CIS5200: Machine Learning Spring 2023

Lecture 1: Introduction and Overview

Date: January 12, 2023 *Author: Surbhi Goel*

Acknowledgements. These notes are heavily inspired by lecture notes from CIS5200 - Spring 2022 and Cornell University's CS 4/5780 — Spring 2022.

Disclaimer. There may be minor typos or errors in the notes. If you notice any, please reach out to the author.

1 Course Overview and Logistics

Check out the course website (`https://machine-learning-upenn.github.io/`) for all information regarding schedule, office hours, course policies, material, etc. Also refer to the slides for a more digestible version of the policies.

Homework 0 will be out on January 12, 2023, and is due on Friday, January 20, 2023. It will count for 2% of the grade, and you will get full credit if you attempt all questions irrespective of whether the answer is correct or wrong.

2 What is Machine Learning?

The goal of machine learning is to design and analyze algorithms that improve performance on an underlying task given experience pertaining to the task. A standard machine learning pipeline looks as follows:

Figure 1: A typical machine learning pipeline.

The experience here refers to the available **training data**, the knowledge consists of a predictive (or decision-making) **model**, and the performance is evaluated based on some underlying **measure or criterion**. In order to specify a machine learning task, we need to specify all of the above: data, model, and performance measure.

Given the task, the **learning algorithm** takes the training data as input, and produces the model as output. In many cases, the learning algorithm is some form of optimization procedure. Different

algorithms lead to different performance measures, and generally stronger performance is preferred. There are however several other considerations and trade-offs, such as robustness, interpretability, computational and statistical efficiency of the model. We will touch upon some of these throughout the course. It is often useful to put your devil's advocate hat on, and question the limitations of different learning algorithms.

3 Types of Machine Learning Tasks

Here are some core types of machine learning tasks.

- *Supervised learning*: Training data consists of instance and label pairs. The goal is to predict well on future instances, for e.g. regression, classification.
- *Unsupervised learning*: Training data consists of only instances, no labels. The goal is to learn some patterns and structures in the data, for e.g. clustering, anomaly detection.
- *Semi-supervised learning*: Training data largely consists of unlabelled data, with access to a small amount of labelled data. The goal is to exploit both unlabelled and labelled data for better performance. This sits in between supervised and unsupervised learning.
- *Online learning*: Training data is received in an online fashion, that is, a single instance at a time. Here the learning algorithm is expected to make a prediction at each timestep prior to receiving the label.
- *Active learning*: This is a form of supervised learning where the learning algorithm gets to choose which instances it wants labels for with usually a cost associated with each labelling request. This is generally useful in settings where labelling cost is high.
- *Reinforcement learning*: Unlike the above methods, no data is given to the learning algorithm (or agent), There is a set of states that an agent can be in, a set of actions that can be taken in each state which transition the agent to another state, and a 'reward' signal based on the state it transitions into. The goal is to learn a model (or policy), that determines which action the agent should take in each state, such that it maximizes the overall reward that is collected by following the policy from a fixed start state.

4 Supervised Learning

Let us dive into the first paradigm, supervised learning. As mentioned above, the learner receives training data consisting of instances and the corresponding labels and the goal is to learn a model that predicts the label correctly on new instances that the learner has not seen before. A few common examples include image classification, spam detection, stock price prediction, etc.

Setup In a typical supervised learning problem, there is an instance space $\mathcal{X}$ containing (descriptions of) instances or objects for which predictions are to be made, a label space $\mathcal{Y}$ from which labels are drawn. The training data consists of a finite number of labeled examples $\mathcal{S} =$

$((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$. The goal is to learn from these examples a predictor $f : \mathcal{X} \to \mathcal{Y}$ that given input x outputs a prediction $y = f(x)$.

Typically the instance space is assumed to be a subset of $\mathbb{R}^d$, that is, the instances are d-dimensional vectors. For instance, in the case of image classification, these can be viewed as the pixel values of all the pixels in the image. There are several types of supervised learning problems which govern the choice of $\mathcal{Y}$. In the setting of binary classification, $\mathcal{Y} = \{0, 1\}$, and the goal is to predict the correct class for the input instance. For example, consider the problem of spam detection in your emails. This is a binary classification task with the classes being spam or not-spam. Another common setting is regression, $\mathcal{Y} \subseteq \mathbb{R}$, and the goal is to predict the real-valued label associated with the instance. The stock prediction example we saw in class is an instance of this.

Hypothesis class Now in order to find a predictor, we need to specify the class of functions we are searching over. This set of functions is referred to as the hypothesis class and we will denote it by $\mathcal{F}$[1]. Usually the choice of the hypothesis class encodes some inductive bias or prior belief about the task at hand. For example, we will see when we study neural networks later in the class, convolutional networks encode translational invariance which is a useful inductive bias for images. The No Free Lunch Theorem tells us that we need to make some assumptions in order to design a successful algorithm, and that no one algorithm can do well on all tasks. Therefore, encoding task-specific assumptions is necessary.

Loss function Finally, we need to evaluate the performance of a predictor. This is usually done using a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ where the first argument is the prediction and the second argument is the true label. The loss function captures how good or bad the prediction is. For instance, in case of binary classification we typically use 0/1 loss which is basically equivalent to counting all the mistakes the algorithm makes. For regression we typically use the squared loss, which penalizes based on how far the prediction is from the truth.

$$\ell_{0/1}(\hat{y}, y) = \begin{cases} 0 & \text{if } \hat{y} = y \\ 1 & \text{otherwise.} \end{cases} \qquad \ell_{\mathsf{sq}}(\hat{y}, y) = (\hat{y} - y)^2.$$

Evaluating the loss function over a training set gives us a way to compare different predictors. Lower loss generally (not always) means a better predictor.

Empirical Risk Minimization Given a loss function and a hypothesis class, we can train/learn a good predictor by minimizing the average loss over the training set $\mathcal{S}$. This optimization is known as Empirical Risk Minimization (ERM).[2]

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \underbrace{\frac{1}{m} \sum_{i=1}^{m} \ell(f(x_i), y_i)}_{\text{Empirical Risk } \hat{R}(f)} .$$

See Figure 2 for a pictorial representation of this pipeline.

[1] It is often denoted by $\mathcal{H}$.

[2] Note that this is not the only way to solve the problem but is a commonly used approach.

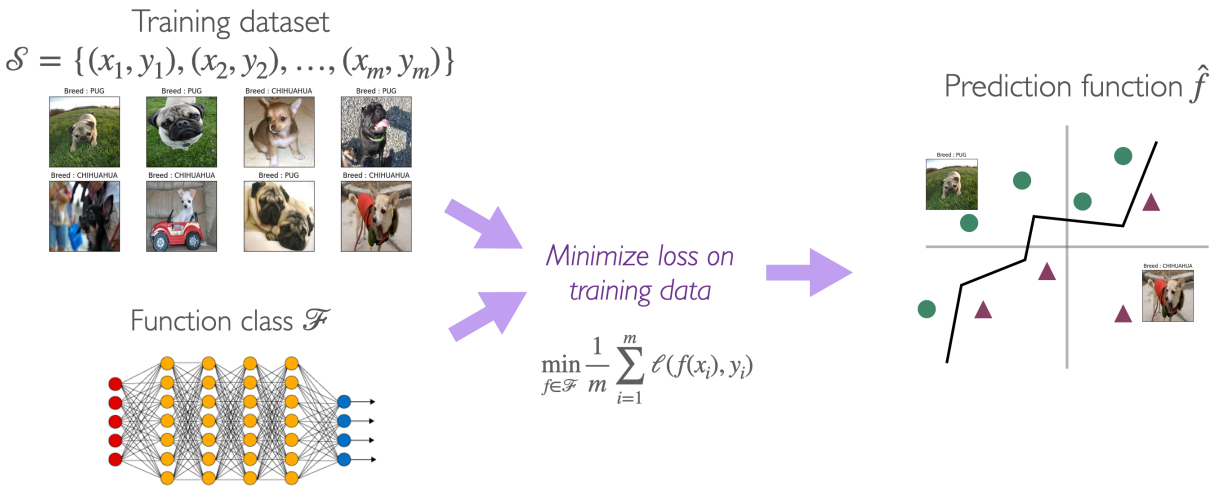

Figure 2: Putting it together: the supervised learning pipeline.

Generalization Let us ask ourselves whether doing well on training data suffices for the notion of learning we have in mind. Consider a rather silly predictor that memorizes the entire training set, that is, outputs the correct value, but predicts 0 everywhere else. This silly predictor gets 0 training error or empirical risk, however, we do not expect this to do well in practice. This is because we expect the learned predictor to perform well on unseen data that it may encounter.

In order to capture this, a commonly made assumption is the i.i.d. assumption, where we assume that the data (both training and future) is generated independently and identically from some underlying (unknown) distribution $\mathcal{D}$. Then we can define the risk of a predictor (generalization error) $\hat{f}$ on unseen examples, as the expected loss of a data point drawn from $\mathcal{D}$,

$$\underbrace{R(f)}_{\text{True Risk}} = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\ell(f(x), y)\right].$$

Observe the following tautology,

$$R(\hat{f}) = \underbrace{(R(\hat{f}) - \hat{R}(\hat{f}))}_{\text{generalization gap}} + \hat{R}(\hat{f}).$$

Our optimization minimized the latter, and we need to ensure that the former (generalization gap) is also small in order to get small generalization error.

In practice, we divide the given training data into training and test set, train on the training set and use the test set to evaluate the generalization gap. The usual split is 80/20. Given a sufficiently large test set, this should be a good enough measure of the true risk of the predictor.

Later in the learning theory section of the course, we will formalize a notion of complexity of a hypothesis class in order to quantify this gap.

Failures Let us think about how our learning pipeline may fail.

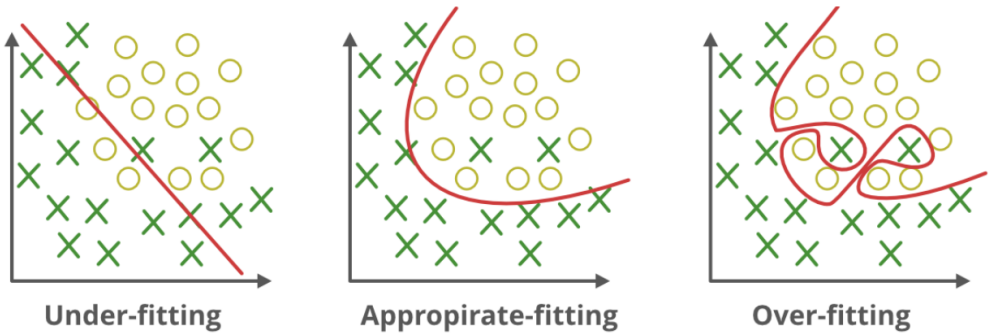

Figure 3: Effect of hypothesis class on the fit to the data. Image sourced from `https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/`.

- *Overfitting*: We could end up minimizing the training loss but find that the test loss is large. Think back to the memorization example, where we observed this as well. Here we say we have overfit to the training data.
- *Underfitting*: We might not be able to make the training loss small which would in turn make the test loss large. This could happen if our chosen function class is not expressive enough to fit our data.

See Figure 3 for an example of these. This highlights a trade-off (known as the bias-variance tradeoff) based on the complexity of the underlying hypothesis class. We will explore this in more detail in subsequent lectures.