Machine Learning Methods

Yonghao Lee

January 2026

Contents

1 Multivariate Calculus for Machine Learning

Machine learning relies heavily on optimization. To train models, we must understand how changes in input parameters (weights) affect the output (loss).

1.1 The Gradient

The gradient is the multi-dimensional generalization of the derivative. It points in the direction of the steepest ascent (the direction of fastest increase).

Definition 1.1: The Gradient

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a scalar-valued function. The gradient of f at point $\mathbf{x} \in \mathbb{R}^d$ is the vector of all partial derivatives:

$$\nabla f(\mathbf{x}) := \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_d} \end{pmatrix} \in \mathbb{R}^d$$

Example 1.1: Linear Function

Let $\mathbf{w} \in \mathbb{R}^d$ and define $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. The gradient is simply the weight vector:

$$\nabla f(\mathbf{x}) = \mathbf{w}$$

Example 1.2: Quadratic Function (L_2 Norm)

Let $f(\mathbf{x}) = \|\mathbf{x}\|^2 = \sum x_i^2$. The gradient is:

$$\nabla f(\mathbf{x}) = 2\mathbf{x}$$

1.2 Jacobians: Vector-to-Vector Functions

When a function maps a vector to another vector ($f : \mathbb{R}^m \to \mathbb{R}^d$), the derivative is a matrix called the **Jacobian**.

Definition 1.2: The Jacobian Matrix

Let $\mathbf{f} : \mathbb{R}^m \to \mathbb{R}^d$. The Jacobian $J_{\mathbf{x}}(\mathbf{f})$ is a $d \times m$ matrix where the i-th row is the transpose of the gradient of the i-th output f_i:

$$J_{\mathbf{x}}(\mathbf{f}) = \begin{bmatrix} \nabla f_1(\mathbf{x})^T \\ \vdots \\ \nabla f_d(\mathbf{x})^T \end{bmatrix}$$

1.3 The Multivariate Chain Rule

This is the engine behind **Backpropagation** in Neural Networks.

$$J_{f \circ g}(\mathbf{x}) = J_f(g(\mathbf{x})) \cdot J_g(\mathbf{x})$$

2 Statistical Learning Theory

Statistical Learning Theory explains *why* learning is possible and *what* guarantees we can make about our models.

2.1 The Prophets Analogy

To understand the core components, imagine we are trying to predict the outcome of football matches.

- **Data Distribution (D):** The set of all matches ever played (past, present, and future). We cannot see this entire set; we only see a small sample.
- **A Hypothesis (h):** A single Prophet who looks at match details ($\mathbf{x}$) and predicts the score ($\mathbf{y}$).
- **Hypothesis Class ($\mathcal{H}$):** The specific set of prophets we decided to listen to (e.g., only prophets who use linear math or only prophets who look at team rankings).

2.2 Empirical Risk Minimization (ERM)

We cannot test our prophets on *every* match in history (D). We can only test them on a small sample S (the training set).

Definition 2.1: Empirical Risk Minimization

We choose the prophet (h_S) who made the fewest errors on our specific sample of matches S.

$$h_S = \arg\min_{h \in \mathcal{H}} \hat{R}_S(h)$$

The Problem: Ideally, we want the prophet with the lowest **True Risk** (error rate on the entire population D). However, we can only measure **Empirical Risk** (error rate on sample S).

2.3 Inductive Bias: The Need for Restrictions

Why can't we just include *every possible* prophet in our set $\mathcal{H}$?

- If $\mathcal{H}$ is too large (e.g., includes a prophet for every possible combination of outcomes), one prophet will predict our training data perfectly purely by **luck**.
- This prophet would have 0% Empirical Risk (perfect on training data) but high True Risk (fails on new data). This is **Overfitting**.

Theorem 2.1: Inductive Bias

To learn successfully, we must restrict our search to a specific, smaller class of models (Inductive Bias). We assume the truth lies within (or near) this smaller set.

2.4 Error Decomposition

The error of our chosen model $R(h_S)$ relative to the absolute best possible prophet in reality (f^*) comes from two sources:

$$\underbrace{R(h_S) - R(f^*)}_{\text{Total Excess Risk}} = \underbrace{(R(f_{\mathcal{H}}) - R(f^*))}_{\text{Approximation Error (Bias)}} + \underbrace{(R(h_S) - R(f_{\mathcal{H}}))}_{\text{Estimation Error (Variance)}}$$

1. **Approximation Error (Bias):** The penalty for choosing a restricted set of prophets. Maybe the best prophet in the world (f^*) wasn't in our set $\mathcal{H}$ to begin with. This error exists because our model is too simple.
2. **Estimation Error (Variance):** The penalty for having finite data. We might have chosen a prophet who got lucky on the sample S but is actually worse than the best prophet in our set ($f_{\mathcal{H}}$).

The Bias-Variance Tradeoff

- **Small Set of Prophets (Small $\mathcal{H}$):**
  - Low Estimation Error (easy to pick the winner with few samples).
  - High Bias (the winner might still be a bad predictor).
- **Large Set of Prophets (Large $\mathcal{H}$):**
  - Low Bias (good prophets are likely included).
  - High Estimation Error (hard to distinguish real skill from luck; requires massive data).

2.5 PAC Learning (Probably Approximately Correct)

PAC Learning answers the question: *How many matches (n) do we need to watch to be confident we picked a good prophet?*

Theorem 2.2: PAC Sample Complexity

For a finite hypothesis class of size $m = |\mathcal{H}|$, to guarantee that our chosen prophet has an error within ϵ of the best one, with probability at least $1 - \delta$, we need:

$$n \geq \frac{2 \log(2m/\delta)}{\epsilon}$$

3 Algorithms: Non-Parametric Methods

3.1 k-Nearest Neighbors (k-NN)

k-NN is a lazy learner—it doesn't learn a function $\mathbf{w}^T\mathbf{x}$. Instead, it memorizes the training data.

- **Logic:** If it looks like a duck (is near a duck), it's probably a duck.
- **Hyperparameter k:** Controls smoothness.
  - **Small k:** Captures noise (High Variance).
  - **Large k:** Smooths boundaries (High Bias).

3.2 Distance Metrics

The choice of distance defines what similar means. The common metrics are:

Definition 3.1: Distance Functions

Given two feature vectors $\mathbf{x}$ and $\mathbf{z}$:

- **Euclidean (L_2):** Straight line distance.

$$D_{L2}(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|_2 = \sqrt{\sum_{i=1}^{d}(x_i - z_i)^2}$$

- **Manhattan (L_1):** Grid/City Block distance.

$$D_{L1}(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|_1 = \sum_{i=1}^{d}|x_i - z_i|$$

- **Cosine Distance:** Angle-based distance (ignores magnitude).

$$D_{Cos}(\mathbf{x}, \mathbf{z}) = 1 - \frac{\mathbf{x}^T\mathbf{z}}{\|\mathbf{x}\|\|\mathbf{z}\|}$$

4 Algorithms: Unsupervised Learning

4.1 K-Means Clustering

K-Means partitions data into K groups by minimizing the variance within each cluster.

Definition 4.1: K-Means Objective

We want to find K centroids $\{\mathbf{c}_1, \dots, \mathbf{c}_K\}$ and assignments $S_1, \dots, S_K$ to minimize:

$$L(S_1, \dots, S_K) = \sum_{k=1}^{K} \sum_{\mathbf{x} \in S_k} \|\mathbf{x} - \mathbf{c}_k\|^2$$

Lloyd's Algorithm (Iterative Solution):

1. **Assignment Step:** Assign every point to the closest centroid.

$$S_k = \{\mathbf{x} : \|\mathbf{x} - \mathbf{c}_k\|^2 \leq \|\mathbf{x} - \mathbf{c}_j\|^2, \forall j\}$$

2. **Update Step:** Recompute the centroid as the mean of its assigned points.

$$\mathbf{c}_k = \frac{1}{|S_k|} \sum_{\mathbf{x} \in S_k} \mathbf{x}$$

3. **Repeat** until convergence.

5 Practical ML Workflow

To avoid overfitting, we must strictly separate our data.

Set	**Purpose**
Training	The textbook. The model learns from this data.
Validation	The practice exam. Used to tune hyperparameters (like k in k-NN).
Test	The final exam. Used **once** to report final accuracy. Never used for tuning.

Table 1: The Golden Rule of Data Splitting