CIS3333: Mathematics of Machine Learning Fall 2024

Lecture: Overview

Date: August 28, 2024 *Author: Eric Wong*

Website. `https://www.cis.upenn.edu/~exwong/moml/`

Time & Location. Mondays & Wednesdays from 12-1:30PM in CHEM B13

Staff & Office Hours.

- Instructor: Eric Wong, Monday & Wednesday after class or by request
- TA: Sima Noorani, Tuesdays from 3-4:30PM
- TA: Avishree Khare, Thursday from 2-3:30PM
- TA: Helen Jin, Friday from 3:30-5PM

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

Canvas. 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 at `https://www.cis.upenn.edu/~exwong/moml/`. Announcements will be posted on Ed Discussion, 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 emails labeled as either spam or ham

A contemporous 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 $\mathcal{X} = \mathbb{R}^D$.
- 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 $X = \begin{bmatrix} x_1^T \\ \vdots \\ x_N^T \end{bmatrix}$ 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: $f_\theta(x) = \theta^T x$ is a linear predictor with parameters $\theta \in \mathbb{R}^D$. The set of linear functions $\mathcal{F} = \{f_\theta : \theta \in \mathbb{R}^D\}$ defines a possible hypothesis class.

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_{θ^*} and its corresponding parameters θ^* that *performs well on the data.*

- 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 θ^* such that predictions on the data $\hat{y}_i = (\theta^*)^T x_i$ have low error, i.e.

  $$\ell(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2 \tag{1}$$

  is low for all $i \in [N]$.
- The total loss over all of the data is known as the *empirical risk*:

  $$R_{\text{emp}}(f, X, Y) = \sum_{i=1}^{N} \ell(f(x_i), y_i) \tag{2}$$

ML Algorithm. In total, an ML algorithm does the following:

$$f^* = \arg\min_{f \in \mathcal{F}} R_{\text{emp}}(f, X, Y) \tag{3}$$

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

1.2 Theoretical Questions for ML

Generalization. The ML algorithm will find a function f that minimizes the risk on the dataset (X, Y). This is often referred to as having low risk on the *training data*. However, what we really want is for the model f to perform well on new data, typically referred to as *testing data*. The question of generalization asks the following:

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? Theorems proving the answer to this question are often referred to as *generalization bounds*. Specifically, these aim to show the following result:

$$\mathbb{P}\underbrace{\left(R_{\text{emp}}(f, X, Y) - R_{\text{true}}(f_{\text{opt}}) < \epsilon\right)}_{\text{good event}} > \underbrace{1 - \delta}_{\text{high probability}} \tag{4}$$

Here, a "good event" is when the performance of our model on our dataset (X, Y) matches the best theoretically possible performance. This is formalized as the empirical risk being ϵ-close to the true risk (for now, you can think of the true risk as the empirical risk estimated with an infinite number of examples). When this happens, we say our model f *generalizes* to new data. Generalization bounds aim to show that generalization happens with high probability.

The first module of this course will cover the fundamentals of probability, culminating in proving the following generalization bound known as uniform convergence:

Theorem 1 (Generalization of Finite Hypothesis Classes)**.** *If the hypothesis class is finite (i.e.* $|\mathcal{F}| < \infty$*) and the loss is bounded* $\ell \leq B$*, then we have*

$$\mathbb{P}\left(R_{emp}(f, X, Y) - R_{true}(f_{opt}) < B\sqrt{\frac{2\log(2|\mathcal{F}|) + 2\log\delta^{-1}}{n}}\right) > 1 - \delta. \tag{5}$$

Representation/approximation. The ML algorithm will search for a 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. For example, consider trying fit a linear model to data sampled from a quadratic function. No matter how hard you search the space of linear functions and use an infinite amount of data, you will never achieve small error! The next class of theorems in ML ask the following:

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

For example, the linear function class can only represent linear functions and not quadratic functions. On the other hand, the neural network hypothesis have what is known as a universal approximation theorem—they can approximate any function to high accuracy. These kinds of results characterize the type of functions that we can hope to learn with a given hypothesis class.

The second module of this course will cover fundamentals of linear algebra, culminating in proving the classic representer theorem, which shows that a class of models called kernel SVMs can represent optimal minimizers over complex, infinite-dimensional function spaces:

Theorem 2 (Representer Theorem)**.** *Fix a kernel* k*, and let* $\mathcal{H}$ *be the corresponding RKHS (a hypothesis class). Then, for an empirical risk function* $R_{emp} : \mathcal{H} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ *and non-decreasing* $\Omega : \mathbb{R} \to \mathbb{R}$*, the minimizer for the empirical risk*

$$f^* = \arg\min_{f\in\mathcal{H}} R_{emp}(f, X, Y) + \Omega(\|f\|^2_{\mathcal{H}} \tag{6}$$

can be expressed as

$$f^*(x) = \sum_{i=1}^{N} \alpha_i k(x_i, x) \tag{7}$$

In other words, the representer theorem says that all minimizers over the infinite hypothesis class known as RKHS can be represented with a finite sum of kernels (i.e. a kernel SVM).

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. For example, linear models often have a ℓ_2

regularizer term because certain techniques can provably solve this problem much faster with the term. On the other hand, other hypothesis such as deep learning are known to have much slower convergence rates, and hence take much longer to train.

The third module of this course will cover fundamentals of calculus, culminating in proving convergence rates for learning algorithms used to minimize the empirical risk. Specifically, we will prove classic convergence rate for gradient descent:

Theorem 3 (Convergence Rate of Gradient Descent)**.** *Suppose we run gradient descent on an L-smooth function R with fixed constant learning rate $\eta = 1/L$. Then, for all time t, we have*

$$R(\theta_{t+1}) - R(\theta^\star) \leq \frac{L\|\theta_1 - \theta^\star\|_2^2}{2t} \tag{8}$$

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, which converge at a rate of $O\left(\frac{1}{\sqrt{t}}\right)$ and are known to be much harder and require many more steps.