Machine Learning

Muchang Bahng

Spring 2024

Contents

I've spent a good amount of time trying to create a map of machine learning. Obviously, it's a big field, with relations to probability, statistics, classical algorithms, deep learning, reinforcement learning, supervised vs unsupervised. It was quite overwhelming to cover, but I think the following structure is best. These notes concern the field of classical machine learning, focusing on supervised and unsupervised algorithms. Reinforcement learning will not be covered here, and deep learning will be covered in a separate set of notes since a lot of space is needed to talk about recent developments and architectures (e.g. RCNN, YOLO, LSTMs, Transformers, VAEs, GANs, etc.). We will focus more on establishing the theoretical foundations of most learning algorithms and analyze interpretable algorithms.

1. Prerequisites: I assume that the reader knows measure theory and some degree of functional analysis. Programming in Python is also helpful for implementation. Basic knowledge of statistics at the level of confidence intervals and hypothesis testing is necessary.
2. I introduce learning schemes using statistical learning theory to establish the big ideas of machine learning (model selection, bias complexity tradeoff) and its theoretical limitations. This introduces concepts like empirical risk minimization, concentration of measure, VC dimension.
3. I then present a list of discriminative algorithms (GLMs, SVMs, decision trees, boosting, clustering, dimensionality reduction and analyze them using the theoretical tools while talking about their implementation.
4. I move onto generative algorithms.

That's it.

1 Statistical Learning Framework

In the most general setting of supervised learning, we start off with a general probability space $(\Omega, \mathcal{F}, \mathbb{P})$. We construct two measurable functions $X : \Omega \to \mathcal{X}$ and $Y : \Omega \to \mathcal{Y}$, which induces probability measures on their codomains, constructing the probability spaces $(\mathcal{X}, \mathcal{F}_\mathcal{X}, \mathbb{P}_X)$ and $(\mathcal{Y}, \mathcal{F}_\mathcal{Y}, \mathbb{P}_Y)$. By sampling N elements from Ω, we essentially sample from the joint probability space $\mathcal{X} \times \mathcal{Y}$, giving us our **data**

$$\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$$

The $\mathbf{x}^{(i)}$'s are called the **covariates** and the $\mathbf{y}^{(i)}$'s are called the **labels** or **targets**. As notation, vectors will be bolded for clarification. In applications, we assume further structures on these spaces, the most important being dimension. Both $\mathcal{X}$ and $\mathcal{Y}$ are usually constructed as the Cartesian product of 1-dimensional continuous subsets of $\mathbb{R}$ (continuous features) or a discrete set (discrete features).

1.1 Empirical Risk Minimization

1.2 PAC Learning

1.3 Uniform Convergence

1.4 Bias Complexity Tradeoff

1.5 VC Dimension

1.6 Nonuniform Convergence

1.7 Rademacher Complexity

2 Linear Models

3 Overview

We will say that the kth feature lives in the space F_k, and so $\mathcal{X}$ is simply

$$\mathcal{X} = \prod_{k=1}^{K} F_k$$

We call each of these features the **explanatory variables**, and $\dim(\mathcal{X}) = K$.
The distance metric is also something that is assumed to exist. In subsets of $\mathbb{R}$, this is already inherent, but what about categorical sets like $\{\text{truck}, \text{car}, \text{plane}\}$? We would like to embed them in some space that does have a distance, and since there is no inherent order between these sets, the distances have to be pairwise equal. This is where **one-hot encoding** comes in, which takes each of the k categories and maps them to the kth unit vector.

$$\texttt{OneHot}(\text{truck}) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \ \texttt{OneHot}(\text{car}) = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \ \texttt{OneHot}(\text{plane}) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix},$$

We have one-hot encoded these categorical variables in $\mathbb{R}^3$, but now we have a multidimensional feature when we said before that features should be 1-dimensional! Therefore, given k categories that have been one-hot encoded into $\mathbb{R}^k$, we can simply construct k binary features. Looking at the example above, we can split them into

1. Whether a child likes a car, which lives in $\{0, 1\}$.
2. Whether a child likes a truck, which lives in $\{0, 1\}$.
3. Whether a child likes a plane, which lives in $\{0, 1\}$.

There are many other examples of features that may be multidimensional. For example, word embeddings in NLP models are represented as 300-dimensional vectors, but for analytical purposes, we can just think of every word as containing 300 different features.
All this trouble of reducing our features to 1-dimensional numerical values that satisfy the consistent properties of distance and order pays off in the end. This allows us to embed our covariates in $\mathcal{X} = \mathbb{R}^D$, where D represents the dimension of all our features after they have been one-hot encoded. So, we have data of the form

$$\begin{aligned} \mathbf{x}^{(1)} &= \left(x_1^{(1)}, x_2^{(1)}, x_3^{(1)}, \dots, x_D^{(1)}\right) \\ \mathbf{x}^{(2)} &= \left(x_1^{(2)}, x_2^{(2)}, x_3^{(2)}, \dots, x_D^{(2)}\right) \\ \dots &= \dots \\ \mathbf{x}^{(N)} &= \left(x_1^{(N)}, x_2^{(N)}, x_3^{(N)}, \dots, x_D^{(N)}\right) \end{aligned}$$

If we just stack the $\mathbf{x}^{(n)}$'s on top of each other, then we get a $N \times D$ **design matrix** $\mathbf{\Phi}$.

3.1 Types of Models

Supervised learning basically boils down to constructing some **hypothesis function** $h : \mathcal{X} \to \mathcal{Y}$ that best predicts what label $\hat{y}$ we should give a new input $\hat{x}$. In practice, we usually deal with **probabilistic models** where rather than giving a point estimate $\hat{y}$, we attempt to model the *distribution* $\mathbb{P}_{Y|X=\hat{x}}$. Even though in the end, we will just output the mean μ of this conditional distribution, modeling the distribution allows us to quantify uncertainty in our measurements.
If Y is a discrete random variable, then we can perhaps model the conditional distribution as a discrete distribution, which is known as a **classification** problem, and if Y is continuous, then we have a **regression** problem. But just like how discrete and continuous distributions are two sides of the same coin of probability measures, both regression and classification are contained within the umbrella of conditional expectation.
There are two ways to model $\mathbb{P}_{Y|X=x}$.

Definition 3.1. Discriminative models attempt to do this directly by modeling only the conditional probability distribution $\mathbb{P}_{Y|X=x}$. We don't care about the underlying distribution of X, but rather we just want to try and predict Y given X. Essentially, we are trying to approximate the conditional expectation $h(X) = \mathbb{E}[Y \mid X]$, which is the best we can do. Given $X = x$, we use our model of $\mathbb{P}_{Y|X=x}$, and our hypothesis function will predict the its mean.

$$h(x) = \mathbb{E}[Y \mid X = x]$$

Definition 3.2. Generative models approximate this conditional probability by taking a more general approach. They attempt to model the joint probability distribution $\mathbb{P}_{X\times Y}$ (also called **inference**), which essentially gives everything we need about the data. Doing this allows us to *generate* more data (hence the name), which may be useful.
One way to approximate the joint distribution is to model the conditional distribution $\mathbb{P}_{X|Y=y}$, which gives the distribution of each labels. Now combined with the probability measure $\mathbb{P}_Y$, we can get the joint distribution. Usually in classification, the $\mathbb{P}_Y$ is easy to approximate (the MLE is simply the fequencies of the labels), so conventionally, modeling $\mathbb{P}_{X\times Y}$ and modeling $\mathbb{P}_{X|Y=y}$ are considered the same thing. Once we have these, we can calculate the joint distribution, but in high-dimensional spaces this tends to be computationally hard. Therefore, we usually resort to simply calculating $\mathbb{P}_{X|Y=y}$ and then using Bayes rule to approximate

$$\mathbb{P}_{Y|X} = \frac{\mathbb{P}_{X|Y}\mathbb{P}_Y}{\mathbb{P}_X}$$

where the normalizing term is computed using Monte Carlo simulations.

Since $\mathbb{E}[Y \mid X] = g(X)$ can be any $\mathcal{F}_Y$-measurable function, there could be crazy functions out there that we might have to predict. This space may even be uncountably infinite dimensional and so we must try to fit an uncountable number of parameters. This is clearly too big, so we shrink our function space to something that is finite and more manageable.

Definition 3.3 (Parametric Models)**.** A **parametric model** is a set of functions $\mathcal{M}_{\boldsymbol{\theta}}$ that can be parameterized by a finite-dimensional vector. The elements of this model are hypotheses functions $h_{\boldsymbol{\theta}}$, with the subscript used to emphasize that its parameters are $\boldsymbol{\theta}$. We have the flexibility to choose any form of h that we want, and that is ultimately a model assumption that we are making.

Example 3.1. Here are some examples:

1. If we assume $h : \mathbb{R}^D \to \mathbb{R}$ to be linear, then h lives in the dual of $\mathbb{R}^D$, which we know to be D-dimensional.
2. If we assume h to be affine, then this just adds one more dimension.
3. If we assume $h : \mathbb{R} \to \mathbb{R}$ to be a kth degree polynomial, then g can be parameterized by a $k+1$ dimensional θ.

However, parametric models may be limited in the way that we are assuming some form about the data. For certain forms of data, where we may have domain knowledge, it is reasonable to use parametric models, but there are cases when we will have absolutely no idea what the underlying distribution is. For example, think of classifying a $3 \times N \times N$ image as a cat or a dog. There is some underlying distribution in the space $[255]^{3N^2} \times \{\text{cat}, \text{dog}\}$, but we have absolutely no idea how to parameterize this. Should it be a linear model or something else? This is when nonparametric models come in. They are not restricted by the assumptions concerning the nature of the population from which the sample is drawn.

Definition 3.4 (Nonparametric Models)**. Nonparametric models** are ones that cannot be expressed in a finite set of parameters. They may be countably or uncountably infinite.

Let us formalize this concept of a model in a definition.

Definition 3.5 (Model)**.** A **model** $\mathcal{M}$ is a certain form of a function/distribution that we want to fit. In supervised learning, it is a predefined subspace contained within the function space of $\mathcal{F}_X$-measurable functions.

An extremely important measure of a model is its complexity.

Definition 3.6 (Model Complexity)**.** The **model complexity** of a parametric model $\mathcal{M}$ is a monotonically increasing function of its number of free parameters.

$$\Omega(\mathcal{M})$$

For example, given that θ is a m-vector of model $\mathcal{M}$, its model complexity can be just the number of its free parameters $\Omega(\mathcal{M}) = m$. However, we can weigh these parameters differently, as it doesn't need to increase linearly.

At this point, the reader may wonder which model may be the best one to choose? This will be discussed in the model selection section, but for now, we must be able to quantify how well a model performs.