Introduction to Machine Learning
Lecture 2

Yonghao Lee

November 17, 2025

1 Recap: ERM and Learning Theory

This section reviews the core theoretical concepts of machine learning, focusing on the trade-off between model complexity and error.

1.1 The "Prophet" Analogy

In this lecture, a "prophet" is used as an analogy for a **hypothesis** or **model** (h). The set of all possible prophets we are considering is the **hypothesis class** (H).

1.2 The Bias-Estimation Tradeoff

When we select a hypothesis class H, there is a fundamental tradeoff:

- **Small Set (H):** If we have only a few prophets to choose from, our model class is simple.
  - **High Bias:** The best prophet in our small set might still be very inaccurate (high "true" error).
  - **Low Estimation Error:** Because the set is small, it's unlikely we'll be fooled by random chance on our data. The prophet we pick is likely to be the *truly* best one in the set.
- **Large Set (H):** If we have many prophets, our model class is complex.
  - **Low Bias:** With so many options, it's likely one of them is very accurate (low "true" error).
  - **High Estimation Error:** With so many prophets, it's very likely that one of them will get lucky and look good on our limited training data *just by chance*, even if it's not truly a good prophet. This is **overfitting**.

1.3 Inductive Bias

Inductive Bias is the set of assumptions we make to select a good hypothesis class H. The goal is to choose a class that is small (to control estimation error) but also contains accurate models (to ensure low bias).

1.4 Formal ML Terms

- **Sample:** A pair of features x and a true output y.
- **Data Distribution (D):** The underlying, true source of all possible (x, y) pairs. We only get to see a small random sample from D.
- **Hypothesis (h):** A model that makes a prediction: $y' = h(x)$.
- **Hypothesis Class (H):** The set of all possible hypotheses we are considering.

1.5 Empirical Risk Minimization (ERM)

ERM is the standard procedure for "learning".

1. We compute the **Empirical Risk** (L_{emp}) for every hypothesis h in our class H. This is just its error rate (or risk) on our n training samples. The formula for 0-1 loss (error percentage) is:
$$L_{emp}(h) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{h(x_i)\neq y_i}$$
where $\mathbf{1}$ is the indicator function (1 if $h(x_i)$ is wrong, 0 if it's right).
2. The ERM algorithm simply selects the hypothesis h_{ERM} that had the lowest empirical risk:
$$h_{ERM} = \arg\min_{h\in H} L_{emp}(h)$$

1.6 Formal Error Measures

- **True Error (Population Error):** The expected error of h on the *entire* data distribution D. This is what we care about but can never directly measure.
- **Bias (Approximation Error):** The *best possible true error* of any hypothesis *within* our class H. This is the error we are "stuck with" because of our limited choice of models.
- **Estimation Error:** The difference between our chosen model's true error and the bias. This is the "extra" error we got because we failed to pick the best h from H (e.g., we were fooled by our training data).
- **Generalization Gap:** The difference between a hypothesis's empirical error (on the training data) and its true error (on D).
- **Overfitting:** This happens when ERM selects a hypothesis h that has a large generalization gap (i.e., it looked great on the training data but is actually bad on new data).

1.7 PAC Learning

PAC (Probably Approximately Correct) learning gives us a guarantee on our learning procedure.

- A procedure is **PAC** if, with high probability $(1-\delta)$, the model it selects has an estimation error no larger than ϵ.
- **Sample Complexity:** For a finite hypothesis class H with size $m = |H|$, to be PAC, the number of training samples n must be at least:
$$n \geq \frac{2\log(2m/\delta)}{\epsilon}$$
This formula shows that as our hypothesis class gets bigger (larger m), we need more data n to avoid overfitting.

2 How Learning Works in Practice

We split our collected data into three sets:

1. **Training Set:** Used to learn the task. This is the data we use to run ERM and find the best hypothesis *within* each class (e.g., find the best kNN model, find the best linear model, etc.).
2. **Validation Set:** Used to select the best model. After training multiple models (e.g., kNN with $K = 1$, kNN with $K = 3$, a decision tree), we see which one performs best on the validation set. This helps us choose **hyper-parameters** (like K).
3. **Test Set:** Used to estimate the true error of our *final*, chosen model. This data must be "locked away" and only used once at the very end.

3 Nearest Neighbor Methods

This is one of the simplest and most intuitive ML methods.

3.1 1-Nearest Neighbor (1-NN)

The main idea is simple:

1. Collect a training dataset of examples (x_i) and their labels (y_i).
2. For a new target point x_{target}, find the *single* training example x_j that is most similar to it.
3. Use that example's label, y_j, as the prediction for x_{target}.

3.2 The "Elephant in the Room": Distance

How do we define "similar"?. The algorithm's success depends entirely on our choice of a **distance function**.

- **Core Idea:** We first define a function F that **represents** each data point x as a vector of numbers (features) $f = F(x)$.
- Then, we compute a geometric distance D between these feature vectors.
- $distance(x_1, x_2) = D(F(x_1), F(x_2))$.

3.3 Common Distance Functions

Given two feature vectors f_1 and f_2, common distances include:

- L_2 **(Euclidean):** The "straight-line" distance.

$$D(f_1, f_2) = \sqrt{\sum_s |f_1^s - f_2^s|^2}$$

- L_1 **(Manhattan):** The "grid" or "city-block" distance.

$$D(f_1, f_2) = \sum_s |f_1^s - f_2^s|$$

- L_0 **(Hamming Distance):** The number of features that are different.

$$D(f_1, f_2) = \sum_s \mathbf{1}_{f_1^s \neq f_2^s}$$

- L_∞ **(Chebyshev):** The single largest difference across any feature.

$$D(f_1, f_2) = \max_s |f_1^s - f_2^s|$$

- **Cosine Distance:** Measures the angle between two vectors. It is good at capturing similarity in *direction*, regardless of magnitude.

$$D(f_1, f_2) = 1 - \frac{f_1 \cdot f_2}{||f_1||_2 ||f_2||_2}$$

3.4 K-Nearest Neighbors (kNN)

The 1-NN algorithm is very sensitive to noise (e.g., one mislabeled training point).

- **Idea:** Instead of 1 neighbor, use the K nearest neighbors.
- **Algorithm:** Find the K training examples closest to the target. For classification, predict the **majority label** among these K neighbors.
- **Choosing K:** This is a hyper-parameter that involves a tradeoff.
  - **K small:** More sensitive to noise, but can capture fine-grained boundaries.
  - **K large:** More robust to noise, but can oversmooth boundaries.
- K is chosen by testing different values on the **validation set**.

3.5 Runtime of kNN

kNN is a "lazy" algorithm.

- **Training Time:** $O(1)$. There is no "training"; it just stores the dataset.
- **Prediction Time:** $O(N \cdot d)$. To classify *one* new sample, it must compute the distance to all N training samples, and each distance calculation takes $O(d)$ time (where d is the number of features). This can be very slow for large datasets.

4 Applications of Distance Methods

4.1 Retrieval

Goal: Given a query (e.g., an image, a text document), find the most similar items from a large database.

- **Algorithm:** Represent the query and all database items as vectors. Compute the distances and retrieve the K nearest examples (the K items with the smallest distance).
- **Examples:** Google Search, Amazon Recommendations.

4.2 Anomaly Detection

Goal: Discover rare, interesting, or unusual phenomena.

- **kNN Method:** An anomaly is a point that is "far away" from its neighbors.
- **Algorithm:** For a target point, compute the *average distance* to its K nearest neighbors. If this average distance is larger than some threshold, declare it an anomaly.

5 Clustering: K-Means

This is an **unsupervised learning** problem, meaning we have data x but **no labels** y. **Idea:** Group the data into K clusters, and use the cluster identity as a new label.

5.1 K-Means: The Objective

Goal: Group data into K clusters such that the **sum of squared distances** *within* all clusters is minimized.

- Each cluster k is represented by a "mean" vector (or centroid) c_k.
- We want to find the cluster assignments $S = (S_1, ..., S_K)$ and the means $c = (c_1, ..., c_K)$ that minimize the following loss function L:

$$L(S_1, ..., S_K) = \min_{S_1..S_K, c_1..c_K} \sum_{k=1}^{K} \sum_{x_i \in S_k} ||x_i - c_k||^2$$