Lecture 10: Supervised Learning: Regression and Classification

Yasuyuki Sawada, Yaolang Zhong

University of Tokyo
yaolang.zhong@e.u-tokyo.ac.jp

January 2, 2026

Lecture Overview

1. Overview of Machine Learning
   - The landscape: Supervised, Unsupervised, Reinforcement Learning
   - A unified notation ($\mathcal{D}$, f_θ, $\mathcal{L}$)
   - The three-step optimization framework
2. Supervised Learning
   - Regression: Linear models, Ridge & Lasso regularization, the Bias-Variance trade-off
   - Classification: Logistic regression, Confusion matrix, the Precision-Recall trade-off
3. Training Neural Networks
   - Automatic Differentiation (Backpropagation)
   - Stochastic Gradient Descent

The Landscape of Machine Learning

Machine Learning is the science of programming computers to learn from data. We generally categorize problems into three main paradigms based on the signal available to the learner:

- 1. **Supervised Learning**: Learn to predict output Y from input X.
  - **Regression**: The output Y is continuous (e.g., predicting house prices, temperature).
  - **Classification**: The output Y is categorical (e.g., spam detection, image recognition).
- 2. **Unsupervised Learning**: Discover structure within input X (no labels Y).
  - Clustering: Grouping similar data points together.
  - Dimensionality Reduction: Compressing data while preserving information.
- 3. **Reinforcement Learning**: Learn a policy to maximize rewards via interaction.
  - Agent-Environment Loop: The learner performs actions which affect future states.
  - Feedback: Learning relies on delayed reward signals rather than immediate correct answers.

A Unified Notation for All Tasks

Despite their differences, all three tasks share the same mathematical skeleton.

The Universal Components

1. **The Data** ($\mathcal{D}$):
   - Supervised: Pairs of $\{(x_i, y_i)\}$.
   - Unsupervised: Just features $\{x_i\}$.
   - RL: Trajectories of $\{(s_t, a_t, r_{t+1}, s_{t+1})\}$.
2. **The Model** (f_θ): The function we wish to learn, parameterized by θ.

$$\hat{y} = f_\theta(x) \quad \text{or} \quad \pi_\theta(a|s)$$

3. **The Objective** ($\mathcal{L}$): A Loss (or Reward) function measuring performance.

$$\min_\theta \mathcal{L}(f_\theta(\mathcal{D}))$$

The Three-Step Optimization Framework

- Step 1 (Data $\mathcal{D}$): Collect or sample the training data.
  - Supervised: pairs $\{(x_i, y_i)\}_{i=1}^{M}$ drawn from true distribution.
  - The quality and quantity of $\mathcal{D}$ fundamentally bounds what we can learn.
- Step 2 (Model f_θ): Choose a parameterized function class.
  - Linear: $f_\theta(x) = \theta^\top x$. Neural Network: $f_\theta(x) = \sigma(W_L \cdots \sigma(W_1 x))$.
  - The architecture determines the *hypothesis space* we search over.
- Step 3 (Objective $\mathcal{L}$ + Solver): Define loss and optimize.
  - Loss function ℓ: The function that measures how far the model's predictions are from the true values.
  - Define objective: $\min_\theta \mathcal{L}(\theta) = \frac{1}{M} \sum_{i=1}^{M} \ell\big(f_\theta(x_i), y_i\big)$
  - Solve via Gradient Descent: $\theta_{t+1} = \theta_t - \alpha \nabla_\theta \mathcal{L}(\theta)$

Supervised Learning: Prediction vs. Inference

The Setup: Given data $\mathcal{D}$, we estimate f such that $Y \approx f(X)$. The *objective* dictates the methodology:

- **1. Prediction (Focus on $\hat{Y}$)**
  - Goal: Minimize error on new inputs (Accuracy is king).
  - Trade-off: We accept complex "Black Box" models if they predict well.
  - Typical Use: Pattern recognition, Forecasting, Churn prediction.

- **2. Inference (Focus on $\hat{f}$ and $\hat{\theta}$)**
  - Goal: Interpret the *mechanism*. We need the functional form and parameter significance (e.g., Elasticity).
  - The Causal Challenge: Inference often implies seeking *Causality*. However, standard ML optimizes for *Correlation*, not Identification.
  - Crucial Caveat: Standard ML does not solve the Endogeneity Problem ($X \perp \epsilon$ is assumed). Identification strategies are required for causal claims but are beyond today's scope.

Task 1: Regression (Definitions)

Definition: Regression covers problems where the response variable Y is quantitative (numerical, continuous). Examples: Salary, Blood Pressure, House Price.

The Model: We assume the true relationship is:

$$Y = f(X) + \epsilon$$

where $f(X)$ is the systematic information provided by X, and ϵ is an irreducible error term (mean zero).

The Loss Function (Mean Squared Error): To train the model (estimate $\hat{f}$), we minimize the average squared difference:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{f}(x_i) \right)^2$$

Task 1: Regression (Example & Functional Form)

Problem: Predict *House Price* (Y) based on *Square Footage* (X_1) and *Number of Bedrooms* (X_2).

Functional Form: Linear Regression The simplest form for $f(X)$ is a linear combination of inputs:

$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

- Parameters (θ): The coefficients $\beta = (\beta_0, \beta_1, \beta_2)$.
- Interpretation: β_1 represents the average increase in Price for one additional unit of Square Footage, holding Bedrooms constant.
- Learning: We find the specific β values that minimize the Residual Sum of Squares (RSS).

Regression with Many Covariates: Motivation

Problem: When the number of covariates is large or regressors are highly correlated:

- Ordinary Least Squares (OLS) estimates can have high variance.
- The model may overfit the training data.
- Interpretation becomes unstable due to multicollinearity.

Solution: Regularization

We modify the regression objective by adding a penalty term that discourages large coefficients:

$$\text{Loss} = \text{Fit to data} + \text{Penalty on model complexity}.$$

This improves out-of-sample prediction by trading off bias and variance.

Regularized Regression: Ridge and Lasso

Consider the linear model: $y_i = \beta_0 + X_i'\beta + \varepsilon_i$.

Ridge Regression (L2 penalty):

$$\min_{\beta} \frac{1}{n}\sum_{i=1}^{n}(y_i - X_i'\beta)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

Shrinks coefficients toward zero; reduces variance, keeps all covariates.

Lasso Regression (L1 penalty):

$$\min_{\beta} \frac{1}{n}\sum_{i=1}^{n}(y_i - X_i'\beta)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

Performs shrinkage and variable selection; some coefficients set exactly to zero.

Intuition: How Ridge and Lasso Shrink Coefficients

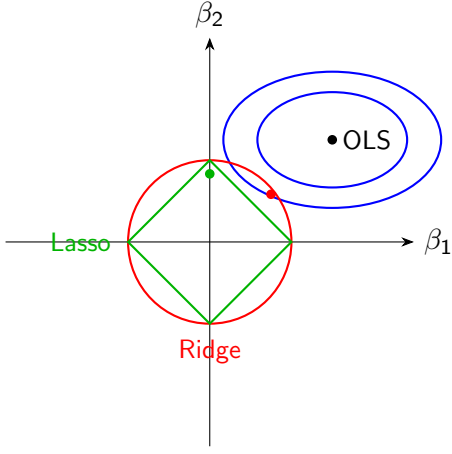
Consider the constrained form of the optimization:

Ridge (L2): $\min_\beta \text{RSS}(\beta)$ s.t. $\sum_j \beta_j^2 \leqslant c$ Lasso (L1): $\min_\beta \text{RSS}(\beta)$ s.t. $\sum_j |\beta_j| \leqslant c$

Key intuition:

- RSS has elliptical level sets centered at OLS solution.
- Ridge: circular constraint $\Rightarrow$ smooth shrinkage.
- Lasso: diamond constraint $\Rightarrow$ coefficients hit corners (sparse).

Shrinkage Intuition: Ridge vs Lasso



- Ridge shrinks coefficients smoothly toward zero.
- Lasso often sets coefficients exactly equal to zero.

The Bias-Variance Trade-off

One of the most important concepts in Supervised Learning (ISLP 2.2.2). It explains why we don't simply use the most complex model possible to fit the training data perfectly.

Decomposition of Expected Test Error: For a test point x_0 where $y_0 = f(x_0) + \epsilon$ with $E[\epsilon] = 0$, $\text{Var}(\epsilon) = \sigma^2$:

$$E\left[(y_0 - \hat{f}(x_0))^2\right] = E\left[(f(x_0) + \epsilon - \hat{f}(x_0))^2\right]$$

$$= \underbrace{E\left[(\hat{f}(x_0) - E[\hat{f}(x_0)])^2\right]}_{\text{Variance of } \hat{f}} + \underbrace{(E[\hat{f}(x_0)] - f(x_0))^2}_{\text{Bias}^2} + \underbrace{\sigma^2}_{\text{Noise}}$$

- **Variance**: Sensitivity to the training set. (High complexity → High Variance → Overfitting).
- **Bias**: Error from simplifying assumptions. (Low complexity → High Bias → Underfitting).
- **Irreducible Error**: The noise floor inherent in the data.

The Challenge: As we increase model complexity, Bias ↓ but Variance ↑. We must find the "sweet spot" (U-shaped curve) that minimizes the *sum* of both.

Task 2: Classification (Definitions)

Definition: Classification covers problems where the response Y is qualitative (categorical). The values of Y belong to a finite set of classes $\mathcal{C}$ (e.g., $\{\text{Spam, Ham}\}$ or $\{\text{Cat, Dog, Fox}\}$).

The Goal: We build a classifier $C(X)$ to assign a class label to a new observation. Often, we first estimate the conditional probability:

$$\hat{p}_k(x) = \Pr(Y = k \mid X = x)$$

We then classify to the class with the highest probability (Bayes Classifier).

The Loss Function (Error Rate): We minimize the fraction of misclassifications: $\frac{1}{n} \sum I(y_i \neq \hat{y}_i)$.

Task 2: Classification (Example & Functional Form)

Problem: Predict *Default Status* ($Y \in \{0, 1\}$) based on *Credit Balance* (X).

Functional Form: Logistic Regression We cannot use a straight line because probabilities must be between 0 and 1. Instead, we use the Logistic Function (Sigmoid):

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

- Mechanism: This S-shaped curve squashes any input value into the range $[0, 1]$.
- Decision Boundary: If $p(X) > 0.5$, we predict "Default" (1); otherwise "No Default" (0).
- Learning: We find β by maximizing the Likelihood of the observed data.

Classification Metrics: The Confusion Matrix

Why Accuracy Isn't Enough: In binary classification (e.g., disease detection), a naive model predicting "No Disease" for everyone achieves 99% accuracy if only 1% are sick—but it's useless.

The Confusion Matrix breaks down predictions into four outcomes:

	Actual Positive (1)	Actual Negative (0)
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

- **TP**: Correctly identified positives. **TN**: Correctly identified negatives.
- **FP** (Type I): False alarm. **FN** (Type II): Missed positive.
- **Precision** = TP/(TP+FP) — of predicted positives, how many correct?
- **Recall** = TP/(TP+FN) — of actual positives, how many detected?

The Precision–Recall Trade-off

Key idea: Changing the decision threshold τ trades off Precision and Recall.

- Lower threshold (predict positives more easily): Recall $\uparrow$, Precision $\downarrow$
- Higher threshold (predict positives more cautiously): Precision $\uparrow$, Recall $\downarrow$

Why this is different from regression:

- Regression uses a single metric (e.g., MSE).
- Classification needs multiple metrics because:
  - classes may be imbalanced,
  - FP and FN have different costs,
  - the optimal threshold depends on the application.

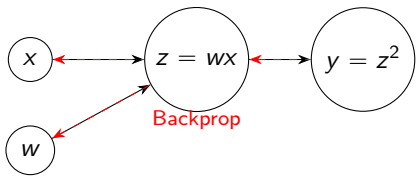
Training Neural Networks: Automatic Differentiation

- A neural network is a composition of functions: $\hat{y} = f(x; \theta)$, where θ collects weights and biases.
- To train, we need gradients $\nabla_\theta \mathcal{L}$. Manual differentiation is impractical for deep networks.
- **Automatic Differentiation** (Auto-diff):
  - Libraries (PyTorch, TensorFlow) record operations during the forward pass.
  - These form a **computational graph**; chain rule applied in reverse (**backpropagation**).
- Key takeaway: Auto-diff gives **exact gradients** with cost $\approx$ one forward pass.

Training Neural Networks: Automatic Differentiation

- Neural networks are built from many simple operations.
- During the forward pass, each operation is recorded.
- These operations form a computational graph.
- Auto-diff applies the chain rule automatically backward.

Key idea: Gradients are computed exactly, without manual differentiation.



Training Neural Networks: Stochastic Gradient Descent

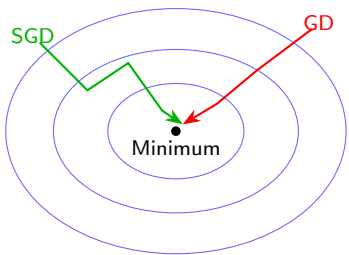
Objective: Minimize average loss: $\mathcal{L}(\theta) = \frac{1}{M} \sum_{i=1}^{M} \ell_i(\theta)$

- **Full Gradient Descent**: Computes gradients using all M observations. Accurate but expensive when M is large.
- **Stochastic (Mini-Batch) GD**:
  - Sample a small batch; form a noisy but cheap gradient estimate.
  - Update: $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{\text{batch}}$
- Why it works: Each update is fast; noise aids exploration; many small steps approximate full GD.

Training Neural Networks: Stochastic Gradient Descent

- Goal: minimize average loss over a large dataset.
- Full gradient descent:
  - Uses all data each step
  - Accurate but slow
- Stochastic gradient descent (SGD):
  - Uses small random batches
  - Fast but noisy updates

Key idea: Many cheap noisy steps can outperform a few expensive exact steps.



References

James, G., Witten, D., Hastie, T., Tibshirani, R. (2021).

An Introduction to Statistical Learning with Applications in Python.

Springer.

https://www.statlearning.com/

Previous Lecture Notes.

Optimization Frameworks for Neural Networks.