CIS3333: Mathematics of Machine Learning Fall 2025

Lecture: Overview

Date: August 27, 2025 *Author: Surbhi Goel*

Acknowledgements: These notes are largely based on notes by Eric Wong from the Fall 2024 offering of this course.

Website: https://surbhi18.github.io/MoML/

Time & Location: Mondays & Wednesdays from 3:30-5:00PM in CHEM 109

Staff:

- **Instructor:** Surbhi Goel
- TA: Ezra Edelman
- TA: Guruprerna Shabhadi
- TA: Maya Gambhir

Office hours will be posted shortly on Ed Discussion.

Textbook: Mathematics for Machine Learning by Deisenroth, Faisal, and Ong.

Canvas: is only used to enter grades and serve as a portal for the rest of the course. For most information (syllabus, assignments, readings, extra notes) see the course website. Announcements will be posted on Ed Discussion, which you can access through Canvas. Homeworks are submitted through the Gradescope link, through Canvas.

1 What is Machine Learning?

From Tom Mitchell's classic textbook, Machine Learning is the study of algorithms that ''

- Improve their performance P
- At some task T
- With experience E

A classic example (spam detection).

- Task T: detecting spam from ham (not-spam)
- Performance measure P: percent of emails correctly classified as spam or ham
- Experience E: a database of emails labeled as either spam or ham

A contemporaneous example (ChatGPT).

- Task T: Generate responses in a conversation

- Performance measure P: High quality responses as measured by humans
- Experience E: dataset of prompts and demonstrations of responses

1.1 Components of an ML Algorithm

An ML algorithm has three parts: data, models, and learning.

Data. ML derives all of its power from experience, or data.

- While data can be many things (videos, images, text, radar, etc.), we typically abstract away the specifics as a dataset of N examples $x_1, \ldots, x_N \in \mathcal{X}$. A common choice is to work in the space of real-valued vectors $\mathcal{X} = \mathbb{R}^D$. For spam detection, x_i might be a vector in $\mathbb{R}^D$ where each dimension represents the count of a specific word in the email. A 100×100 grayscale image can be "flattened" into a vector $x_i \in \mathbb{R}^{10000}$.
- Sometimes, these are paired with labels $y_1, \ldots, y_N \in \mathcal{Y}$. A common choice is to work with real valued labels, so the whole dataset is $(x_1, y_1), \ldots, (x_N, y_N) \in \mathbb{R}^D \times \mathbb{R}$.
- Often, we compactly represent this as $X \in \mathbb{R}^{N \times D}$ (N examples, D features) and $Y \in \mathbb{R}^N$

$$X = \begin{bmatrix} x_1^T \\ \vdots \\ x_N^T \end{bmatrix} \text{ and } Y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

- The data that an ML algorithm has access to is typically referred to as *training data*.

Hypothesis class. Given a dataset, an ML algorithm then defines a hypothesis class, which is a set of possible functions (sometimes referred to as models).

- ML models are functions that, when given an input x, produces an output y.
- If the output is a real valued scalar, we write a predictor as $f : \mathbb{R}^D \to \mathbb{R}$
- Here, $f(x)$ applies the predictor f to an example x and returns a real number.
- We often write f_θ to denote that f has a set of parameters $\theta \in \Theta$ that can be varied to get different functions, and the hypothesis class is therefore $\mathcal{F} = \{f_\theta : \theta \in \Theta\}$
- Example: For predicting house prices from size, the hypothesis class could be all straight lines. This is an example of a *linear hypothesis class*, where models are of the form $f_\theta(x) = \theta^T x$.[1] The parameters $\theta \in \mathbb{R}^D$ define a specific line, and the full hypothesis class is the set of all possible lines, $\mathcal{F} = \{f_\theta : \theta \in \mathbb{R}^D\}$.

Learning. Given a dataset and a hypothesis class, the ML algorithm then tries to find the "best" function in the hypothesis class. This process is called *learning*.

- The goal of learning is to use the data to find a function $f_{\hat{\theta}}$ and its corresponding parameters $\hat{\theta}$ that *performs well on the data*.

[1] We typically assume that the input vector x is augmented with a constant feature, e.g., $x_0 = 1$, to account for the bias/intercept term.

- While "performing well" can be many things (i.e. accuracy, mean squared error), we typically abstract away the specifics as a *loss function* $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ that captures how "wrong" an output is. Then, performing well is equivalent to achieving low loss.
- For the hypothesis class of linear functions, the goal is to find a specific parameter $\hat{\theta}$ such that predictions on the data $\hat{y}_i = (\hat{\theta})^T x_i$ have low error, i.e.

  $$\ell(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2$$

  is low for all $i \in [N]$, where $[N] = \{1, \ldots, N\}$. This is the squared error, or ℓ_2-loss.
- The average loss on the training data is known as the *empirical risk*:

  $$\hat{R}(f) = \frac{1}{N} \sum_{i=1}^{N} \ell\big(f(x_i), y_i\big).$$

- The expected loss on new data is known as the *true risk*:

  $$R(f) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\big[\ell\big(f(x), y\big)\big],$$

  where $\mathcal{D}$ is the data distribution the training data is drawn from.

ML Algorithm. In total, an ML algorithm seeks to find the best function by minimizing the empirical risk, a strategy known as *Empirical Risk Minimization (ERM)*:

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \hat{R}(f)$$

This connects back to our original definition of machine learning:

- Improve their performance P, achieved by minimizing the total empirical risk $\hat{R}$ (learning)
- For some task T, tackled with a hypothesis class $\mathcal{F}$ (model)
- Using experience E, given by examples $\{(x_i, y_i)\}_{i\in[N]}$ (data)

1.2 Theoretical Questions for ML

To understand the theoretical underpinnings of ML, we explore three fundamental questions concerning *generalization*, *representation*, and *convergence*. This course is structured around proving one major theorem for each of these concepts.

We don't expect you to understand the details of these theorems yet; they are presented here to preview the goals of the course, and we will build the necessary mathematical tools to unpack them.

Generalization. The learning algorithm picks $\hat{f}$ by minimizing the *empirical risk* on the training data (X, Y). However, what we really want is for the model $\hat{f}$ to perform well on new data drawn from the same distribution, typically referred to as *testing data*. The question of generalization asks:

Why does minimizing the risk on the training data result in good performance on new, unseen testing data?

For example, why should my spam detector work on new spam emails? Why does ChatGPT work on new prompts that I make up?

The fundamental goal of machine learning is to find a model that performs well on new, unseen data. To formalize this, it helps to separate two quantities: the *empirical risk*, $\hat{R}(f)$, which is the average loss on the training data we have, and the *true risk*, $R(f)$, which is the expected loss on all possible data from the underlying distribution. We can only calculate and minimize the empirical risk, but we really want to find a model with low true risk.

A potential pitfall is *overfitting*: our learning algorithm might find a function $\hat{f}$ that fits the random quirks of our specific training sample perfectly (achieving low empirical risk), but fails to capture the true underlying pattern (resulting in high true risk). To ensure our model generalizes, we need a formal way to relate the empirical risk to the true risk.

The first module of this course will cover the fundamentals of probability, culminating in proving the following formal generalization bound, which provides exactly such a guarantee.

Theorem 1 (Generalization of Finite Hypothesis Classes)**.** *If the hypothesis class is finite*[2] *(i.e.* $|\mathcal{F}| < \infty$*) and the loss is bounded* $0 \leq \ell \leq 1$*, then with probability at least* $1 - \delta$ *(over the draw of the* N *training examples), the following holds for all* $f \in \mathcal{F}$*:*

$$\left|R(f) - \hat{R}(f)\right| \lesssim \sqrt{\frac{\log\left(|\mathcal{F}|\right) + \log(1/\delta)}{N}}.$$

The $\lesssim$ notation means that the left-hand side is upper bounded by the right-hand side up to a constant factor.

This theorem provides a formal guarantee that what we learn is meaningful. The bound on the right-hand side has two key dependencies:

- It decreases as the number of data points N increases (at a rate of $1/\sqrt{N}$). More data leads to better generalization.
- It increases with the size of the hypothesis class $|\mathcal{F}|$. A more complex hypothesis class is harder to generalize and requires more data.

This guarantee is also stronger than we asked for; it holds *simultaneously for all* predictors $f \in \mathcal{F}$, not just the $\hat{f}$ we found.

Representation and Tractability. The ML algorithm will search for a $\hat{f}$ from the hypothesis class $\mathcal{F}$ that minimizes the empirical risk. But we may not know if there exists a function in the hypothesis class that is a "good" solution in the first place. This leads to the fundamental question of representation:

[2] A finite hypothesis class means we are only choosing between a finite number of functions.

What kinds of functions can a hypothesis class represent or approximate?

A simple linear model, for instance, can't capture complex, non-linear patterns. One way to make it more expressive is to map the data into a higher-dimensional feature space and then apply a linear model there. For instance, we could augment a feature x with x^2 and x^3 to learn polynomial functions. What if we took this to the extreme and mapped our data to an *infinite-dimensional* feature space? A linear model in such a space would be incredibly powerful, potentially able to represent any function.

This creates a new, seemingly impossible challenge: how can we possibly represent/learn a model that has an infinite number of parameters? This is a question of *tractability*.

The second module of this course culminates in the classic Representer Theorem, which provides an elegant solution to this tractability problem. The key insight is to use mathematical tools called *kernels*[3]. Kernels allow us to work in infinite-dimensional feature spaces implicitly, and the theorem shows that for regularized empirical risk minimization, the optimal solution still has a surprisingly simple, finite form that depends only on the training data.

Theorem 2 (Representer Theorem)**.** *Fix a positive definite kernel k with reproducing kernel Hilbert space*[4] *$\mathcal{H}$. The minimizer for any objective of the form*

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \hat{R}(f) + \Omega(\|f\|_{\mathcal{H}}^2)$$

where $\hat{R}(f)$ is the empirical risk and Ω is a non-decreasing regularizer, can be expressed as

$$\hat{f}(x) = \sum_{i=1}^{N} \alpha_i k(x_i, x)$$

In other words, the representer theorem says that the minimizer over the infinite-dimensional hypothesis class $\mathcal{H}$ can be represented as a finite sum of kernels centered on the training data.

Convergence. The first theorem answered when models generalize to new data (and how much data we need to do so). The second theorem answered what functions we can hope to model in the first place. However, these results all ignore the problem of learning the model from the data—solving the minimization problem. Convergence aims to answer the following question:

How long does it take to learn a model from data?

Different techniques for solving the minimization problem as well as different hypothesis classes can drastically alter how long the learning process takes.

A common method for minimizing the empirical risk is *gradient descent*. It is an iterative algorithm that starts with an initial guess for the parameters θ and repeatedly takes a small step in the direction of the steepest descent of the loss function—that is, the negative of the gradient. Think of it as trying to find the bottom of a valley by always taking a step downhill.

[3] A function that measures similarity between points in a way that guarantees certain nice mathematical properties.

[4] An infinite-dimensional space of functions where we can still have notions of distance similar to Euclidean space.

The third module of this course will cover fundamentals of calculus, culminating in the following convergence rate for gradient descent. This result is crucial because it tells us how quickly an algorithm can learn, which determines how long we must train a model to achieve a desired accuracy.

Theorem 3 (Convergence Rate of Gradient Descent)**.** *Suppose we run gradient descent on a convex*[5]*, smooth*[6] *function $\hat{R}$ with fixed constant learning rate η depending on the smoothness of the function. Then, for all time t, we have*

$$\hat{R}(\theta_{t+1}) - \hat{R}(\theta^\star) \lesssim \frac{\|\theta_1 - \theta^\star\|_2^2}{2t}$$

where $\theta^\star$ is the global minimum.

In other words, if the empirical risk satisfies the assumptions, then running the gradient descent algorithm is guaranteed to converge at a rate of $O\left(\frac{1}{t}\right)$, i.e. in order to get ϵ error, we must run $O(1/\epsilon)$ steps of gradient descent. This is in contrast to other approaches such as Stochastic Gradient Descent (SGD)[7]. For general convex functions, SGD converges at a slower rate of $O\left(\frac{1}{\sqrt{t}}\right)$, requiring more steps to achieve the same accuracy. However, SGD is often preferred for large datasets because each individual step is computationally much cheaper than a full gradient descent step.

[5] A convex function is 'bowl-shaped' and has the crucial property that any local minimum is also a global minimum.

[6] A function is smooth if its gradient does not change too quickly.

[7] SGD is a variant of gradient descent that updates the parameters using the gradient calculated from only a single, randomly chosen data point (or a small "mini-batch" of points) at each step.