Chapter 5

Introduction to Prediction Problems

Machine learning addresses many problem settings, which can sometimes feel overwhelming. As a non-exhaustive list, these include supervised learning (with classification and regression), semi-supervised learning, unsupervised learning, completion under missing features, structured prediction, learning to rank, statistical relational learning, active learning, and temporal prediction (with time series prediction and policy evaluation in reinforcement learning and online learning). For some of these settings, such as active learning and reinforcement learning, the data collection is a central part of the algorithm and can significantly determine the quality of the learned predictive models. Most other settings assume that data has been collected—without our ability to influence that collection—and now we simply need to analyze that data and learn the best predictors that we can. In this passive setting, we can either assume that the data is i.i.d.—which is the most common—or that there are dependencies between data points—such as in time series prediction or statistical relational learning. There are also settings where the data is incomplete, say because someone did not fill in their age in a survey or a person has not seen and rated a movie. Finally, in some scenarios we are given a data set and expect to train a model to make predictions on the data that comes in the future (inductive setting); however, in certain domains the data on which predictions are desired may already be available and so the knowledge of these points may influence how the model is trained and applied (transductive setting).

One ontology, therefore, could consider the following dimensions to categorize machine learning problems: passive vs. active, i.i.d. vs. non-i.i.d., complete vs. incomplete, inductive vs. transductive. As with all ontologies, each problem will not perfectly fit into these categories. Further, it is likely that most data collection is not completely passive (even if only because the human modeler influences collection of data), is likely not i.i.d. (even if we intended it to be), and likely has some missing components. Nonetheless, algorithms will make these assumptions, to varying degrees, even if the data does not satisfy the assumptions. For the majority of these notes, we will focus on the simplest setting: passive, i.i.d. and complete. Further, because so many techniques arise from the basic classification and regression models within supervised learning, we will focus mostly on these.

5.1 Supervised learning

We start by defining a data set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}$, where $\boldsymbol{x}_i \in \mathcal{X}$ is the i-th object and $y_i \in \mathcal{Y}$ is the corresponding target designation. We usually assume that $\mathcal{X} = \mathbb{R}^d$, in which case $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})$ is a d-dimensional vector called a *data point*, *example* or *instance*. Each dimension of x_j is typically called a *feature* or an *attribute*. We will often organize the dataset into a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ where each row corresponds to a data point and each column corresponds to a feature, see Figure 5.1.

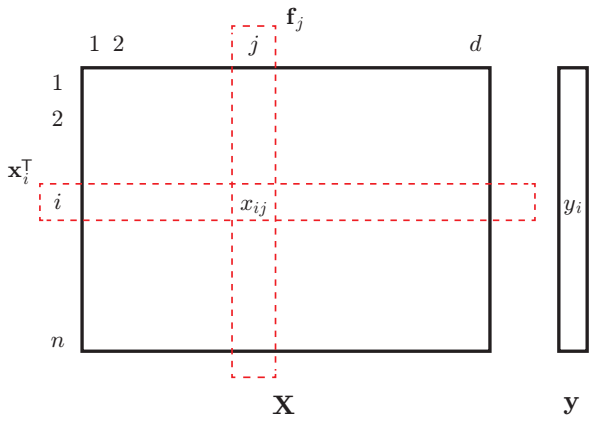

Figure 5.1: Data set representation and notation. $\mathbf{X}$ *is an* n*-by-*d *matrix representing data points (rows) and features (columns), whereas* $\mathbf{y}$ *is an* n*-by-*1 *vector of targets. Each data point can be seen as a* d*-dimensional column vector* $\mathbf{x}$*.* $\mathbf{X}$ *is sometimes referred to as the design matrix.*

We generally distinguish between two related but different types of prediction problems: classification and regression. Generally speaking, we have a classification problem when $\mathcal{Y}$ is discrete and a regression problem when $\mathcal{Y}$ is continuous. Additionally, we might specify this distinction in the solution approach; i.e., using a classification approach (e.g., perceptron) or a regression approach (e.g., linear regression). In both prediction scenarios, we assume that the features are easy to collect for each object (e.g., by measuring the height of a person or the square footage of a house), while the target variable is difficult to observe or expensive to collect (e.g., final selling price of a house). Such situations usually benefit from the construction of a computational model that predicts targets from a set of input values. The model is trained using a set of input objects for which target values have already been collected.

In **classification** we construct a function that predicts discrete class labels; this function is typically called a *classifier*. The cardinality of $\mathcal{Y}$ in classification problems is usually small; e.g., $\mathcal{Y} = \{\text{healthy}, \text{diseased}\}$. An example of a data set for classification with $n = 3$ data points and $d = 5$ features is shown in Table 5.1.

Classification problems can be further subdivided into multi-class and multi-label problems. A multi-class problem consists of providing the single label for an input. For example, for (simple) blood-type with $\mathcal{Y} = \{\text{A}, \text{B}, \text{AB}, \text{O}\}$, a patient can only be labeled with one of these labels. Within multi-class problems, if there are only two classes, it is called binary classification, such as the example in Table 5.1. In multi-label, an input can be associated with more than one label. An example of a multi-label problem is the classification of text documents into categories such as $\{\text{sports}, \text{medicine}, \text{travel}, \text{politics}\}$. Here, a single document may be related to more than one value in the set; e.g., an article on sports medicine. The learned function can now return multiple outputs.

Typically, to make the outputs more consistent between these two settings, the output for both multi-class and multi-label is an indicator vector. For $m = |\mathcal{Y}|$, the prediction for blood types might be $(0, 1, 0, 0)$ to indicate blood-type B and the prediction for four article labels could be $(1, 1, 0, 0)$ if it is both an article pertaining to sports and medicine. Of course, multi-class classification can be seen as an instance of multi-label classification with exactly

	wt [kg]	ht [m]	T [°C]	sbp [mmHg]	dbp [mmHg]	y
$\boldsymbol{x}_1$	91	1.85	36.6	121	75	-1
$\boldsymbol{x}_2$	75	1.80	37.4	128	85	$+1$
$\boldsymbol{x}_3$	54	1.56	36.6	110	62	-1

Table 5.1: An example of a binary classification problem: prediction of a disease state for a patient. Here, features indicate weight (wt), height (ht), temperature (T), systolic blood pressure (sbp), and diastolic blood pressure (dbp). The class labels indicate presence of a particular disease; e.g., diabetes. This data set contains one positive data point ($\boldsymbol{x}_2$) and two negative data points ($\boldsymbol{x}_1$, $\boldsymbol{x}_3$). The class label shows a disease state; i.e., $y_i = +1$ indicates the presence while $y_i = -1$ indicates absence of disease.

one label being possible. Similarly, multi-label classification can be seen as multi-class on the output space being the power set of the original output space. The distinction between these settings is thus not clear-cut and, in practice, is often related to the meaning of the concepts in the output space.

The output space $\mathcal{Y}$ can also have internal structure; e.g., strings, trees, or graphs. This classification scenario is usually referred to as *structured-output learning*. The cardinality of the output space in structured-output learning problems is often very high. For example, when predicting functionality of a protein (i.e., what it does in and outside the cell), an entire subgraph of the ontology graph needs to be predicted, respecting all hierarchical relationships as certain functionality is an instance of other functionality ("protein binding" and "DNA binding" are both instances or specifications of the concept "macromolecular binding").

In **regression** we construct a function to predict continuous targets, such as when $\mathcal{Y} = \mathbb{R}$ or $\mathcal{Y} = [0, \infty)$. An example of a regression problem is shown in Table 5.2.

	size [sqft]	age [yr]	dist [mi]	inc [$]	dens [ppl/mi^2]	y
$\boldsymbol{x}_1$	1250	5	2.85	56,650	12.5	2.35
$\boldsymbol{x}_2$	3200	9	8.21	245,800	3.1	3.95
$\boldsymbol{x}_3$	825	12	0.34	61,050	112.5	5.10

Table 5.2: An example of a regression problem: prediction of the price of a house in a particular region. Here, features indicate the size of the house (size) in square feet, the age of the house (age) in years, the distance from the city center (dist) in miles, the average income in a one square mile radius (inc), and the population density in the same area (dens). The target indicates the price a house is sold at, here in hundreds of thousands of dollars.

How should we formalize the problem? There does not exist a strict distinction between classification and regression. For example, consider the output space $\mathcal{Y} = \{0, 1, 2\}$. We can treat this as a multi-class classification problem, or we could presume $\mathcal{Y} = [0, 2]$ and learn a regression model. We can then threshold the predictions returned by the regression model, by rounding them to the closest integer.

How then do we decide which problem formulation to use? Though the mathematical procedures in machine learning are precise, deciding how to formulate real-world problem is subtle, and so inherently less clear-cut. The selection of a particular way of modeling depends on the analyst and their knowledge of the domain as well as technical aspects of learning. In this example, we could ask: is there inherently an ordering to the outputs $\{0, 1, 2\}$? If not, say they correspond to Prefers apples, Prefers oranges, Prefers bananas, then it may be a poor choice to model the output as an interval, which often implies ordering. On the other hand, regression functions can be easier to learn and often produce surprisingly good classification predictions. Further, if there is an ordering to these class, say Good, Better, Best or Tall, Grande, Venti, then most classification models—which do not assume an ordering on the outputs—would not be able to take advantage of this ordering to improve prediction performance.

Selecting the function class, and objective, just like selecting distributions and priors, is an important step in using machine learning effectively. Fortunately, there is a wealth of knowledge, especially empirically, that can guide this selection. As we learn more about the methods, combined with some information about structure in our domain of interest, we will become better at specifying the model class.

5.2 Unsupervised and semi-supervised learning

Data sets are not always complete. In some cases, we can only acquire labels for a small subset of instances, or we cannot get any labels at all. For example, when predicting whether a cat is in an image or not, we would need a human to take each image and label it with a 0 or 1. This labeling can be expensive, and so we can only expect that a small number of all pictures with cats have such an associated label. Using supervised learning only on this labeled subset is likely to produce a poor predictor, because of limited data.

Semi-supervised learning deals with taking advantage of all the unlabeled data, to supplement the small labeled data set, by finding structure in the features. For example, the features may lie on a lower-dimensional manifold; this structure could be inferred from the unlabeled data, and potentially more effectively restrict the function class to learn on the labeled data. *Unsupervised learning* is focused only on obtaining this structure, without the goal of learning a function to predict targets, because no targets are provided. We have already seen unsupervised learning when discussing the EM algorithm and its variants. It will be discussed more thoroughly in Section 9.2.2, as a part of representation learning.

A special form of semi-supervised learning that occurs in classification is when all labeled examples belong to a single class, say positive class. This setting often occurs in open-world domains where the absence of an observation cannot be considered to be the evidence of absence and, thus, one cannot practically collect negative data. It is referred to as *positive-unlabeled learning*. An example of this setting is predicting whether an individual likes a particular comment in a social network that allows only positive feedback. Prediction of binding between different types of molecules is another instance where it is difficult and sometimes impossible to label a negative interaction. Interestingly, positive-unlabeled learning can also be seen as an instance of traditional supervised learning in which there exists class-label noise in the negative examples.

The unsupervised and semi-supervised settings can be seen as instances of a larger setting of learning under missing data. In general, it may not only be difficult to gather the

outputs, but also some of the features. For example, when collecting patient data, it is likely that some patients will omit some information. Even though it is difficult to gather the information "has disease", it can also be difficult to ensure that other more straightforward data such as "age" or "weight" is gathered. Further, one might even ask why there is a distinction between features and targets—they are all associated information about one item, like a patient. Given that a patient does have a disease, you may want to use this feature and their age to predict their weight—which they so blithely chose not to disclose. This more general way of approaching the problem can be useful when data is missing and leads to the general problem of completion. Different techniques are often used in such a setting and we will not address it further until Section 9.2.2. For now, we shall keep the focus on supervised learning, which we will still be able to use even when some features are missing; e.g., by using simple heuristics for dealing with this missing data.

5.3 Optimal classification and regression models

Our goal now is to establish the performance criteria that will be used to evaluate predictors $f : \mathcal{X} \to \mathcal{Y}$ and subsequently define optimal classification and regression models. We will formalize this by assuming a situation where $\boldsymbol{X}$ is the random vector representing features, Y is the random variable representing the target, and that their joint probability distribution $p(\boldsymbol{x}, y)$ is either known or can be learned from data.

We will first look at classification and suppose that we are given a cost (or loss) function $\text{cost} : \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$, where for each prediction $f(\boldsymbol{x}) = \hat{y}$ and true target value y the classification cost can be expressed as a constant $\text{cost}(\hat{y}, y)$, regardless of the input $\boldsymbol{x} \in \mathcal{X}$ given to the classifier. This cost function can simply be stored as a $|\mathcal{Y}| \times |\mathcal{Y}|$ cost matrix.

The criterion for optimality of a classifier will be probabilistic. In particular, we are interested in minimizing the expected cost, for random variable $C = \text{cost}(f(\boldsymbol{X}), Y)$,

$$\begin{aligned}\mathbb{E}[C] &= \int_{\mathcal{X}} \sum_{y} \text{cost}(\hat{y}, y) p(\boldsymbol{x}, y) d\boldsymbol{x} \\ &= \int_{\mathcal{X}} p(\boldsymbol{x}) \sum_{y} \text{cost}(\hat{y}, y) p(y|\boldsymbol{x}) d\boldsymbol{x},\end{aligned}$$

where the integration is over the entire input space $\mathcal{X} = \mathbb{R}^d$. From this equation, we can see that the optimal classifier can be expressed as

$$f_{\text{BR}}(\boldsymbol{x}) = \arg\min_{\hat{y} \in \mathcal{Y}} \left\{ \sum_{y} \text{cost}(\hat{y}, y) p(y|\boldsymbol{x}) \right\},$$

for any $\boldsymbol{x} \in \mathcal{X}$. We will refer to this classifier as the *Bayes risk classifier*. One example where the Bayes risk classifier can be useful is the medical domain. Suppose our goal is to decide whether a patient with a particular set of symptoms ($\boldsymbol{x}$) should be sent for an additional lab test ($y = 1$ if yes and $y = -1$ if not), with cost c_{lab}, in order to improve diagnosis. However, if we do not perform a lab test and the patient is later found to have needed the test for proper treatment, we may incur a significant penalty, say c_{lawsuit}. If $c_{\text{lawsuit}} \gg c_{\text{lab}}$, as it is expected to be, then the classifier needs to appropriately adjust its outputs to account for the cost disparity in different forms of incorrect prediction.

In many practical situations, however, it may not be possible to define a meaningful cost matrix and, thus, a reasonable criterion would be to minimize the probability of a classifier's error $P(f(\boldsymbol{x}) \neq y)$. This corresponds to the situation where the cost function is defined as

$$\text{cost}(\hat{y}, y) = \begin{cases} 0 & \text{when } y = \hat{y} \\ \\ 1 & \text{when } y \neq \hat{y} \end{cases}$$

After plugging these values in the definition for $f_{\text{BR}}(\boldsymbol{x})$, the Bayes risk classifier simply becomes the maximum a posteriori (MAP) classifier. That is,

$$f_{\text{MAP}}(\boldsymbol{x}) = \arg\max_{y \in \mathcal{Y}} \{p(y|\boldsymbol{x})\}.$$

Therefore, if $p(y|\boldsymbol{x})$ is known or can be accurately learned, we are fully equipped to make the prediction that minimizes the total cost. In other words, we have converted the problem of minimizing the expected classification cost or probability of error, into the problem of learning functions, more specifically learning probability distributions.

The analysis for regression is a natural extension of that for classification. Here too, we are interested in minimizing the expected cost of prediction of the true target y when a predictor $f(\boldsymbol{x})$ is used. The expected cost can be expressed as

$$\mathbb{E}[C] = \int_{\mathcal{X}} \int_{\mathcal{Y}} \text{cost}(f(\boldsymbol{x}), y) p(\boldsymbol{x}, y) dy d\boldsymbol{x},$$

where $c : \mathbb{R} \times \mathbb{R} \to [0, \infty)$ is again some cost function between the predicted value $f(\boldsymbol{x})$ and the true value y. For simplicity, we will consider

$$\text{cost}(f(\boldsymbol{x}), y) = (f(\boldsymbol{x}) - y)^2,$$

which results in

$$\begin{aligned} \mathbb{E}[C] &= \int_{\mathcal{X}} \int_{\mathcal{Y}} (f(\boldsymbol{x}) - y)^2 p(\boldsymbol{x}, y) dy d\boldsymbol{x} \\ &= \int_{\mathcal{X}} p(\boldsymbol{x}) \underbrace{\int_{\mathcal{Y}} (f(\boldsymbol{x}) - y)^2 p(y|\boldsymbol{x}) dy}_{g(f(\boldsymbol{x}))} d\boldsymbol{x}. \end{aligned}$$

Assuming $f(\boldsymbol{x})$ is flexible enough to be separately optimized for each unit volume $d\boldsymbol{x}$, we see that minimizing $\mathbb{E}[C]$ leads us to the problem of minimizing

$$g(u) = \int_{\mathcal{Y}} (u - y)^2 p(y|\boldsymbol{x}) dy,$$

where we used a substitution $u = f(\boldsymbol{x})$. We can now differentiate g with respect to u as

$$\begin{aligned} &\frac{\partial g(u)}{\partial u} = 2 \int_{\mathcal{Y}} (u - y) p(y|\boldsymbol{x}) dy = 0 \\ &\implies u \underbrace{\int_{\mathcal{Y}} p(y|\boldsymbol{x}) dy}_{=1} = \int_{\mathcal{Y}} y p(y|\boldsymbol{x}) dy \end{aligned}$$

which results in the optimal solution

$$\begin{aligned} f^*(\boldsymbol{x}) &= \int_{\mathcal{Y}} yp(y|\boldsymbol{x})dy \\ &= \mathbb{E}[Y|\boldsymbol{x}]. \end{aligned}$$

Therefore, the optimal regression model in the sense of minimizing the square error between the prediction and the true target is the conditional expectation $\mathbb{E}[Y|\boldsymbol{x}]$. It may appear that in the above equations, setting $f(\boldsymbol{x}) = y$ would always lead to $\mathbb{E}[C] = 0$. Unfortunately, this would be an invalid operation because for a single input $\boldsymbol{x}$ there may be multiple possible outputs y and they can certainly appear in the same data set. To be a well-defined function, $f(\boldsymbol{x})$ must always have the same output for the same input. $\mathbb{E}[C] = 0$ can only be achieved if $p(y|\boldsymbol{x})$ is a delta function for every $\boldsymbol{x}$.

Having found the optimal regression model, we can now write the expected cost in the cases of both optimal and suboptimal models $f(\boldsymbol{x})$. That is, we are interested in expressing $\mathbb{E}[C]$ when

1. $f(\boldsymbol{x}) = \mathbb{E}[Y|\boldsymbol{x}]$
2. $f(\boldsymbol{x}) \neq \mathbb{E}[Y|\boldsymbol{x}]$.

We have already found $\mathbb{E}[C]$ when $f(\boldsymbol{x}) = \mathbb{E}[Y|\boldsymbol{x}]$. The expected cost can be simply expressed as

$$\mathbb{E}[C] = \int_{\mathcal{X}} \int_{\mathcal{Y}} (\mathbb{E}[Y|\boldsymbol{x}] - y)^2 p(\boldsymbol{x}, y) dy d\boldsymbol{x},$$

which corresponds to the (weighted) squared error between the optimal prediction and the target everywhere in the feature space. This is the best scenario in regression; we cannot achieve a better solution on average for this squared error cost.

The next situation is when $f(\boldsymbol{x}) \neq \mathbb{E}[Y|\boldsymbol{x}]$. Here, we will proceed by decomposing the squared error as

$$\begin{aligned} (f(\boldsymbol{x}) - y)^2 &= (f(\boldsymbol{x}) - \mathbb{E}[Y|\boldsymbol{x}] + \mathbb{E}[Y|\boldsymbol{x}] - y)^2 \\ &= (f(\boldsymbol{x}) - \mathbb{E}[Y|\boldsymbol{x}])^2 + 2\underbrace{(f(\boldsymbol{x}) - \mathbb{E}[Y|\boldsymbol{x}])(\mathbb{E}[Y|\boldsymbol{x}] - y)}_{g(\boldsymbol{x},y)} + (\mathbb{E}[Y|\boldsymbol{x}] - y)^2 \end{aligned}$$

We now take a more detailed look at $g(\boldsymbol{x}, y)$ when placed under the expectation over $p(\boldsymbol{x}, y)$

$$\begin{aligned} \mathbb{E}[g(\boldsymbol{X}, Y)] &= \int_{\mathcal{X}} \int_{\mathcal{Y}} (f(\boldsymbol{x}) - \mathbb{E}[Y|\boldsymbol{x}])(\mathbb{E}[Y|\boldsymbol{x}] - y) p(\boldsymbol{x}, y) dy d\boldsymbol{x} \\ &= \int_{\mathcal{X}} \int_{\mathcal{Y}} f(\boldsymbol{x}) \mathbb{E}[Y|\boldsymbol{x}] p(\boldsymbol{x}, y) dy d\boldsymbol{x} - \int_{\mathcal{X}} \int_{\mathcal{Y}} f(\boldsymbol{x}) yp(\boldsymbol{x}, y) dy d\boldsymbol{x} \\ &\quad + \int_{\mathcal{X}} \int_{\mathcal{Y}} \mathbb{E}[Y|\boldsymbol{x}] yp(\boldsymbol{x}, y) dy d\boldsymbol{x} - \int_{\mathcal{X}} \int_{\mathcal{Y}} \mathbb{E}[Y|\boldsymbol{x}] \mathbb{E}[Y|\boldsymbol{x}] p(\boldsymbol{x}, y) dy d\boldsymbol{x} \\ &= \int_{\mathcal{X}} f(\boldsymbol{x}) \mathbb{E}[Y|\boldsymbol{x}] p(\boldsymbol{x}) d\boldsymbol{x} - \int_{\mathcal{X}} f(\boldsymbol{x}) p(\boldsymbol{x}) \int_{\mathcal{Y}} yp(y|\boldsymbol{x}) dy d\boldsymbol{x} \\ &\quad \int_{\mathcal{X}} \mathbb{E}[Y|\boldsymbol{x}] \mathbb{E}[Y|\boldsymbol{x}] p(\boldsymbol{x}) d\boldsymbol{x} - \int_{\mathcal{X}} \mathbb{E}[Y|\boldsymbol{x}] \mathbb{E}[Y|\boldsymbol{x}] p(\boldsymbol{x}) d\boldsymbol{x} \\ &= 0. \end{aligned}$$

Therefore, we can now express the expected cost as

$$\begin{aligned}\mathbb{E}[C] &= \int_{\mathcal{X}} \int_{\mathcal{Y}} (f(\boldsymbol{x}) - y)^2 p(\boldsymbol{x}, y) dy d\boldsymbol{x} \\ &= \int_{\mathcal{X}} (f(\boldsymbol{x}) - \mathbb{E}[Y|\boldsymbol{x}])^2 p(\boldsymbol{x}) d\boldsymbol{x} + \int_{\mathcal{X}} \int_{\mathcal{Y}} (\mathbb{E}[Y|\boldsymbol{x}] - y)^2 p(\boldsymbol{x}, y) dy d\boldsymbol{x},\end{aligned}$$

where the first term is the "distance" between the trained model $f(\boldsymbol{x})$ and the optimal model $\mathbb{E}[Y|\boldsymbol{x}]$ and the second term is the "distance" between the optimal model $\mathbb{E}[Y|\boldsymbol{x}]$ and the correct target value y. These terms are often called the *reducible* and *irreducible* errors, respectively. If we extend the class of functions f to predict $\mathbb{E}[Y|\boldsymbol{x}]$, we can reduce the first expected error. However, the second distance is an inherent error, where for any inputs $\boldsymbol{x}$, there is a distribution over possible values that we can observe. This relates to the problem of partial observability, where there is always some stochasticity due to a lack of information. This irreducible distance could potentially be further reduced by providing more features (i.e., extending $\boldsymbol{x}$). However, for a given data set, with the given features, this error is irreducible.

To sum up, we argued here that optimal classification and regression models critically depend on knowing or accurately learning the posterior distribution $p(y|\boldsymbol{x})$. This task can be solved in different ways, but a straightforward approach is to assume a functional form for $p(y|\boldsymbol{x})$, say $p(y|\boldsymbol{x}, \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a set of weights or parameters that are to be learned from the data. Alternatively, we can learn the class-conditional and prior distributions, $p(\boldsymbol{x}|y)$ and $p(y)$, respectively. Using

$$\begin{aligned}p(y|\boldsymbol{x}) &= \frac{p(\boldsymbol{x}, y)}{p(\boldsymbol{x})} \\ &= \frac{p(\boldsymbol{x}|y)p(y)}{\sum_y p(\boldsymbol{x}, y)} \\ &= \frac{p(\boldsymbol{x}|y)p(y)}{\sum_y p(\boldsymbol{x}|y)p(y)}\end{aligned}$$

we can see that these two learning approaches are equivalent in theory. The choice depends on our prior knowledge and/or preferences.

Models obtained by directly estimating $p(y|\boldsymbol{x})$ are called *discriminative models* and models obtained by directly estimating $p(\boldsymbol{x}|y)$ and $p(y)$ are called *generative models*. Direct estimation of the joint distribution $p(\boldsymbol{x}, y)$ is relatively rare as it may be more difficult to hypothesize its parametric or non-parametric form from which the parameters are to be found. Finally, in some situations we can train a model without an explicit probabilistic approach in mind. In these cases, we typically aim to show a good performance of an algorithm in practice, say on a large number of data sets, according to a performance measure relevant to the problems at hand.

5.4 Bayes Optimal Models

We saw earlier that optimal prediction models reduce to the learning of the posterior distribution $p(y|\boldsymbol{x})$ which is then used to minimize the expected cost (risk, loss). However, in practice, the probability distribution $p(y|\boldsymbol{x})$ must be modeled using a particular functional

form and a set of tunable coefficients. When $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{0, 1\}$, one such example is used in logistic regression, where

$$p(1|\boldsymbol{x}) = \frac{1}{1 + e^{-(w_0 + \sum_{j=1}^{d} w_j x_j)}}$$

and $p(0|\boldsymbol{x}) = 1 - p(1|\boldsymbol{x})$. Here $(w_0, w_1, \ldots, w_d) \in \mathbb{R}^{d+1}$ is a set of weights that are to be inferred from a given data set $\mathcal{D}$ and $\boldsymbol{x} \in \mathbb{R}^d$ is an input data point. A number of other types of functional relationships can be used as well, providing a vast set of possibilities for modeling distributions.

To be more precise about these functional forms, we should adjust our notation to denote the distribution over y given $\boldsymbol{x}$ as

$$p(y|\boldsymbol{x}) = p(y|\boldsymbol{x}, f),$$

where f is a particular function from some function (hypothesis) space $\mathcal{F}$. We can think of $\mathcal{F}$ as a set of all functions from a specified class, say for all $(w_0, w_1, \ldots, w_d) \in \mathbb{R}^{d+1}$ in the example above, but we can also extend the functional class beyond simple parameter variation to incorporate non-linear decision surfaces. We typically select one function, given the data—say the maximum likelihood or MAP solution

We could instead consider the distribution of $Y|\boldsymbol{x}$ over all plausible functions f. In a typical learning problem, we are given a data set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ and are asked to model $p(y|\boldsymbol{x})$. For this purpose, we will think of $\mathcal{D}$ as a realization of a random variable D and will assume that $\mathcal{D}$ was drawn according to the true underlying distribution $p(\boldsymbol{x}, y)$. Thus, our task is to express $p(y|\boldsymbol{x}, \mathcal{D})$. Using the sum and product rules, will rewrite our original task of estimating $p(y|\boldsymbol{x})$ as

$$\begin{aligned} p(y|\boldsymbol{x}, \mathcal{D}) &= \int_{\mathcal{F}} p(y|\boldsymbol{x}, f, \mathcal{D}) p(f|\boldsymbol{x}, \mathcal{D}) df \\ &= \int_{\mathcal{F}} p(y|\boldsymbol{x}, f) p(f|\boldsymbol{x}, \mathcal{D}) df. \end{aligned}$$

Here we used conditional independence between output Y and the data set D once a particular model f was selected based on $\mathcal{D}$; thus, $p(y|\boldsymbol{x}, f, \mathcal{D}) = p(y|\boldsymbol{x}, f)$. This equation, gives us a sense that the optimal decision can be made through a mixture of distributions $p(y|\boldsymbol{x}, f)$, where the weights are given as posterior densities $p(f|\boldsymbol{x}, \mathcal{D})$. In finite hypothesis spaces $\mathcal{F}$ we have that

$$p(y|\boldsymbol{x}, \mathcal{D}) = \sum_{f \in \mathcal{F}} p(y|\boldsymbol{x}, f) p(f|\boldsymbol{x}, \mathcal{D}),$$

and $p(f|\boldsymbol{x}, \mathcal{D})$ are posterior probabilities. We may further assume that $p(f|\boldsymbol{x}, \mathcal{D}) = p(f|\mathcal{D})$, in which case the weights can be precomputed based on the given data set $\mathcal{D}$. This leads to more efficient calculations of posterior probabilities.

In classification, we can rewrite our original MAP classifier as

$$f_{\text{MAP}}(\boldsymbol{x}, \mathcal{D}) = \arg\max_{y \in \mathcal{Y}} \left\{ p(y|\boldsymbol{x}, \mathcal{D}) \right\},$$

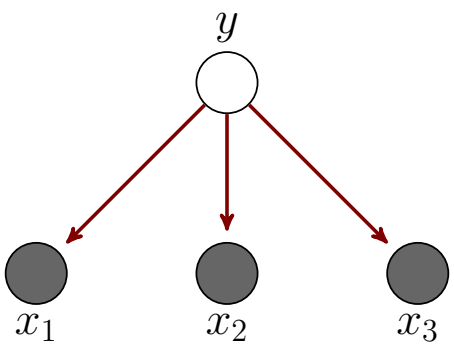

Figure 5.2: Naive Bayes graphical model, with three features.

which readily leads to the following formulation

$$\begin{aligned} f_{\text{MAP}}(\boldsymbol{x}, \mathcal{D}) &= \arg\max_{y \in \mathcal{Y}} \left\{ \int_{\mathcal{F}} p(y|\boldsymbol{x}, f, \mathcal{D}) p(f|\mathcal{D}) df \right\} \\ &= \arg\max_{y \in \mathcal{Y}} \left\{ \int_{\mathcal{F}} p(y|\boldsymbol{x}, f) p(\mathcal{D}|f) p(f) df \right\}. \end{aligned}$$

It can be shown that no classifier can outperform the *Bayes optimal classifier*. Interestingly, the Bayes optimal model also hints that a better prediction performance can be achieved by combining multiple models and averaging their outputs. This provides theoretical support for ensemble learning and methods such as bagging and boosting.

One problem in Bayes optimal classification is efficient calculation of $f_{\text{MAP}}(\boldsymbol{x}, \mathcal{D})$, given that the function (hypothesis) space $\mathcal{F}$ is generally uncountable. One approach to this is sampling of functions from $\mathcal{F}$ according to $p(f)$ and then calculating $p(f|\mathcal{D})$ or $p(\mathcal{D}|f)$. This can be computed until $p(y|\boldsymbol{x}, \mathcal{D})$ converges.

5.5 Naive Bayes approach

Naive Bayes is a generative approach to prediction and can be used for both classification and regression. As mentioned earlier, discriminative approaches aim to learn $p(y|\boldsymbol{x})$. Generative approaches, on the other hand, learn $p(\boldsymbol{x}, y) = p(\boldsymbol{x}|y)p(y)$. As one can imagine, this can be a more difficult undertaking, as we also need to learn the distribution over the features themselves. Naive Bayes idea is to simplify learning of this joint distribution by making a strong assumption that the features are conditionally independent given the target. This assumption is demonstrated by the graphical model in Figure 5.2. We will first discuss naive Bayes classifiers and subsequently turn our attention to regression.

5.5.1 Naive Bayes classification

Let $\mathcal{X}$ and $\mathcal{Y}$ be an input and output spaces, respectively, with $\mathcal{Y}$ being discrete. As with discriminative classifiers, the decision rule for labeling a data point is

$$\begin{aligned} \hat{y} &= \operatorname*{argmax}_{y \in \mathcal{Y}} p(y|\boldsymbol{x}) \\ &= \operatorname*{argmax}_{y \in \mathcal{Y}} \{p(\boldsymbol{x}|y)p(y)\} \end{aligned}$$

because $p(y|\boldsymbol{x}) \propto p(\boldsymbol{x}|y)p(y)$. For two classes, this corresponds to picking class 1 if $p(Y = 1|\boldsymbol{x}) \geq p(Y = 0|\boldsymbol{x})$ and class 0 otherwise. Under the naive Bayes assumption, the features are independent given the class label. Therefore, assuming d-dimensional inputs, we can write

$$p(\boldsymbol{x}|y) = \prod_{j=1}^{d} p(x_j|y). \tag{5.1}$$

We will now see how we can estimate probabilities $p(x_j|y)$ and $p(y)$, but will separately consider discrete and continuous features.

Naive Bayes classifier with discrete features

Let $\mathcal{D} = \{(\boldsymbol{x}_i, y\}_{i=1}^n$ be an input data set, where $\mathcal{X} = \{0, 1\}^d$ and $\mathcal{Y} = \{0, 1\}$. A suitable choice for these univariate probabilities is a Bernoulli distribution, since each X_j is binary, giving

$$p(x_j|Y = k) = \alpha_{j,k}^{x_j}(1 - \alpha_{j,k})^{1-x_j}.$$

The parameters for the Bernoulli distributions are $\alpha_{j,k} = p(X_j = 1|Y = k)$, with a different parameter $\alpha_{j,k}$ for each feature j and each class value k. We can easily learn this parameter from data $\mathcal{D}$ as

$$\alpha_{j,k} \stackrel{est}{=} \frac{\text{number of times } x_j = 1 \text{ for class } k}{\text{number of examples labeled as class } k}.$$

where $\stackrel{est}{=}$ reads as "is estimated as". Similarly, we can learn the prior $\alpha_k = p(Y = k)$ using

$$\alpha_k \stackrel{est}{=} \frac{\text{number of examples labeled as class } k}{\text{total number of examples}}.$$

The prediction $\hat{y}$ on a new point $\boldsymbol{x}$ is then

$$\begin{aligned}
\hat{y} &= \operatorname*{argmax}_{k \in \mathcal{Y}} p(Y = k|\boldsymbol{x}) \\
&= \operatorname*{argmax}_{k \in \mathcal{Y}} p(\boldsymbol{x}|Y = k)p(Y = k) \\
&= \operatorname*{argmax}_{k \in \mathcal{Y}} \prod_{j=1}^{d} p(x_j|Y = k)p(Y = k) \\
&= \operatorname*{argmax}_{k \in \mathcal{Y}} \prod_{j=1}^{d} \alpha_{j,k}\alpha_k
\end{aligned}$$

Exercise: The parameter estimation above is intuitive, and comes from similarly deriving the maximum likelihood solution. Assuming that you have n examples and the chosen distribution $p(x_j|y)$ is Bernoulli as described above, derive the maximum likelihood parameters $\alpha_{j,k}, \alpha_k$ for $j = 1, \ldots, d$ and $k = 0, 1$. To make things simpler, use the logarithm of the likelihood.

Notice that this approach could also be accomplished for more classes than just two. It can be similarly extended to feature alphabets of higher cardinality by exploiting multinomial distributions. Alternatively, a good practical fix is to replace each feature j with $|\mathcal{X}_j|$

binary features using one-hot representation. That is, if the original feature value comes from the alphabet $\{1, 2, 3\}$ and a particular example $\boldsymbol{x}_i$ has the value $x_{ij} = 2$, the one-hot representation would create three new binary features in the place of X_j for which feature 2 will be set to 1 and all others to 0.

The lack of sufficient data for learning in high-dimensional spaces can lead to some pathological situations in naive Bayes. For example, it can happen that at least one $\alpha_{j,k} = 0$ for every k, in which case the approximated class-conditional probabilities can be 0 regardless of the input. It is therefore recommended that all $\alpha_{j,k}$ be kept positive. A principled approach to it is to rely on Bayesian inference with suitably chosen hyperparameters in the priors. Such corrections of maximum likelihood estimates are often referred to as smoothing.

Naive Bayes classifier with continuous features

For continuous features, a Bernoulli distribution is no longer appropriate for $p(x_j|y)$ and we need to choose a different conditional distribution $p(\boldsymbol{x}|y)$. A common choice is a Gaussian distribution, now with a different mean and variance for each feature and class, $\mu_{j,k}, \sigma^2_{j,k}$

$$p(x_j|Y = k) = (2\pi\sigma^2_{j,k})^{-1/2} \exp\left(-\frac{(x_j - \mu_{j,k})^2}{2\sigma^2_{j,k}}\right).$$

Since Y is still discrete, we can approximate $p(y)$ using counts as before. The maximum likelihood mean and variance parameters correspond to the sample mean and sample variance for each given class separately. This involves computing the mean and variance of feature j across the examples labeled with class k; i.e.,

$$\mu_{j,k} \overset{est}{=} \frac{\sum_{i=1}^n 1(y_i = k)x_j}{\text{number of examples labeled as class } k}$$
$$\sigma^2_{j,k} \overset{est}{=} \frac{\sum_{i=1}^n 1(y_i = k)(x_j - \mu_{j,k})^2}{\text{number of examples labeled as class } k}.$$

Our choices are certainly not limited to the normal family. Other distributions can be used in this setting just as easily.

Exercise: Derive the maximum likelihood formulation for a Gaussian naive Bayes model, and check that the solution does in fact match the sample mean and variance for each feature and class separately, as above.

5.5.2 Naive Bayes regression

The naive Bayes approach for regression also relies on estimating the posterior density $p(y|\boldsymbol{x})$ as

$$p(y|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|y)p(y)}{\int p(\boldsymbol{x}|y)p(y)dy},$$

where $p(\boldsymbol{x}|y) = \prod_{j=1}^d p(x_j|y)$. Each conditional density $p(x_j|y)$ is now estimated using

$$p(x_j|y) = \frac{p(x_j, y)}{p(y)}.$$

This formulation now allows us to model each joint feature-target distribution $p(x_j, y)$ separately. A simple solution to this problem is to use two-dimensional Gaussian distributions,

but more sophisticated approaches, such as kernel density estimation, tend to work well. The case with discrete features now becomes equivalent to estimating $p(y|X_j = k)$ and $p(X_j = k)$, where k is one of the discrete feature values of X_j.