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 ubiquitous 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 efficiency (how many data are required to estimate something accurately) and computational efficiency (how much time and memory are required to compute something) 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. In contrast, 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

1 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 f that our model can produce, called the **hypothesis class**. Learning, then, can be viewed as selecting a particular predictor (characterized by its parameter settings) from this hypothesis class. In the sequel, we shall see a number of popular types of learner (linear models, neural networks, decision trees, etc.), and the types of functions f they produce.

Example 1-1 : Spam Filtering

Suppose we are designing a simple spam filter for our email. For each email, we observe a set of characteristics, or **features**, x 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.

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*,

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

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:

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

where $\mathbb{1}[Y \neq f(X)]$ indicates the Iverson bracket function, which evalutes to 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,

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

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,

$$\begin{aligned} \frac{\partial}{\partial f_x} J(f_x) = \int 2\,(Y - f_x)\, p(Y|X = x)\, dY \quad &= \quad 0 \\ \Rightarrow \qquad \int 2\,Y\, p(Y|X = x)\, dY \;&=\; \int 2\, p(Y|X = x) dY\; f_x \end{aligned}$$

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 estimator in the sense of minimizing the expected squared error is

$$\textbf{Squared-error optimal predictor:} \qquad f^*(x) = \mathbb{E}[Y|X = x],$$

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 correct (and thus the lowest probability of error), we should pick the most probable value y,

$$\textbf{Error rate optimal predictor:} \qquad f^*(x) = \arg\max_y p(Y = y|X = x).$$

[1] The squared error is a convex function.

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.

Example 1-2 : Spam Filtering and Optimal Prediction

Returning to our spam filtering example, suppose that we know the true probability distribution of our features x (known address and same institution) and target y (spam or not). Then, the minimum error rate decision is to simply predict the more probable class for each possible configuration of x. Taking a specific joint probability distribution as an example, we can compute the most likey value of y for each x, and thus the optimal prediction f^*:

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. We can also easily compute the error rate (probablity of error) for this policy using the joint probability values in the table: $\Pr[Y \neq f^*(X)] = 0.08 + 0.04 + 0.03 + 0.01 = 0.16$.

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 should 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:

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

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**,

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

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.

Generally speaking, machine learning models **fit** a model to the data by identifying a a function $f(\cdot)$ that has low empirical loss. An important sub-class of machine learning methods, called *probabilistic* models, work by fitting a probability distribution $\hat{p}$ to the observed data D, and then use $\hat{p}$ to determine the prediction function f. *Discriminative* probabilistic models estimate the conditional probability $p(Y|X)$, while *generative* probabilistic models estimate the joint probability $p(X, Y)$. Although the latter is often more difficult to estimate, since $p(X, Y) = p(Y|X)p(X)$ contains information about both the predictive relationship from X to Y, and also about the distribution of the features X, a generative model can be useful for understanding whether new data points are similar to the data on which the model was trained (since if not, our model may be unreliable!), or to be applicable to data in which some of the feature values X_j have not been observed (missing values). We will see examples of both probabilistic and non-probabilistic learning models.

Example 1-3 : Spam Filtering from Data

Returning to our trivial spam example, it is easy to compute the minimizer of the empirical error rate. Suppose we have $m = 30$ training examples (which are drawn from the joint distribution in Example 1-2); we group these data 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

We can compute the empirical error rate of this predictor, which is $J_{01}(f, D) = (2+3+2+0)/30 \approx .23$. Given a sufficient amount of data, our estimated probabilities will be close to the true probabilities, and thus so will our decision function; for example, in this case we see that our predicted function is actually optimal, $\hat{f} = f^*$, even though our empirical probabilities are only noisy estimates of the true probabilities.

In this case, the model parameters θ correspond to the probabilities, e.g., $\theta = [\hat{p}(Y = 1|X = 00) \ldots \hat{p}(Y = 1|X = 11)]$; these parameter values are a part of the definition of the learner $f(x; \theta)$, and by changing them we can alter the input/output behavior of f. A value $\hat{\theta}$ of the parameters is then selected, or "fit" to the training set D (here by computing the empirical conditional probabilities), producing a particular function $\hat{f}(x) = f(x; \hat{\theta})$ that we expect will predict Y well on the data in D. Our spam filtering model is an example of a **probabilistic learner**, in which we not only estimate Y but also our uncertainty in our prediction (here characterized by $\hat{p}(Y|x)$). Even many classifiers that do not explicitly estimate probabilities can output a relative degree of confidence, sometimes called a "soft" prediction, to characterize about which examples the model is most or least sure.

Inductive Bias and Overfitting

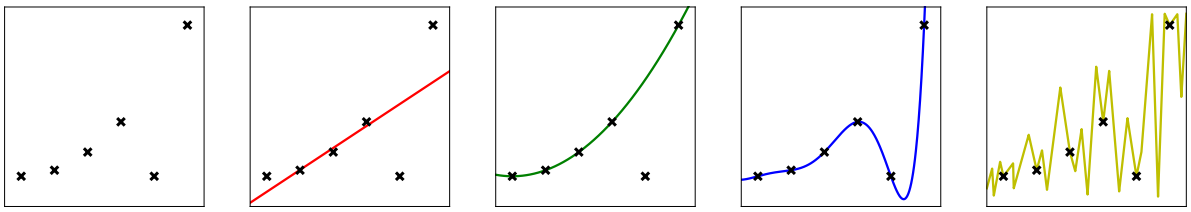
Already in our spam filtering example however, we can see a potential issue – with our limited data set, we never observed any spam examples with $X = [11]$, making our empirical estimate of the probability of this outcome zero. This can become a serious problem as the number of possible outcomes of 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 – with only $n = 40$ features we would already fill several terabytes of memory. 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. For such outcomes, we will be left with extremely poor estimates of their 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.

Inductive bias is necessary to learning. This is easy to see in a regression setting: let X be a real-valued (scalar) feature, and Y a real-valued target. Our dataset consists of a finite number of observations $D = \{(x^{(i)}, y^{(i)})\}$. But even if we assume that Y is noiseless, i.e., that observing X allows us to perfectly predict Y, there are still an infinite number of possible functions $f(x)$ that pass through the points in D – essentially, f could do *anything* in between the observed points. Which of these functions are more likely? Smooth functions? Polynomials? Functions consisting of several linear segments? Add to this all the functional relationships that could hold if we also assume Y can be noisy: all the functions that pass "close to" the points in D. Inductive bias is what lets us decide which of these explanations we prefer.

Example 1-4 : Inductive bias illustration

To illustrate this point, consider a simple regression example, whose data are plotted in panel (a). Each subsequent panel (b)-(e) shows a curve representing a valid, if more or less plausible, explanation of the data points:



(a) (b) (c) (d) (e)

(a) An example data set. (b) A linear relationship between X and Y, with noise. (c) A quadratic relationship, which fits most of the points better and leaves just one poorly-explained, "outlier" data point. (d) A smooth, noiseless function. (e) A complex, non-smooth noiseless function.

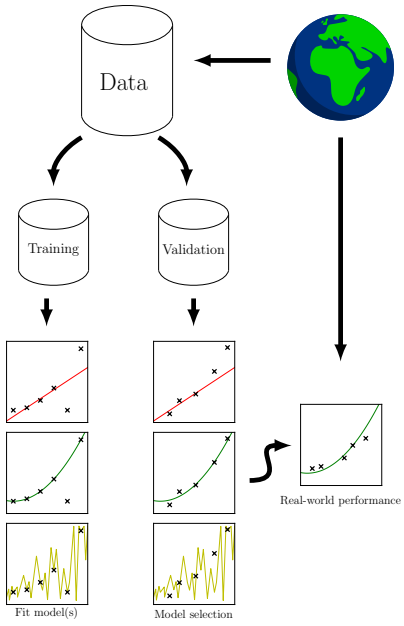
In other words, Y could have a simple relationship to X, with any inconsistencies being explained by noise, or a more complex relationship that exactly hits the observed values. Again, each model explains the observed data in a valid way, but suggests very different predictions for new values. Which models we prefer indicates the inductive bias in our human minds – our ability to go beyond the observations based on our sense of what types of relationships are more or less plausible, as informed by our past experience with functions and data.

Overfitting

Example 1-4 also hints at another key point – that functions which fit the observed data better, such as panels (d) and (e), are not always the best predictors. The function in panel (e), for example, correctly reproduces the values of $y^{(i)}$ for each $x^{(i)}$ in D, and thus has zero *empirical* mean squared error, $J_{MSE}(f, D)$. However, it does not seem likely to be the best explanation of the true relationship between X and Y, i.e., the minimum of the *expected* mean squared error, $J_{SE}(f)$.

In some sense, the disconnect arises from the fact that we know that $\hat{f}$ has been selected, or fit, *using* the data D, and thus performing well on D is no longer sufficient to convince us that $\hat{f}$ is a good predictor. As an analogy, a student might demonstrate that they know a subject well by passing an exam; but the feat becomes less convincing if they are allowed to study beforehand using the same exam and its correct answers. Instead, we give students "similar" practice questions, but test them on new problems they have not seen, in order to evaluate whether they have learned the underlying concepts correctly and not simply memorized the practice answers. Analagously to a class exam, in machine learning we have the concept of a **test data set** and empirical **test loss**, which provide an estimate of the expected test error by evaluating the model's performance on a collection of examples that have been withheld from the training process, i.e., were not made available when selecting $\hat{f}$.

In practice, we may also have one or more additional data sets, called **validation data**, which are also used for model assessment. Again following our exam analogy, a student might elect to test herself using practice exams, in order to decide whether she had studied sufficiently. In machine learning, we can use validation data to assess our models (deciding when to stop training, or to select among several different types of models, such as those in Example 1-4), in order to estimate how we will perform at test time.



The difference between validation data and test data is really one of scope. If we try enough different models on our validation data, we will eventually find one that seems to do well – but does it actually predict accurately, or have we just gotten lucky on those particular questions? Therefore, we typically withold evaluating on the test data until all stages of selecting and fitting the models are finished. Again, in our analogy, a student does not get to see the test until she has completely finished her studying process.

Many machine learning competitions, such as those on Kaggle [kag], use multiple partitions of data for training and evaluation. They may provide a large set of training data, which the user may then partition further into training and evaluation; then, some data may be reserved for ongoing feedback to

the participants (leaderboard placement, etc.), but the final model evaluation will be done, and a winner chose, using data whose performance has never been reported to the participants, making it difficult for them to simply search for a "lucky" model.

Naïve Bayes models

Returning to our spam classification problem in Example 1-1, what can we do if we have a large number of features? 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:

$$\begin{aligned} p(Y|X) &= \frac{p(X|Y)\, p(Y)}{p(X)} \\ &= \frac{p(X|Y)\, p(Y)}{\sum_y p(Y = y) p(X|Y = y)}, \end{aligned} \tag{1.1}$$

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

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.,

$$p(X|Y) = p(X_1|Y) \cdot p(X_2|Y) \cdot \ldots \cdot p(X_n|Y) = \prod_j p(X_j|Y)$$

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_j each having d possible values. Then, an arbitrary joint distribution $p(X, Y)$ consists of $c \cdot d^n$ 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 $p(X_j|Y = y)$ has only $d - 1$ free parameters, meaning that a conditionally independent $p(X, Y)$ can be expressed using only $(c-1)+c \cdot n(d-1)$ free parameters (for $p(Y)$ plus each class conditional distribution).

Example 1-5 : Naïve Bayes

Consider the data set shown on the right. We have seven data points, each with three binary-valued features and a binary class Y. For Naïve Bayes, we estimate the following empirical probabilities:

X_1	X_2	X_3	Y
0	0	0	0
1	1	0	0
0	1	1	0
0	0	1	1
1	0	0	1
1	1	0	1
1	1	1	1

$$\begin{aligned} p(Y = 0) &= 3/7 & p(Y = 1) &= 1 - P(Y = 0) = 4/7 \\ p(X_1 = 1|Y = 0) &= 1/3 & p(X_1 = 1|Y = 1) &= 3/4 \\ p(X_2 = 1|Y = 0) &= 2/3 & p(X_2 = 1|Y = 1) &= 2/4 = 1/2 \\ p(X_3 = 1|Y = 0) &= 1/3 & p(X_3 = 1|Y = 1) &= 2/4 = 1/2 \end{aligned}$$

Applying Bayes rule as in (1.1), we can compute the probability of, say, class $Y = 1$ given an observation $X = [1, 0, 1]$ as,

$$\begin{aligned} p(Y=1|X=101) &= \frac{p(Y=1)\,p(X_1=1|Y=1)\,p(X_2=0|Y=1)\,p(X_3=1|Y=1)}{p(Y=1)\,p(X_1=1|Y=1)\,p(X_2=0|Y=1)\,p(X_3=1|Y=1)+p(Y=0)\,p(X_1=1|Y=0)\,p(X_2=0|Y=0)\,p(X_3=1|Y=0)} \\ &= \frac{(4/7)\cdot(3/4)\cdot(1/2)\cdot(1/2)}{(4/7)\cdot(3/4)\cdot(1/2)\cdot(1/2) \quad + \quad (3/7)\cdot(1/3)\cdot(1/3)\cdot(1/3)} \\ &\approx 0.87 \end{aligned}$$

indicating that we should predict $Y = 1$.

Notice that our Naïve Bayes model requires us to estimate seven probabilities, as compared to 15 free parameters in the full joint distribution over X and Y. The structure imposed by the independence assumptions is a form of inductive bias that lets the model transfer information from the patterns we observed in training to make predictions about patterns we have never observed, such as $X = [1, 0, 1]$. Without this inductive bias, we would have no data with which to estimate $p(Y|X = 101)$.

Gaussian Bayes models

A nice property of the reformulation in (1.1) 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:

$$p(X|Y=y) = \mathcal{N}(X\,;\,\mu_y, \Sigma_y) = (2\pi)^{-n/2}|\Sigma_y|^{-1/2}\exp\Big(-\frac{1}{2}(X-\mu_y)\Sigma_y^{-1}(X-\mu_y)^T\Big)$$

where μ_y and Σ_y are the mean and covariance parameters of the Gaussian, and $|\cdot|$ is the matrix determinant. We can then simply estimate the parameters of that model by fitting it to the data points with class y, and then apply (1.1) to evaluate which class Y is most probable given a new observation X.

A standard way to estimate the parameters is to use the empirical means and covariances, which also correspond to the maximum likelihood estimates of the parameters:

$$\begin{aligned} m_y &= \#\{i : y^{(i)} = y\} \\ \mu_y &= \frac{1}{m_y}\sum_{i:y^{(i)}=y} x^{(i)} \\ \Sigma_y &= \frac{1}{m_y}\sum_{i:y^{(i)}=y} (x^{(i)}-\mu_y)^T \odot (x^{(i)}-\mu_y) \end{aligned}$$

where $x^T \odot x$ is an outer product, so that Σ_y is an $n \times n$ matrix.

As in the discrete case, we can reduce the number of parameters of the model by making assumptions about $p(X|Y)$. The general Gaussian model has $(c-1)$ probabilities for $p(Y)$, plus $c \cdot n$ parameters for the c class means, and $c \cdot \frac{n(n+1)}{2}$ parameters for the covariances (since each Σ_y is symmetric). Applying a naïve Bayes assumption of conditional independence of the features, for a Gaussian model, forces the Σ_y to be diagonal matrices, reducing the associated number of parameters from $c \cdot \frac{n(n+1)}{2}$ to $c \cdot n$. Alternatively, we could assume that all c Gaussian distributions share the same covariance, $\Sigma_y = \Sigma$ for all y. We can combine these two assumptions by additionally forcing Σ to be diagonal.

Any such assumptions correspond to a form of inductive bias – a preference for, or even constraint requiring, some type of explanation of the data. As before, these assumptions can help us estimate our model when we have few data: for example, if we have only a few data points i with $y^{(i)} = 1$ (but plenty with $y^{(i)} = 0$), but we assume that $\Sigma_1 = \Sigma_0 = \Sigma$, we can estimate the covariance Σ using the whole data set, rather than the noisier estimate of Σ_1 obtained using only the few $y^{(i)} = 1$ examples.

Decision boundaries. How can we characterize the types of functions f that arise from these models? One way to understand a decision function $f(x)$ for continuous-valued features x is to look at its *decision boundary*, which are the set of points x at which f transitions from predicting one class to another.

Let us consider deciding between two classes, $Y = 0$ versus $Y = 1$ using Bayes rule. We predict Y based on which class is more probable at a given point $X = x$:

$$\begin{aligned}
\text{predict } \hat{y} = 1 \Leftrightarrow \quad & \log \Big[p(Y=0)p(x|Y=0)\Big] < \log \Big[p(Y=1)p(x|Y=1)\Big] \\
\Leftrightarrow \quad & \log \left[\tfrac{p(Y=0)}{p(Y=1)}\right] < \log p(x|Y=1) - \log p(x|Y=0) \\
\Leftrightarrow \quad & 2\log \left[\tfrac{p(Y=0)}{p(Y=1)}\right] + \log \left[\tfrac{|\Sigma_1|}{|\Sigma_0|}\right] < (x-\mu_0)\Sigma_0^{-1}(x-\mu_0)^T - (x-\mu_1)\Sigma_1^{-1}(x-\mu_1)^T
\end{aligned}$$

Let us examine this decision rule in more detail in three, increasingly general cases.

The centroid rule. If both covariances $\Sigma_0 = \Sigma_1 = I$, the identity matrix, and $p(Y = 0) = p(Y = 1)$, i.e., both classes are equally probable, our prediction rule reduces to:

$$\text{predict } \hat{y} = 1 \qquad \Leftrightarrow \qquad \|x - \mu_1\|^2 < \|x - \mu_0\|^2$$

or, simply predict the class whose centroid μ_y is closer to the test point x.

Equal covariances. Now suppose only that the two covariances are equal, $\Sigma_0 = \Sigma_1 = \Sigma$. Denote by r the log-odds ratio, $r = \log[p(Y = 1)/p(Y = 0)]$. Then we have,

$$\begin{aligned}
\text{predict } \hat{y} = 1 \Leftrightarrow \quad & 0 < 2r + (x-\mu_0)\Sigma^{-1}(x-\mu_0)^T - (x-\mu_1)\Sigma^{-1}(x-\mu_1)^T \\
& = 2r + -2(x\Sigma^{-1}\mu_0^T) + 2(x\Sigma^{-1}\mu_1^T) + (\mu_0\Sigma^{-1}\mu_0^T) + (\mu_1\Sigma^{-1}\mu_1^T) \\
& = x\Sigma^{-1}(\mu_1 - \mu_0) + (\text{constant terms}),
\end{aligned}$$

which means that the decision boundary (where we transition from predicting one class to the other) is a linear function of the observed features x.

General covariances. In the general case, when $\Sigma_0 \neq \Sigma_1$, a similar argument shows that the decision boundary is a quadratic function of x,

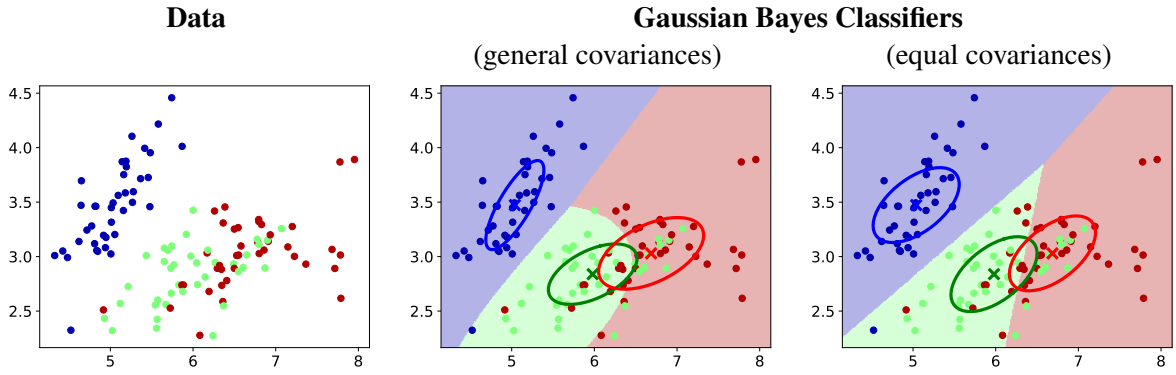
$$\text{predict } \hat{y} = 1 \Leftrightarrow \qquad 0 < x(\Sigma_0^{-1} - \Sigma_1^{-1})x^T - 2x(\Sigma_0^{-1}\mu_0 - \Sigma_1^{-1}\mu_1) + (\text{constant terms}).$$

Characterizing the decision boundaries gives us another way of understanding, and visualizing, the types of decision functions that our learner can produce. We illustrate this point with an example:

Example 1-6 : Fisher Iris Data

Consider a data set on irises, slightly adapted[2] from that of Fisher [1936]. By measuring several features of an iris flower (specifically, its petal and sepal length and width), we can design a classifier that predicts to which of three different species of iris (*Iris setosa*, *Iris versicolor*, and *Iris virginica*) a given specimen belongs.

If we restrict ourselves to only two features (sepal length and width, for example), we can plot these data points and color them according to their class (blue, green, and red, respectively). Then, if we estimate a Gaussian model $p(X|Y = y)$ for each class y, we can use Bayes rule to predict the class at each point in the plot, and visualize the resulting decision function and its boundaries:



If we force the three Gaussian models to share the same covariance, we see that as discussed, the decision boundaries between each pair of classes change from the more general quadratic form, to a straight line.

Classification, decisions, and hypothesis testing

A critical component of most machine learning is a mechanism to quantify and compare the performance of different models, typically codified in terms of a loss function. We have already seen two common loss functions, the mean squared error J_{MSE} and the misclassification error rate or 0/1 loss, J_{01}. While we will see a number of other possible loss functions in subsequent chapters, here we discuss the relationship between classification tasks, statistical hypothesis testing and decision theory, to give some context to classification losses.

Measuring classification errors

Suppose that we are trying to make a binary decision, for example, deciding whether a patient has a particular disease. In such cases, we can label one of the two classes, $Y = 1$, as "positive" (e.g., patient has the disease), and the other (typically denoted by either $Y = 0$ or $Y = -1$) as "negative" (patient does not). We observe some information, X, about the patient (symptoms, test results, etc.) and make our decision, $\hat{Y} = f(X)$. In such a setting, there are two types of mistakes we can make, which might have different consequences. First, we could predict that the patient has the disease ($\hat{Y} = 1$) when in reality

[2] Specifically, to avoid the need to address tiebreaking in our later computations, we add a small amount of noise to the feature values, and remove a few data points. In general, tiebreaking can be handled arbitrarily, but the results may depend on the tiebreaking rule.

they do not ($Y = 0$), called a **false positive** or **type 1** error. Or, we could predict that the patient does not have the disease ($\hat{Y} = 0$) when in fact they do ($Y = 1$), called a **false negative** or **type 2** error.

Different types of error may have different real-world consequences, making us prefer to make one type of mistake over another. In spam filtering, we might prefer to predict negative (not spam), since the cost of a mislabeled spam message is that the user must delete it manually, while the cost of a mislabeled real email is that the user may miss important information. Conversely, in a medical test, the cost of a false negative may be that the patient recieves no treatment and becomes very ill, while a false positive may simply lead to more accurate but expensive testing. For this reason, classifiers often report false positive and false negative error rates separately, rather than as a single, overall error rate; for a given data set D we can compute:

$$\text{fpr} = \frac{\sum_i \mathbb{1}[y^{(i)} = 0 \;\&\; f(x^{(i)}) = 1]}{\sum_i \mathbb{1}[y^{(i)} = 0]} \quad \text{(false positive rate)} \qquad \text{fnr} = \frac{\sum_i \mathbb{1}[y^{(i)} = 1 \;\&\; f(x^{(i)}) = 0]}{\sum_i \mathbb{1}[y^{(i)} = 1]} \quad \text{(false negative rate)}$$

Closely related to the false positive and false negative rates are the **sensitivity** and **specificity** of the test, which are measures of accuracy that correspond to the "true positive rate" and "true negative rate" respectively:

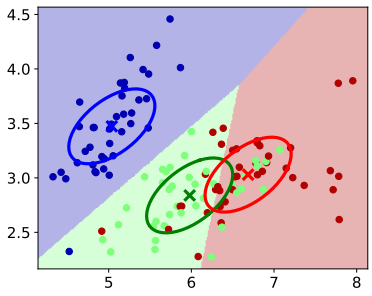
$$\text{sen} = \frac{\sum_i \mathbb{1}[y^{(i)} = 1 \;\&\; f(x^{(i)}) = 1]}{\sum_i \mathbb{1}[y^{(i)} = 1]} \quad \text{(sensitivity)} \qquad \text{spe} = \frac{\sum_i \mathbb{1}[y^{(i)} = 0 \;\&\; f(x^{(i)}) = 0]}{\sum_i \mathbb{1}[y^{(i)} = 0]} \quad \text{(specificity)}$$

The sensitivity of a test measures how well it detects positive examples, while the specificity of a test measures how well it rejects negative examples.

In problems with more than two classes (i.e., $c > 2$), we can understand what types of mistakes are being made by computing the **confusion matrix**, a $c \times c$ matrix whose i, j entry corresponds to the number of data points s for which the true class $y^{(s)} = i$, and our prediction $f(x^{(s)}) = j$. From this representation, it is easy to see which classes of data tend to be mis-predicted as ("confused with") other classes.

Example 1-7 : Fisher Iris Confusion Matrix

Suppose that we have the Gaussian Bayes classifier at right. We can compute its confusion matrix on the data D as,



	$\hat{y} =$	0	1	2
	0	39	1	0
$y =$	1	0	27	11
	2	0	11	29

From this table, it is easy to see that class $y = 0$ (blue) are predicted quite accurately, while classes $y = 1$ and $y = 2$) (green, red) are more easily mis-predicted as each other, in approximately equal numbers.

If certain types of errors are more important than others, one way to capture this is to associate a reward (often negative, indicating a cost) with certain predictions or errors. Suppose we associate reward $r(y, \hat{y})$ with making prediction $\hat{y}$ when the true class is y. Then the optimal prediction, in the sense of maximizing the expected reward, has the form

$$f^*(x) = \arg\max_{\hat{y}} \sum_y p(Y = y|x) r(y, \hat{y});$$

we can see that this reduces to the Bayes optimal f^* when r penalizes all errors equally, e.g., $r(y, \hat{y}) = -\mathbb{1}[y \neq \hat{y}]$.Typically, the best prediction for a given y is the correct class, and we can define its reward as $r(y, y) = \max_{\hat{y}} r(y, \hat{y}) = 0$ without loss of generality, with other, incorrect predictions having negative values.

When the class Y is binary and $r(y, y) = 0$, we can see that f^* can be written,

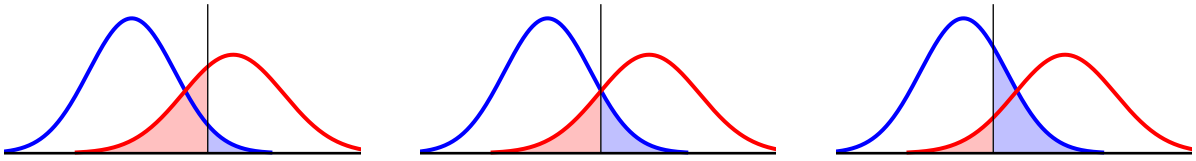
$$\text{Decide } f^*(x) = 1 \text{ iff} \qquad p(Y = 0|x) r(0, 1) \geq p(Y = 1|x) r(1, 0)$$

$$p(Y = 0|x) \leq \frac{r(1, 0)}{r(0, 1)} p(Y = 1|x),$$

so that we have the same rule as a Bayes classifier, except that we include a term that makes our decision prefer class 1 by multiplying by the ratio of costs, $\alpha = r(1, 0)/r(0, 1)$. So, for example, if the cost $-r(1, 0)$ of a false negative is $3\times$ higher than the cost $-r(0, 1)$ of a false positive, then we will predict $f(x) = 1$ unless the probability of class 0 is more than three times more than the probability of class 1.

Example 1-8 : Balancing Error Types

Suppose that we have a scalar measurement $x \in \mathbb{R}$, binary $Y \in \{0, 1\}$, and each $p(X|Y = y)$ is Gaussian with equal variance, $\mathcal{N}(x; \mu_y, \sigma^2)$, with $\mu_1 > \mu_0$. Then, the optimal $f^*(x)$ will be to pick class 1 if $x > t(\alpha)$ for some threshold t that depends on the cost ratio α. If we plot the scaled density $p(x, Y = 1)$, so that the area under the density is $p(Y = 1)$, we can visualize the probability of a false negative ($Y = 1$, but we predict class 0) as the area under the density to the left of threshold t. Adjusting α moves the threshold left or right, changing the balance of false positives (shaded blue – data from the negative class that are above the threshold and so classified as positive) versus false negatives (shaded red – data from the positive class below the threshold):



$\alpha < 1$
(reduce false positives, but more false negatives)

$\alpha = 1$
(fewest errors)

$\alpha > 1$
(reduce false negatives, but more false positives)

As we change α, we can decrease one error type at the cost of increasing the other; setting $\alpha = 1$ has the lowest error rate (total area).

In general, any model that can output confidence estimates ("soft" predictions) can be adjusted to change the balance of error types, simply by changing the threshold used to convert the confidence estimates into class decisions (or, "hard" predictions).

ROC curves

Given that we can change the balance of the types of errors that our learner makes, it may be of interest to characterize the trade-off in type 1 versus type 2 errors we could obtain. Is our model good at identifying high-confidence negative examples, but has more difficulty with high-confidence positive examples? Are there examples of both types that it can easily identify, but also a number of data that are ambiguous? One way to visualize the performance of our model across different settings is the **receiver operating characteristic curve**, or **ROC curve**. The ROC curve was originally developed during World War II to characterize the ability of radar receiver operators to correctly detect enemy airplanes (hence the name). Practically speaking, the ROC curve is a graphical characterization of the tradeoff between false and true positives as the classifier's decision threshold is altered.

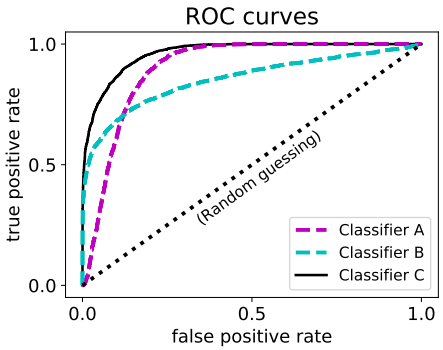
Suppose that we have a decision rule that achieves a false positive rate f and true positive rate t, which we plot at location $(f, t) \in [0, 1]^2$. Ideally, our classifier is close to point $(0, 1)$, corresponding to a perfect classifier (no false positives, all true positives correct). If we imagine changing the decision threshold, we can reduce f, but at the expense of also reducing t, or vice versa, and we trace these points out to form the ROC curve. At one extreme, we predict all negatives, $\hat{f}(x) \equiv 0$, in which case we have no false positives, but also zero true positives, $(f, t) = (0, 0)$. At the other extreme, we predict all positives, $\hat{f}(x) \equiv 1$, giving $(f, t) = (1, 1)$.

Moreover, suppose that we have two different classifiers $\hat{f}$ and $\hat{f}'$, with rates (f, t) and (f', t') respectively. Then, if we *randomly* select $\hat{f}$'s decision with probability ρ, and $\hat{f}'$ with probability $(1 - \rho)$, we can see that our false positive and true positive rates will be $\rho f + (1 - \rho)f'$ and $\rho t + (1 - \rho)t'$, so that we can achieve rates at any point along the line conneting (f, t) and (f', t'). As a correlary, a completely random classifier that simply predicts positive with probability ρ (ignoring x completely) achieves point (ρ, ρ). Any reasonable model should perform better than this, and so its ROC curve will lie above the $(0, 0) - (1, 1)$ diagonal.

We say that one classifier *dominates* another if it has an equal or higher true positive rate at every possible false positive rate, i.e., its ROC curve is always above the other classifier's. However, it is possible that neither classifier dominates the other, in which case one classifier may be better than the other, depending on how we weight false positives versus false negatives.

Example 1-9 : ROC Curves

The figure at right shows ROC curves for three different classifiers on a synthetically generated data set. We see that neither classifier A or B (dashed lines) dominates the other. Classifier B performs better at finding true positives while allowing few or no false positives (high false positive cost), while classifier A is better if we wish to find all or nearly all true positives while allowing few false positives (high false negative cost). Classifier C dominates both A and B; it gives a higher true positive rate at every possible false positive rate. Random guessing gives an equal fraction of false positives and true positives (dotted line).



The ROC curve gives us detailed information about how the model performs for different error type costs. It can also be used to provide a single summary of performance, such as the *area under the curve*

or **AUC** score, which evaluates the total area under the ROC curve. A perfect classifier will have an AUC of one, while random guessing will have AUC of $\frac{1}{2}$.

Connection to hypothesis testing

Many of the same concepts that underly classification also form the foundation of statistical hypothesis testing, for evaluating scientific hypotheses. In hypothesis testing, we typically have an uninteresting or "null" hypothesis, H_0, and an alternative hypothesis, H_1. We gather some experimental data, which we hope to use to answer the question: is H_0 potentially correct, or do we reject it in favor of H_1? For example, if we have developed an experimental drug, H_0 might be that it has no effect on the disease, while H_1 might be that it improves outcomes. How can we decide?

For any test, we can define its level of significance as its false positive rate: the probability that although H_0 is true, we happen to observe data that leads us to predict H_1. Meanwhile, the power of the test (for a specific H_1) is simply its true positive rate: the probability that, if H_1 is true, the data we observe will pass our test. As with our ROC curves, if we demand fewer false positives (a smaller signficance level), we risk reducing the true positive rate. (The related concept of the **p-value** of our observed data is the smallest significance level for which those data would predict H_1.)

Neyman and Pearson characterized the "most powerful" test for a given significance level, and showed that, if it exists, it is equivalent to the **likelihood ratio test**, which compares the likelihood ratio, $p(D|H_1)/p(D|H_0)$, to a threshold.

include? add/clarify?