CHAPTER

1 Introduction

Machine learning began as an offshoot of the field of artificial intelligence, built around the idea that an intelligent system should improve with experience, over time. However, it has since found numerous applications in systems for which "general" artificial intelligence is not necessary, including recommendation systems, targeted advertising, text summarization, anomaly detection, and much more.

Machine learning is, at its core, a data science: how to organize, understand, and make inferences from a finite set of observations. It is thus very closely tied to the field of statistics, and often builds on statistical analysis techniques. However, traditional statistics is often focused on what conclusions can be drawn from a given set of observations with high confidence (for example, the unbiquitous p-value in science); machine learning is often more focused on good empirical predictive performance. Machine learning is also closely tied to applied mathematics, particularly optimization theory, and to algorithmic analysis in computer science, with both statistical and computational efficiency being first-order concerns.

Machine learning is often applied to problems whose solutions are difficult to describe in explicit, algorithmic ways. For example, tasks such as face detection (finding all the faces in an image) are very useful in practice – the locations of faces can be used to autofocus a camera, or as a preprocessor to recognizing individuals – and humans are very good at this task, but quantifying how we unconciously identify faces so easily is very difficult. Instead, it is relatively easy to show the computer what we mean by a face, and ask it to learn by example.

Types of machine learning

There are several general types of machine learning problems, with different characteristics.

Supervised learning focuses on trying to learn a predictive relationship between an observation, or feature vector x, and a "target" y. These are situations where we can obtain the "best answer" for our examples, and would like to teach the computer program to reproduce these answers. More formally, given a set of m examples, i.e., example observations $x^{(i)}$ and the desired output $y^{(i)}$, we try to find a function $f(x)$ that accurately reflects the relationship between x and y in the examples. When our prediction $f(x)$ is real-valued, this is often termed regression. In contrast, when our prediction is required to be categorical (discrete), it is called classification; classification problems often correspond to decisions or actions (such as flagging an email as being spam).

Unsupervised learning does not have a notion of a specific target to predict, but rather tries to understand the underlying structure of the observations $x^{(i)}$. Often, the goal is to find a simpler way of

summarizing the data, for example, identifying which data points are most similar, or organizing them into groups of related data points. Sometimes, this can be used to understand what values of the data are typical, either to fill in a partially missing measurement (called imputation), or to identify data points that are unusual (called anomaly detection). Two common techniques in unsupervised learning are clustering, in which the data are understood by grouping them together, and dimensionality reduction, in which we look for a "simplified" representation of high-dimensional data that, for example, tries to preserve topological properties (which points are close to each other) while embedding them in only two or three dimensions for visualization.

The related area of semi-supervised learning uses unlabeled data to understand the underlying structure of the data, and leverages this information to try to do a better job at some supervised learning task. This can be helpful if we have only a few labeled examples or they are expensive to obtain, but easy access to large volumes of unlabeled data.

Reinforcement learning refers to learning problems in which we do not have direct supervision about what the correct action or prediction y is, but we can get indirect feedback about its quality. Reinforcement learning is often used for training a model to make sequences of actions over time, for example a robot or game-playing AI, but is also useful in a number of simpler scenarios where the best action is not known apriori, such as online advertising.

Part I

Supervised Learning

CHAPTER

2 Supervised Learning: Basics

In supervised learning, our goal is to learn a prediction function $f(x)$, which takes in a set of observed features x and outputs a prediction of a target, y. When our prediction $f(x)$ is allowed to be real-valued, this is called regression; when our prediction is categorical (discrete), it is called classification. While some techniques are specific to either classification or regression, most supervised learning methods translate readily between the two tasks with only minor differences, and we will typically present them together.

In general, supervised learning methods work by defining a model, or learner, $f(x;\theta)$, which is a flexible class of prediction functions defined by some parameters θ; by changing the values of θ we can induce different input/output behavior for f.

If we consider the set of all possible parameter settings θ, we can trace out the set of all possible predictors that our model can produce, called the hypothesis class.

As an example, suppose we are designing a simple spam filter for our email. For each email, indexed by i, we observe a set of characteristics or features $x^{(i)}$ that we will use to make our prediction of whether that email should be labelled as spam. For the moment, let us suppose that we observe two binary features, $x = [x_1, x_2]$, where $x_1 \in \{0, 1\}$ indicates whether the email sender is in our contacts list, and $x_2 \in \{0, 1\}$ indicates whether the email sender is at our institution. Because the values of x are easy to enumerate, we can express any function $f(x)$ as a table.

Notation: We typically imagine that the data are drawn from some unknown joint distribution $p(X, Y)$, over random variables X and Y; we will use captial letters to indicate the random variables, and lowercase letters x, y to indicate a value of X or Y, respectively.

A key element of learning is the *performance measure*, which captures how we plan to evaluate the performance of our learner. Typically, this takes the form of a loss function, which measures how well we perform on some data set D and which we will generically denote $J(f, D)$.

In the case of regression problems, in which our predictions $f(X;\theta)$ are real-valued, it is typical to score our predictions based on how close they are to the correct answer Y. For example, the most common loss function in practice is the *expected squared error*,

$$J_{\text{SE}}(f) = \mathbb{E}_{p(X,Y)}(Y - f(X))^2$$

In the case of classification, where both the target Y and the prediction $f(\cdot)$ are discrete valued, it often makes more sense to ask simply whether our prediction is correct. In this case, the most common loss

function is the *classification error rate*, which measures the probability of making an incorrect prediction:

$$J_{01}(f) = \Pr[Y \neq f(X)] \quad = \quad \mathbb{E}_{p(X,Y)} \mathbb{1}[Y \neq f(X)]$$

where $\mathbb{1}[Y \neq f(X)]$ indicates the Iverson bracket function, which is one if the associated logical statement is true, and zero otherwise. For this reason, the error rate is sometimes called the *zero-one* loss.

Optimal prediction

In the case that the distribution $p(X, Y)$ is explicitly known, it is possible to derive the optimal predictors for many loss functions. While this is difficult to apply in practice, due to the fact that the probability distributions of real phenomena are typically unknown, it serves as a useful grounding for later discussion.

For regression, suppose that we wish to minimize the expected squared error loss over all possible functions $f(x)$. Because $f(x)$ is unrestricted, it suffices to consider each value of x individually; for convenience, denote this value by $f_x = f(x)$. Then, the optimal prediction f_x can be obtained by minimizing the loss function,

$$J(f_x) = \mathbb{E}_{p(Y|X=x)}(Y - f_x)^2 = \int (Y - f_x)^2 \, p(Y|X = x) \, dY$$

over f_x. It is easy to verify that the expected squared error has a single extremum, which is a minimum.[1] We can find its value by taking the derivative and solving, i.e,

$$\frac{\partial}{\partial f_x} J(f_x) = \int 2 \, (Y - f_x) \, p(Y|X = x) \, dY \quad = 0$$

$$\Rightarrow \qquad \int 2 \, Y \, p(Y|X = x) \; = \; \int 2 \, p(Y|X = x) dY \; f_x$$

and noting that $\int p(Y|X = x)dY = 1$ and that $\int Y \, p(Y|X = x)dY = \mathbb{E}[Y|X = x]$, we see that the optimal squared-error estimator is $f^*(x) = \mathbb{E}[Y|X = x]$, the conditional expectation of Y given X.

For the classification error rate, the optimal estimator is similarly easy to characterize. Again, considering arbitrary functions $f(x)$ means that it suffices to find the optimal estimate for some (arbitrary) observed value x. Given $X = x$, the probability of any particular target value y is the conditional probability, $p(Y = y|X = x)$. Thus, to have the highest probability of being correcti (and thus the lowest probability of error), we should pick the most probable value y,

$$f^*(x) = \arg\max_y p(Y = y|X = x).$$

By definition, this will classify examples with that value of Y correctly, and all others incorrectly, giving overall error rate

$$J_{01}(f^*) = \mathbb{E}_X \big[\, 1 - \max_y p(Y = y|X = x) \, \big].$$

In this case, the optimal predictor f^* is called the Bayes optimal predictor, and its error rate $J_{01}(f^*)$ is called the Bayes optimal error rate.

[1] The squared error is a convex function.

Example: spam filtering, two binary features

Joint probability

x	$p(Y=0\|x)$	$p(Y=1\|x)$
00	0.08	0.30
01	0.12	0.04
10	0.17	0.03
11	0.25	0.01

$\Rightarrow$

Conditional probability

x	$p(Y=0\|x)$	$p(Y=1\|x)$
00	0.21	0.79
01	0.75	0.25
10	0.85	0.15
11	0.96	0.04

$\Rightarrow$

Prediction

x	$f^*(x)$
00	1
01	0
10	0
11	0

So, the Bayes optimal predictor (having the lowest possible error rate) on this problem is to predict spam for any email not in our contacts or institution, but keep all others; the error rate (probablity of error) for this policy is $0.08 + .04 + .03 + .01 = 0.16$.

Note that the Bayes optimal error rate is the theoretically best possible probability of error using the feature observation x, but that different features may have more or less information, giving different optimal error rates. For example, if we had additional information about the email, such as its content, we will be able to do a better job of discrminating between spam and real email.

Empirical loss functions

Of course, we do not typically have access to the true probability distribution $P(X, Y)$; instead, we have only a data set of examples, called the training set:

$$D = \{x^{(i)}, y^{(i)} \ : \ i = 1 \dots m\}$$

consisting of m feature vectors $x^{(i)}$ and their associated target values $y^{(i)}$. We can use the training examples to estimate the expected loss, using the empirical loss, for example the mean squared error,

$$J_{\text{MSE}}(f, D) = \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - f(x^{(i)}))^2$$

which measures the average error over the data D, or the empirical error rate,

$$J_{01}(f, D) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}[y^{(i)} \neq f(x^{(i)})],$$

which measures number of examples in D that are misclassified by f.

Continuing our trivial spam example, it is easy to see the minimizer of the empirical error rate. Suppose we have $m = 30$ training examples, which we group by their values of X and Y to fill in a table:

Data

x	#$Y=0$	#$Y=1$
00	2	7
01	3	3
10	4	2
11	9	0

$\Rightarrow$

Empirical conditional

x	$\hat{p}(Y=0\|x)$	$\hat{p}(Y=1\|x)$
00	0.22	0.78
01	0.50	0.50
10	0.67	0.33
11	1.0	0.0

$\Rightarrow$

Prediction

x	$\hat{f}(x)$
00	1
01	0
10	0
11	0

Given a sufficient amount of data, our estimated probabilities will be close to the true probabilities, and thus so will our decision function.

However, already we can see a potential issue – with our limited data set, we never observed any spam examples with $X = [11]$, making our empirical estimate of the probability of this outcome zero. This can become a serious problem as the number of possible outcomes of X increase. Suppose, for example, that to improve our prediction we decide to gather more information about each email. Given more information, we should be able to make a better prediction about whether the email is worth reading. In practice, we can easily gather very large numbers of potentially useful features, in the hundreds if not thousands or even millions. For example, the content of the email is obviously informative: certain words, like "homework" or "grade", tend to indicate non-spam, while others ("lottery", "winner") often indicate spam. However, we quickly arrive at a problem: if we observe n binary features, $x = [x_1, \ldots, x_n]$, our table of probabilities has 2^{n+1} entries. Such a table quickly becomes impractical to store in memory (we would fill several terabytes with only $n = 40$ features). Worse, as we have more possible values of X, the probability of any particular outcome decreases, leading to more and more outcomes for which we will observe few or no data, and have extremely poor estimates of their associated probability.

Given all the features we might decide to measure, it is also very likely that a new example x will not be exactly the same as any previously observed example. This illustrates the importance of *inductive bias* in a learner: the ability to infer a prediction on unseen data points based on similar but not identical examples. Our brains do this all the time; when we recognize an image of a chair, it's not because we've seen that picture of a chair before, but rather (perhaps) because it has features (four legs, a seat, a back) that are sufficiently similar to other chairs we've seen.

A simple regression example also illustrates the importance of inductive bias. Suppose we are predicting the relationship between a real-valued single (scalar) feature X and a real-valued target Y. **See example from slides.** With continuously many x, we need to be able to predict values in between our observations. The only way to do this is by assuming something about the underlying function, such as smoothness. The type of model we assume will dictate what form our predictor will make, and thus both how closely it can capture the true relationship between X and Y, and also how easily the model can be estimated from a given number of observed data.

Naïve Bayes models

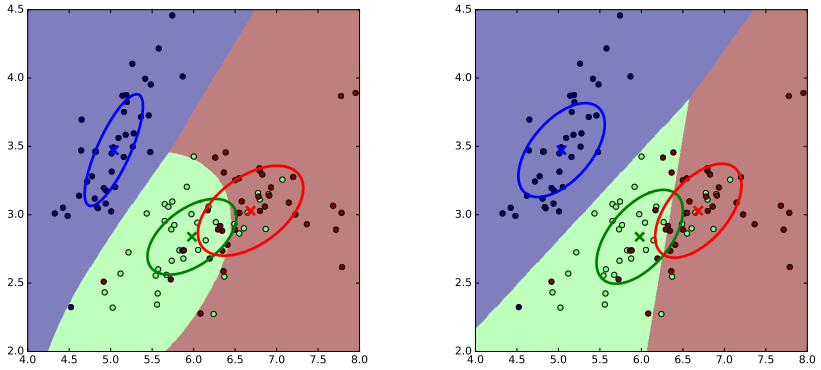
What can we do if we have a large number of features, then? One option is to use assumptions about the distribution to simplify our model, and make it easier to estimate with our available data size. A classic example is the *naïve Bayes* model.

First, we will re-express the probability distribution of X and Y in a more convenient form. Since in our classification problem, we are assuming that Y is discrete, with only a few classes, let us apply the chain rule to write the joint probability as $p(X, Y) = p(Y)\, p(X|Y)$. Here, $p(Y)$ is the probability of each class (before observing any data), and $p(X|Y = y)$, for each possible y, are the class-conditional probabilities: the distributions of the features that would be observed with data from each class. Then, to evaluate the conditional or posterior probability, we can apply Bayes rule:

$$p(Y|X) = \frac{p(X|Y)\, p(Y)}{p(X)} \tag{2.1}$$

$$= \frac{p(X|Y)\, p(Y)}{\sum_c p(Y = c) p(X|Y = c)}, \tag{2.2}$$

where the last expression uses the law of total probability to re-express $p(X)$ in terms of $p(Y)$ and $p(X|Y)$.



(a) (b)

Figure 2.1: Bayes classifiers using Gaussian class conditional distributions $p(X|Y = y)$ on the Fisher Iris data. (a) Estimated Gaussian distributions (ellipses) for each class, along with the model's prediction of the most likely class at each point. (b) Estimated Gaussian models when the three classes share a covariance model (`equal=True`).

Now, a Naïve Bayes classifier applies a simple assumption about $p(X|Y)$: that the features are conditionally independent of one another given the class value, i.e.,

$$p(X|Y) = p(X_1|Y) \cdot p(X_2|Y) \cdot \ldots \cdot p(X_n|Y) = \prod_i p(X_i|Y)$$

This assumption drastically reduces the number of parameters required to estimate the probability distribution. Suppose that Y has c possible values (classes), and we have n features X_i each with d possible values. Then, an arbitrary $p(X, Y)$ has $c \cdot d^n$ probabilities, with no constraints except the fact that they must be positive and sum to one; hence, $c \cdot d^n - 1$ "free" parameters (parameters that are not fully determined given the others). In contrast, each $p(X_i|Y = y)$ has only $d - 1$ free parameters, meaning that a conditionally independent $p(X, Y)$ can be expressed using only $c + c \cdot n(d - 1)$ free parameters (for $p(Y)$ plus each class conditional distribution).

Example **tbd**

Gaussian Bayes models

A nice property of the reformulation in (2.1) is that it can also be applied to give Bayes classifiers when the features X are not discrete. While we could simply discretize X, we may need a large number of bins, particularly if X has many features. Instead, we can assume some convenient form for the class-conditional models $p(X|Y = y)$, for example, a Gaussian model. We can then simply estimate the parameters of that model by fitting it to the data points with class y, and then apply (2.1) to evaluate which class Y is most probable given a new observation X.

add notes; form of decision boundary for Gaussian Bayes classifiers, and special cases?

2.1 Overfitting the training data

fill in overfitting: fits the observed data better, but probably learns the underlying concept less well.