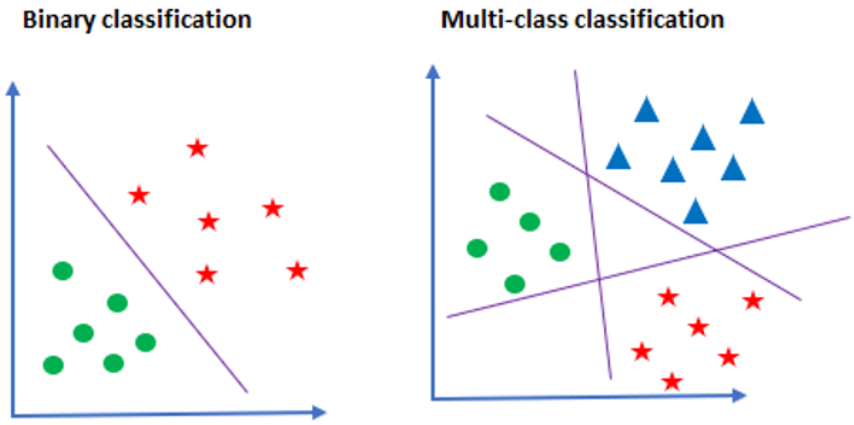

Figure 1: Binary Classification vs Multi Classification

2 Our Goal in Machine Learning

The fundamental goal in supervised machine learning is to find a function that can accurately map inputs to outputs based on observed data.
Given a dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$, we aim to find a "good" function f that can generalize beyond the training examples.
Here, D is the training dataset consisting of n samples, where each sample is a pair (x_i, y_i):

- $x_i \in X$ is the input (feature vector) from the input space X
- $y_i \in Y$ is the output (label) from the output space Y

$$f : X \to Y$$

The key challenge is finding a function that not only fits the training data well but also generalizes to unseen data. This leads us to the concept of empirical risk minimization.

3 Empirical Risk Minimization (ERM)

Empirical Risk Minimization (ERM) is a fundamental principle in machine learning that provides a systematic approach to finding the optimal function from a hypothesis class.

ERM Principle: We aim to minimize the empirical risk (or empirical loss), which is the average loss over all training samples:

$$\min_{f} \frac{1}{n} \sum_{i=1}^{n} l(y_i, f(x_i))$$

where l is a loss function that measures how far $f(x_i)$ is from the true label y_i.
This can also be expressed as:

$$\arg \min_{f:X \to Y} \sum_{i=1}^{n} l(y_i, f(x_i))$$

This is known as the Empirical Risk Minimization (ERM) framework, where the goal is to find a function f that minimizes the total error on the training data.

3.1 Steps in ERM

The ERM process involves two key steps:

Empirical Risk Minimization Process

1: **Choose a Loss Function** $l : Y \times \hat{Y} \to \mathbb{R}^+$ that maps the predicted label and true label to a non-negative real number representing the error.
2: Example loss functions include:
   - 0-1 Loss for classification: $l(y, \hat{y}) = \mathbb{1}(y \neq \hat{y})$
   - Squared Loss for regression: $l(y, \hat{y}) = (y - \hat{y})^2$
   - Log Loss for probabilistic classification: $l(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$
3: **Find** $f : X \to Y$ that minimizes the empirical risk:

$$\sum_{i=1}^{n} l(y_i, f(x_i))$$

3.2 Assumption for Simplicity

To make the problem more tractable, let's make some simplifying assumptions:

- $X \subseteq \mathbb{R}^d$: Inputs are d-dimensional vectors in the real space.
- $Y = \hat{Y} = \mathbb{R}$: Outputs are real numbers (i.e., this is a regression problem).

With these assumptions, we can choose an appropriate loss function:

Loss Function for Regression: For regression problems, we typically choose the squared loss:

$$l(y, \hat{y}) = (y - \hat{y})^2$$

This measures the square of the difference between the true label y and the predicted value $\hat{y}$.
The squared loss has several desirable properties:

- It penalizes larger errors more heavily than smaller ones
- It's differentiable everywhere, making optimization easier
- It has connections to maximum likelihood estimation under Gaussian noise assumptions

Final Optimization Problem: Under these assumptions, the ERM problem becomes:

$$\arg \min_{f:\mathbb{R}^d \to \mathbb{R}} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

Our goal here is to find a function f that maps d-dimensional vectors to real numbers and minimizes the sum of squared errors. But what kind of functions should we consider? This brings us to the concept of hypothesis classes.

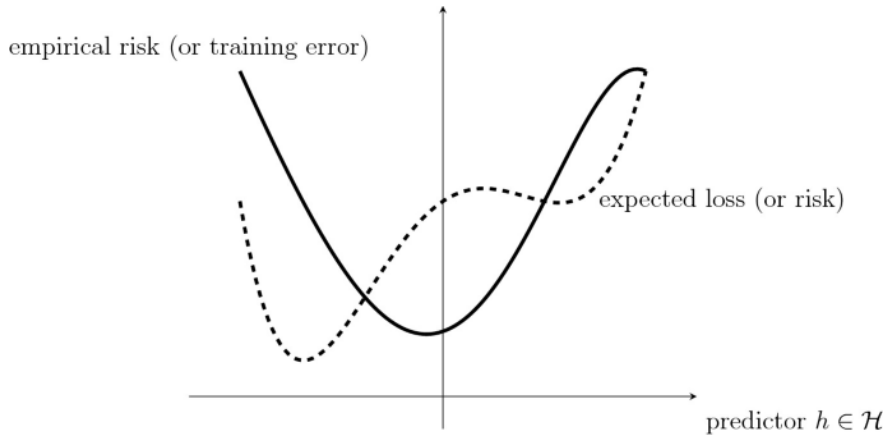

Figure 2: Empirical Risk Minimization

4 Hypothesis/Function Class

The hypothesis class (or function class) is a crucial concept in machine learning that defines the set of functions we're willing to consider as potential solutions.

Hypothesis Class: The hypothesis or function class is the set of all candidate functions f that map from the input space X to the output space Y.

We consider the input space $X \subseteq \mathbb{R}^d$ (d-dimensional real vectors) and the output space $Y = \mathbb{R}$ (real numbers). The choice of hypothesis class represents a fundamental trade-off in machine learning:

- Too restrictive: May not capture the true underlying pattern (underfitting)
- Too flexible: May fit noise in the training data (overfitting)

Let's explore some common hypothesis classes:

4.1 Linear Function Example

The linear function class is one of the simplest and most widely used hypothesis classes in machine learning.
Linear Function Class: The hypothesis class $\mathcal{F}_\Theta^{\text{lin}}$ is the set of all linear functions:

$$\mathcal{F}_\Theta^{\text{lin}} = \{f : \mathbb{R}^d \to \mathbb{R} \mid f(x) = w^T x + b, \forall x \in X\}$$

where:

$$w \in \mathbb{R}^d \text{ is a weight vector,}$$

$$b \in \mathbb{R} \text{ is the bias term.}$$

This set is parameterized by the pair (w, b), meaning each function f in this class is uniquely defined by a specific choice of w and b.
Each linear function $f_{\text{lin}}(w, b)$ is expressed as:

$$f_{\text{lin}}(w, b) = w^T x + b = \sum_{i=1}^{d} w_i x_i + b$$

where

$$w = (w_1, w_2, \ldots, w_d)$$

is a fixed vector of weights, and b is a scalar bias term.
The difference between two functions in this class is noted as:

$$\text{diff}(f) = f_{\text{lin}}(w, b) - f_{\text{lin}}(\tilde{w}, \tilde{b}) = (w - \tilde{w})^T x + (b - \tilde{b})$$

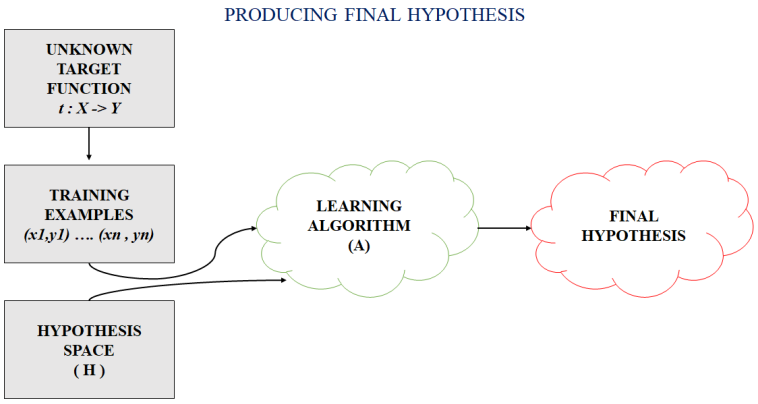

Figure 3: Linear Function Class.

Parameter Class/Set: The parameter class Θ is the set of all parameters (w, b) that define the linear functions in the hypothesis class:

$$\Theta = \{(w, b) \in \mathbb{R}^{d+1} \mid w \in \mathbb{R}^d, b \in \mathbb{R}\}$$

This represents all possible weight vectors and bias terms, and thus all possible linear functions in d-dimensional space.
This means

$$\Theta \subseteq \mathbb{R}^{d+1},$$

indicating that the parameter set is a subset of the $(d + 1)$-dimensional real space.

4.2 General Function Class

We can generalize the concept of a parameterized function class beyond linear functions.

Parameterized Function Class: The definition of a general function class is as follows:

$$\mathcal{F}_\Theta = \{f_\theta : X \to Y \mid f_\theta(x) = f(x;\theta),\ \forall \theta \in \Theta\}.$$

Here, $\mathcal{F}_\Theta$ is the set of all functions f_θ that map from the input space X to the output space Y.
Each function f_θ is parameterized by θ, which belongs to the parameter set Θ.
Essentially, Θ is the collection of all possible parameters that can define functions in the class.

4.3 Examples of Function Classes

Let's explore several important function classes that are commonly used in machine learning:

4.3.1 Constant Functions

The simplest possible function class consists of constant functions, which output the same value regardless of the input.

Constant Function Class:

$$\mathcal{F}_\Theta^{\text{const}} = \{f_\theta : X \to Y \mid f_\theta(x) = c,\ \forall x \in X\}\,.$$

Each function f_θ is simply a constant value c for all inputs x. Here, the parameter θ maps directly to the constant c. The parameter class Θ in this case is the set of all real numbers $\mathbb{R}$, since any real number can serve as the constant c.
While extremely simple, constant functions can serve as baselines or as components in more complex models.

4.3.2 Linear Function Class

The linear function class is fundamental in machine learning and serves as the building block for many more complex models.

Theorem 1 (Linear Function Class)**.**

$$\mathcal{F}_\Theta^{lin} = \{f_\theta : X \to Y \mid f_\theta(x) = w^T x + b\}$$

Let's break down the components:

- w^T is a transposed weight vector (like $[w_1, w_2, ..., w_d]$)
- x is the input vector (like $[x_1, x_2, ..., x_d]$)
- $w^T x$ represents the dot product ($w_1x_1 + w_2x_2 + ... + w_dx_d$)
- b is a bias term (like the y-intercept in a line equation)

The parameters $\theta = (w, b) \in \mathbb{R}^{d+1}$ means:

- w comes from $\mathbb{R}^d$ (d-dimensional real space)
- b comes from $\mathbb{R}$ (real numbers)
- Together they form a $(d+1)$-dimensional parameter vector

Linear functions have several advantages:

- They're simple to understand and interpret
- They're computationally efficient to train
- They often serve as good baselines
- They can be extended to more complex models

4.3.3 Quadratic Function Class

When linear functions aren't expressive enough to capture the underlying patterns, we can move to quadratic functions.

Quadratic Function Class:

$$\mathcal{F}_{\Theta}^{\text{quad}} = \{f_\theta : X \to Y \mid f_\theta(x) = x^T Hx + w^T x + b\}$$

Let's break this down:

- $x^T Hx$ is the quadratic term:
  - H is a $d \times d$ matrix of parameters
  - This creates quadratic relationships between inputs
- $w^T x$ is the same linear term as before
- b is still the bias term

The parameters $\theta = (H, w, b)$ now include:

- $H \in \mathbb{R}^{d \times d}$ (a matrix of parameters)
- $w \in \mathbb{R}^d$ (the linear weights)
- $b \in \mathbb{R}$ (the bias)

The key concept here is that these function classes represent different levels of complexity:

- Linear functions can only create straight lines/planes
- Quadratic functions can create curves and more complex surfaces

Think of it like this: if you're trying to fit data points:

- Linear functions can only draw straight lines through them
- Quadratic functions can draw curves, making them more flexible but also more complex to optimize

4.3.4 Polynomial Feature Map

Feature maps allow us to transform the input space into a higher-dimensional space where linear models can capture non-linear relationships in the original space.

Polynomial Feature Map:

$$\varphi : \mathcal{X} \to \mathbb{R}^p \text{ (Feature Map)} \tag{1}$$

$$\text{where } \varphi(x) = (1, x_1, \ldots, x_k, x_1^2, \ldots, x_k^2) \tag{2}$$

Purpose:

- Transforms input data into higher-dimensional space to capture non-linear relationships
- Makes non-linearly separable data linearly separable in higher dimensions

Example: Let's say you have a 2D input $x = (x_1, x_2)$:

- Original features: (x_1, x_2)
- After polynomial mapping (degree 2):

$$\varphi(x) = (1, x_1, x_2, x_1^2, x_1x_2, x_2^2) \tag{3}$$

- This transforms 2D data into 6D space

This approach allows us to:

- Use linear models in the transformed space to capture non-linear relationships in the original space
- Control the complexity of the model by choosing the degree of the polynomial
- Apply the "kernel trick" for efficient computation in high-dimensional spaces

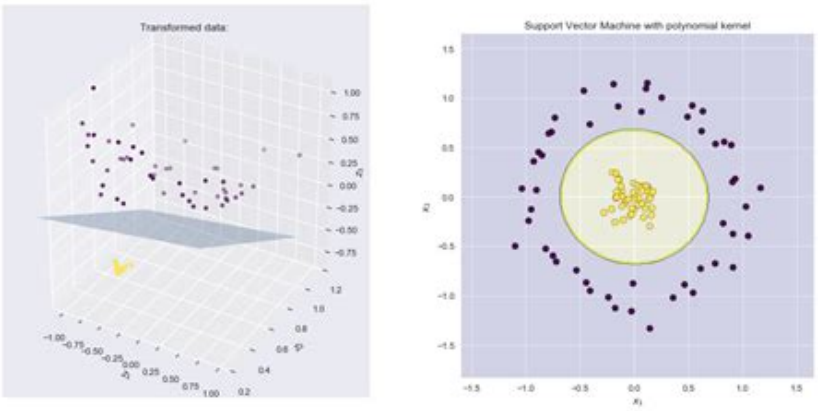

Figure 4: Polynomial Kernel

5 Classification Functions

Classification functions are specifically designed to map inputs to discrete classes or probabilities. Let's examine two important types:

5.1 Threshold Classification

Threshold classification creates a decision boundary by applying a threshold to a linear function.

Threshold Classification: This function maps inputs to binary outputs $\{0, 1\}$:

$$f_\theta(x) = \begin{cases} 1 & \text{if } w^T x + b > c \\ 0 & \text{if } w^T x + b \leq c \end{cases}$$

Where:

- w is the weight vector
- b is the bias
- c is the threshold
- $\theta = (w, b, c) \in \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}$

The parameter class $\Theta = \mathbb{R}^{d+2}$.
This creates a decision boundary that separates data into two classes. Geometrically, this boundary is a hyperplane in the input space defined by the equation $w^T x + b = c$.
Threshold classifiers have several important properties:

- They make hard decisions, assigning each input to exactly one class
- The decision boundary is linear (a hyperplane)
- They're simple and interpretable
- They can be sensitive to outliers

5.2 Logistic Function

Unlike threshold classifiers, logistic functions provide smooth probability outputs rather than hard decisions.

Logistic Function: This maps inputs to probabilities $[0, 1]$:

$$f_\theta(x) = \frac{1}{1 + e^{-(w^T x + b)}}$$

Where:

- The function produces a smooth S-shaped curve
- It approaches 0 and 1 asymptotically

- $\theta = (w, b)$ are the parameters
- $\Theta = \mathbb{R}^{d+1}$ is the parameter class

The logistic function has several advantages:

- It outputs probabilities rather than hard decisions
- It's differentiable everywhere, making optimization easier
- It has a natural interpretation in terms of log-odds
- It forms the basis for logistic regression, a widely used classification algorithm

Key Differences and Relationships:

- Polynomial functions are useful for capturing non-linear relationships
- Threshold classifiers are good for clear-cut binary decisions
- Logistic functions are ideal for probabilistic classification

The choice of function class depends on the specific requirements of the problem, including:

- The nature of the data and underlying relationships
- The need for interpretability versus predictive power
- Computational constraints
- The importance of uncertainty quantification

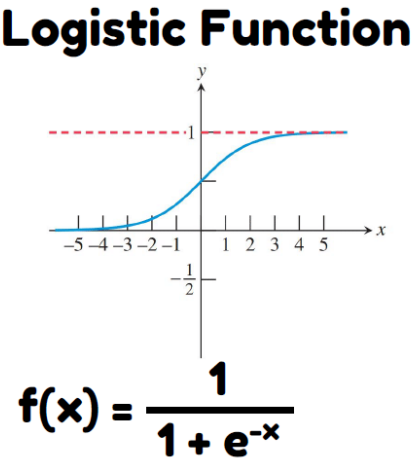

Figure 5: Logistic Regression.

6 Linear Regression Task Setup

Linear regression is one of the most fundamental and widely used machine learning algorithms. Let's explore its formulation in detail.

6.1 Understanding the Spaces

Before diving into the algorithm, it's important to understand the spaces involved:
Input Space: $X \subseteq \mathbb{R}^d$
This means our input space X is a subset of the d-dimensional real numbers. Each input $x \in X$ is a vector with d dimensions, representing the features or predictors.
Output Space: $Y = \mathbb{R}$
The output space Y is the real number line, meaning our target values are real numbers. This is what distinguishes regression from classification.
Predicted Output Space: $\hat{Y} = \mathbb{R}$
The predicted output space $\hat{Y}$ is also the real number line, matching the true output space.

6.2 The Function Class

In linear regression, we restrict ourselves to the class of linear functions:

Linear Function Class for Regression: The linear function class is defined as:

$$\mathcal{F}_{\Theta}^{\text{lin}} = \{f_\theta : \mathbb{R}^d \to \mathbb{R} \mid f_\theta(x) = w^T x + b\}$$

where:

- $w \in \mathbb{R}^d$ is the weight vector
- $b \in \mathbb{R}$ is the bias term

Each function $f_\theta(x)$ takes a d-dimensional input x, produces a real-valued output, and uses a weight vector w and a bias term b. The dot product $w^T x$ is calculated as:

$$w^T x = \sum_{j=1}^{d} w_j x_j$$

This linear relationship means that:

- Each feature contributes independently to the prediction
- The effect of changing a feature is constant across its range
- The relationship between features and the target is assumed to be linear

6.3 Loss Function

For regression problems, the squared loss function is the most common choice:

Squared Loss Function: The loss function $l : Y \times \hat{Y} \to \mathbb{R}^+$ is defined as:

$$l(y, \hat{y}) = (y - \hat{y})^2$$

The squared loss has several important properties:

- It penalizes larger errors more heavily than smaller ones
- It's differentiable everywhere, making optimization easier
- It leads to a closed-form solution for linear regression
- It corresponds to maximum likelihood estimation under Gaussian noise assumptions

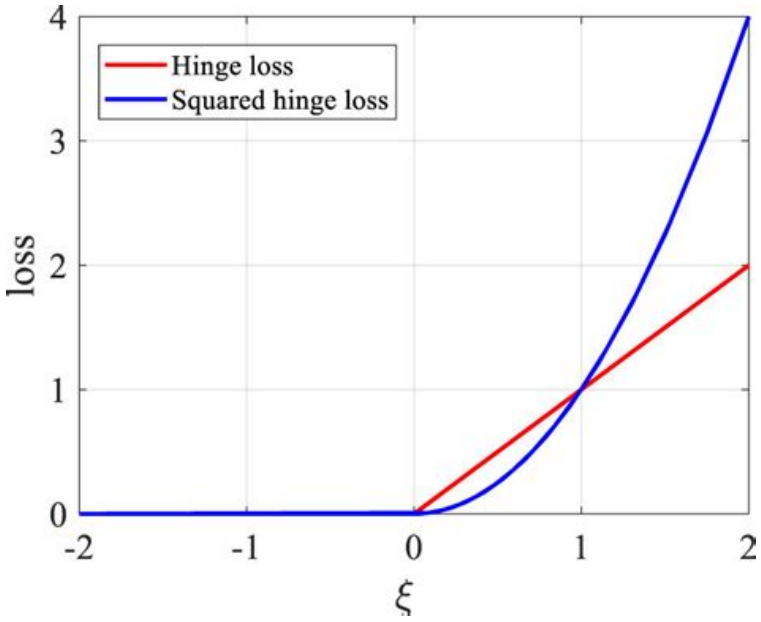

Figure 6: Loss Functions

6.4 Empirical Risk Minimization (ERM)

In linear regression, the ERM objective function often includes a regularization term:
Regularized ERM for Linear Regression:

$$\sum_{i=1}^{n} l(f_\theta(x_i), y_i) + \lambda \|w\|_2^2$$

where:

- $\sum_{i=1}^{n}$ indicates summation over all n training examples.
- $(w^T x_i + b - y_i)^2$ is the squared error loss for each training example i.
- $\lambda \|w\|_2^2$ is the regularization term, where $\lambda \geq 0$ and $\|w\|_2^2 = \sum_j w_j^2$.

λ (Lambda):
Lambda is the regularization parameter. Its role is to control the amount of regularization applied to the model.

When λ is large, it increases the penalty for large weights, thus simplifying the model and potentially preventing overfitting.
When λ is small (or zero), the model focuses more on minimizing the loss on the training data, which might lead to overfitting.

$\|w\|_2^2$ (L2 norm of weights):
This is the L2 regularization term (also known as Ridge regularization). It is the sum of the squares of all the weight parameters:

$$\|w\|_2^2 = \sum_j w_j^2$$

Adding this term to the objective function helps prevent the weights from becoming too large, thus improving the model's generalization to unseen data.

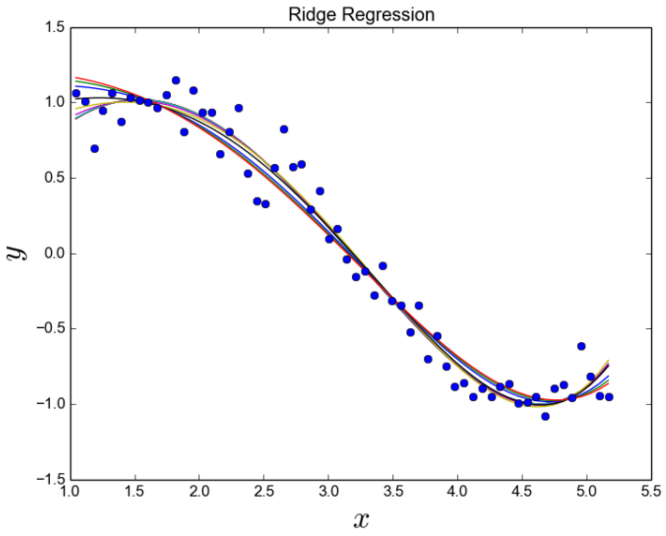

Figure 7: L2 Norm(Ridge Regression)

6.5 Data Representation

In practice, we often represent our data in matrix form for efficient computation:

- Target Vector: $Y = [y_1, y_2, \ldots, y_n]^T \in \mathbb{R}^n$
- Input Feature Matrix: $X = [x_1, x_2, \ldots, x_n]^T \in \mathbb{R}^{n \times d}$

This matrix representation allows us to express the predictions for all examples concisely as $Xw + b\mathbf{1}$, where $\mathbf{1}$ is a vector of ones.

6.6 Optimization Problem

Putting everything together, the optimization problem for regularized linear regression is:

Linear Regression Optimization:

$$\min_{w,b} \left(\sum_{i=1}^{n} (w^T x_i + b - y_i)^2 + \lambda \sum_{j=1}^{d} w_j^2 \right)$$

The goal is to find w and b that minimize this objective function, balancing between fitting the training data well and keeping the model simple through regularization.

7 Linear Regression with Regularization: Derivation Process

Let's derive the closed-form solution for regularized linear regression step by step.

7.1 Step 1: Setting up the Problem

We begin by representing our parameters and data in a convenient form:

Problem Setup

We represent our parameter vector as:

$$\theta = \begin{bmatrix} w \\ b \end{bmatrix}, \quad \text{where } w \in \mathbb{R}^d \text{ and } b \in \mathbb{R}.$$

Our input data is represented as:

$$\tilde{X}_i = \begin{bmatrix} x_i \\ 1 \end{bmatrix} \in \mathbb{R}^{d+1}.$$

The prediction is given by:

$$\hat{y}_i = w^T x_i + b = \theta^T \tilde{X}_i.$$

This representation allows us to treat the bias term b as just another parameter, simplifying our derivation.

7.2 Step 2: The Optimization Problem

We can now express our objective function in matrix form:

Matrix Form of Optimization:

We aim to minimize the following objective:

$$\sum_{i=1}^{n}(w^T x_i + b - y_i)^2 + \lambda\|w\|_2^2.$$

This can be rewritten in matrix form as:

$$\|y - \tilde{X}\theta\|^2 + \lambda\|w\|_2^2.$$

Where $\tilde{X}$ is the augmented feature matrix that includes a column of ones for the bias term, and θ is our combined parameter vector.

7.3 Step 3: Finding the Solution

To find the minimum, we take the gradient with respect to θ and set it to zero:
Gradient-Based Solution

$$\nabla_\theta[\|y - \tilde{X}\theta\|^2 + \lambda\|w\|_2^2] = 0.$$

This results in:

$$-2\tilde{X}^T(y - \tilde{X}\theta) + \lambda \begin{bmatrix} I_d \\ 0 \end{bmatrix} \theta = 0.$$

Where I_d is the $d \times d$ identity matrix, and the last row of zeros corresponds to not regularizing the bias term.
Simplifying further:

$$-2\tilde{X}^T y + 2\tilde{X}^T\tilde{X}\theta + \lambda \begin{bmatrix} I_d \\ 0 \end{bmatrix} \theta = 0.$$

7.4 Step 4: Solving for θ

Closed-Form Solution
Rearranging terms, we have:

$$(2\tilde{X}^T\tilde{X} + \lambda \begin{bmatrix} I_d & 0 \\ 0 & 0 \end{bmatrix})\theta = 2\tilde{X}^T y.$$

Thus, the solution is:

$$\theta^* = (2\tilde{X}^T\tilde{X} + \lambda \begin{bmatrix} I_d & 0 \\ 0 & 0 \end{bmatrix})^{-1} 2\tilde{X}^T y.$$

Simplifying and focusing on the weight vector w, we get:

$$w^* = (X^T X + \lambda I)^{-1} X^T y.$$

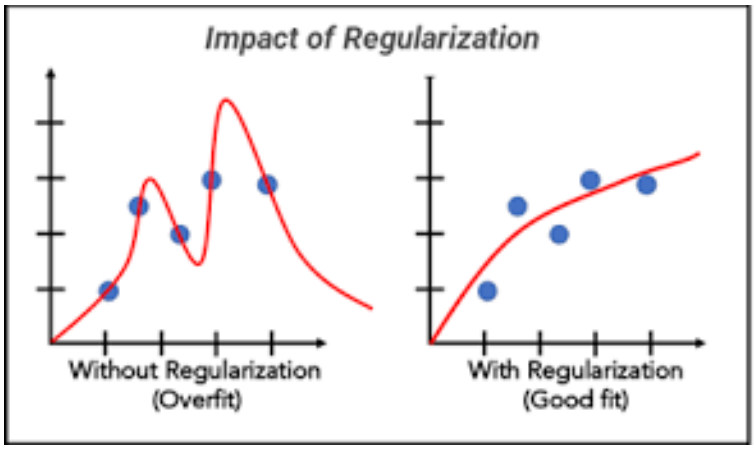

Figure 8: Linear Regression with regularization

7.5 Key Points to Understand

- The λI term adds to the diagonal of $X^T X$, making it invertible even when $X^T X$ is singular.
- When $\lambda = 0$, this reduces to the standard least squares solution: $w^* = (X^T X)^{-1} X^T y$.
- As λ increases, it puts more emphasis on keeping the weights small.
- The solution balances between fitting the data (minimizing $\|y - Xw - b\mathbf{1}\|^2$) and keeping the model simple (minimizing $\|w\|_2^2$).
- The first d components of θ^* give w^*, and the last component gives b^*.

Properties of the Solution

The regularized linear regression solution has several important properties:

- Closed-form solution (no iterative optimization needed).
- Unique global minimum (the objective function is strictly convex).
- Stability due to regularization (less sensitive to small changes in the data).
- Computational efficiency (can be computed directly using matrix operations).
- Interpretability (the weights directly indicate the importance of features).

Next Lecture

The next lecture will cover the following topics:
(i) Logistic Regression,
(ii) Maximum Likelihood Estimation,
(iii) Gradient Descent Optimization.

References:

1. Machine Learning by Tom Mitchell
2. Lecture notes by Prof. Aadirupa Saha from course CS 412 - Introduction to Machine Learning
3. The Elements of Statistical Learning by Hastie, Tibshirani, and Friedman-Section 3.2
4. Probabilistic Machine Learning: An Introduction, by Kevin Murphy.-Sections: 11.1, 11.2,11.3