CHAPTER 1

Introduction to Statistical Machine Learning

We start with a gentle introduction to statistical machine learning. Readers familiar with machine learning may wish to skip directly to Section 2, where we introduce semi-supervised learning.

Example 1.1. You arrive at an extrasolar planet and are welcomed by its resident little green men. You observe the weight and height of 100 little green men around you, and plot the measurements in Figure 1.1. What can you learn from this data?

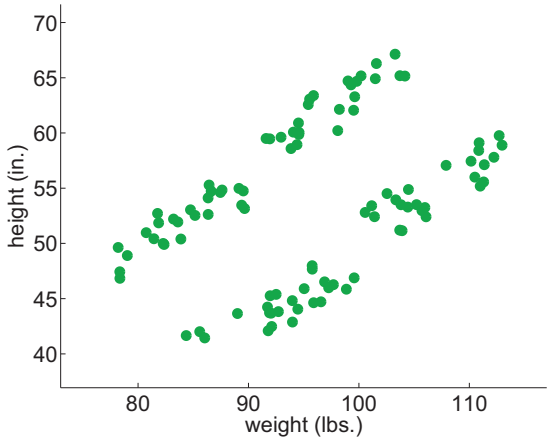


Figure 1.1: The weight and height of 100 little green men from the extrasolar planet. Each green dot is an instance, represented by two features: weight and height.

This is a typical example of a machine learning scenario (except the little green men part). We can perform several tasks using this data: group the little green men into subcommunities based on weight and/or height, identify individuals with extreme (possibly erroneous) weight or height values, try to predict one measurement based on the other, etc. Before exploring such machine learning tasks, let us begin with some definitions.

1.1 THE DATA

Definition 1.2. Instance. An *instance* $\mathbf{x}$ represents a specific object. The instance is often represented by a D-dimensional *feature vector* $\mathbf{x} = (x_1, \ldots, x_D) \in \mathbb{R}^D$, where each dimension is called a *feature*. The length D of the feature vector is known as the dimensionality of the feature vector.

The feature representation is an abstraction of the objects. It essentially ignores all other information not represented by the features. For example, two little green men with the same weight and height, but with different names, will be regarded as indistinguishable by our feature representation. Note we use boldface $\mathbf{x}$ to denote the whole instance, and x_d to denote the d-th feature of $\mathbf{x}$. In our example, an instance is a specific little green man; the feature vector consists of $D = 2$ features: x_1 is the weight, and x_2 is the height. Features can also take discrete values. When there are multiple instances, we will use x_{id} to denote the i-th instance's d-th feature.

Definition 1.3. Training Sample. A *training sample* is a collection of instances $\{\mathbf{x}_i\}_{i=1}^n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, which acts as the input to the learning process. We assume these instances are sampled independently from an underlying distribution $P(\mathbf{x})$, which is unknown to us. We denote this by $\{\mathbf{x}_i\}_{i=1}^n \overset{\text{i.i.d.}}{\sim} P(\mathbf{x})$, where i.i.d. stands for independent and identically distributed.

In our example, the training sample consists of $n = 100$ instances $\mathbf{x}_1, \ldots, \mathbf{x}_{100}$. A training sample is the "experience" given to a learning algorithm. What the algorithm can learn from it, however, varies. In this chapter, we introduce two basic learning paradigms: *unsupervised learning* and *supervised learning*.

1.2 UNSUPERVISED LEARNING

Definition 1.4. Unsupervised learning. Unsupervised learning algorithms work on a training sample with n instances $\{\mathbf{x}_i\}_{i=1}^n$. There is no teacher providing supervision as to how individual instances should be handled—this is the defining property of unsupervised learning. Common unsupervised learning tasks include:

- clustering, where the goal is to separate the n instances into groups;
- novelty detection, which identifies the few instances that are very different from the majority;
- dimensionality reduction, which aims to represent each instance with a lower dimensional feature vector while maintaining key characteristics of the training sample.

Among the unsupervised learning tasks, the one most relevant to this book is *clustering*, which we discuss in more detail.

Definition 1.5. Clustering. Clustering splits $\{\mathbf{x}_i\}_{i=1}^n$ into k clusters, such that instances in the same cluster are similar, and instances in different clusters are dissimilar. The number of clusters k may be specified by the user, or may be inferred from the training sample itself.

How many clusters do you find in the little green men data in Figure 1.1? Perhaps $k = 2$, $k = 4$, or more. Without further assumptions, either one is acceptable. Unlike in supervised learning (introduced in the next section), there is no teacher that tells us which instances should be in each cluster.

There are many clustering algorithms. We introduce a particularly simple one, *hierarchical agglomerative clustering*, to make unsupervised learning concrete.

Algorithm 1.6. Hierarchical Agglomerative Clustering.

Input*: a training sample* $\{\mathbf{x}_i\}_{i=1}^n$*; a distance function* $d()$.
1. Initially, place each instance in its own cluster (called a singleton cluster).
2. ***while*** *(number of clusters* > 1*)* ***do****:*
3. Find the closest cluster pair A, B*, i.e., they minimize* $d(A, B)$.
4. Merge A, B *to form a new cluster.*
Output*: a binary tree showing how clusters are gradually merged from singletons to a root cluster, which contains the whole training sample.*

This clustering algorithm is simple. The only thing unspecified is the distance function $d()$. If $\mathbf{x}_i, \mathbf{x}_j$ are two singleton clusters, one way to define $d(\mathbf{x}_i, \mathbf{x}_j)$ is the Euclidean distance between them:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\| = \sqrt{\sum_{s=1}^{D} (x_{is} - x_{js})^2}. \tag{1.1}$$

We also need to define the distance between two non-singleton clusters A, B. There are multiple possibilities: one can define it to be the distance between the closest pair of points in A and B, the distance between the farthest pair, or some average distance. For simplicity, we will use the first option, also known as *single linkage*:

$$d(A, B) = \min_{\mathbf{x} \in A, \mathbf{x}' \in B} d(\mathbf{x}, \mathbf{x}'). \tag{1.2}$$

It is not necessary to fully grow the tree until only one cluster remains: the clustering algorithm can be stopped at any point if $d()$ exceeds some threshold, or the number of clusters reaches a predetermined number k.

Figure 1.2 illustrates the results of hierarchical agglomerative clustering for $k = 2, 3, 4$, respectively. The clusters certainly look fine. But because there is no information on how each instance *should* be clustered, it can be difficult to objectively evaluate the result of clustering algorithms.

1.3 SUPERVISED LEARNING

Suppose you realize that your alien hosts have a gender: female or male (so they should not all be called little green men after all). You may now be interested in predicting the gender of a particular

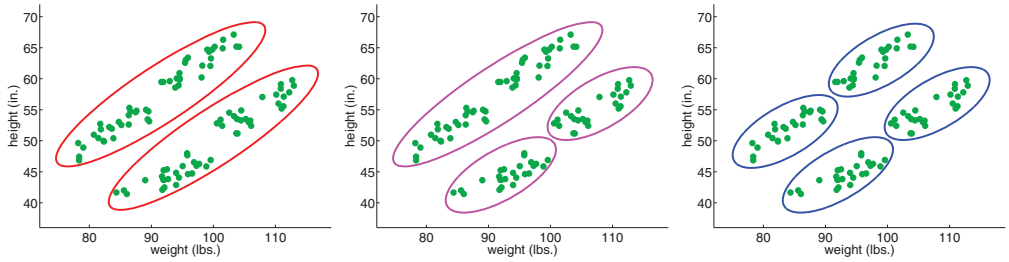


Figure 1.2: Hierarchical agglomerative clustering results for $k = 2, 3, 4$ on the 100 little green men data.

alien from his or her weight and height. Alternatively, you may want to predict whether an alien is a juvenile or an adult using weight and height. To explain how to approach these tasks, we need more definitions.

Definition 1.7. Label. A label y is the desired prediction on an instance $\mathbf{x}$.

Labels may come from a finite set of values, e.g., {female, male}. These distinct values are called *classes*. The classes are usually encoded by integer numbers, e.g., female $= -1$, male $= 1$, and thus $y \in \{-1, 1\}$. This particular encoding is often used for binary (two-class) labels, and the two classes are generically called the negative class and the positive class, respectively. For problems with more than two classes, a traditional encoding is $y \in \{1, \ldots, C\}$, where C is the number of classes. In general, such encoding does not imply structure in the classes. That is to say, the two classes encoded by $y = 1$ and $y = 2$ are not necessarily closer than the two classes $y = 1$ and $y = 3$. Labels may also take continuous values in $\mathbb{R}$. For example, one may attempt to predict the blood pressure of little green aliens based on their height and weight.

In supervised learning, the training sample consists of pairs, each containing an instance $\mathbf{x}$ and a label y: $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$. One can think of y as the label on $\mathbf{x}$ provided by a teacher, hence the name *supervised* learning. Such (instance, label) pairs are called *labeled data*, while instances alone without labels (as in unsupervised learning) are called *unlabeled data*. We are now ready to define supervised learning.

Definition 1.8. Supervised learning. Let the domain of instances be $\mathcal{X}$, and the domain of labels be $\mathcal{Y}$. Let $P(\mathbf{x}, y)$ be an (unknown) joint probability distribution on instances and labels $\mathcal{X} \times \mathcal{Y}$. Given a training sample $\{(\mathbf{x}_i, y_i)\}_{i=1}^n \overset{\text{i.i.d.}}{\sim} P(\mathbf{x}, y)$, supervised learning trains a function $f : \mathcal{X} \mapsto \mathcal{Y}$ in some function family $\mathcal{F}$, with the goal that $f(\mathbf{x})$ predicts the true label y on future data $\mathbf{x}$, where $(\mathbf{x}, y) \overset{\text{i.i.d.}}{\sim} P(\mathbf{x}, y)$ as well.

Depending on the domain of label y, supervised learning problems are further divided into *classification* and *regression*:

Definition 1.9. Classification. Classification is the supervised learning problem with discrete classes $\mathcal{Y}$. The function f is called a *classifier*.

Definition 1.10. Regression. Regression is the supervised learning problem with continuous $\mathcal{Y}$. The function f is called a *regression function*.

What exactly is a good f? The best f is by definition

$$f^* = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \, \mathbb{E}_{(\mathbf{x}, y) \sim P} \left[c(\mathbf{x}, y, f(\mathbf{x})) \right], \tag{1.3}$$

where argmin means "finding the f that minimizes the following quantity". $\mathbb{E}_{(\mathbf{x}, y) \sim P} [\cdot]$ is the expectation over random test data drawn from P. Readers not familiar with this notation may wish to consult Appendix A. $c(\cdot)$ is a *loss function* that determines the cost or impact of making a prediction $f(\mathbf{x})$ that is different from the true label y. Some typical loss functions will be discussed shortly. Note we limit our attention to some function family $\mathcal{F}$, mostly for computational reasons. If we remove this limitation and consider all possible functions, the resulting f^* is the *Bayes optimal predictor*, the best one can hope for on average. For the distribution P, this function will incur the lowest possible loss when making predictions. The quantity $\mathbb{E}_{(\mathbf{x}, y) \sim P}[c(\mathbf{x}, y, f^*(\mathbf{x}))]$ is known as the *Bayes error*. However, the Bayes optimal predictor may not be in $\mathcal{F}$ in general. Our goal is to find the $f \in \mathcal{F}$ that is as close to the Bayes optimal predictor as possible.

It is worth noting that the underlying distribution $P(\mathbf{x}, y)$ is unknown to us. Therefore, it is not possible to directly find f^*, or even to measure any predictor f's performance, for that matter. Here lies the fundamental difficulty of statistical machine learning: one has to *generalize* the prediction from a finite training sample to any unseen test data. This is known as *induction*.

To proceed, a seemingly reasonable approximation is to gauge f's performance using training sample error. That is, to replace the unknown expectation by the average over the training sample:

Definition 1.11. Training sample error. Given a training sample $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, the training sample error is

$$\frac{1}{n} \sum_{i=1}^{n} c(\mathbf{x}_i, y_i, f(\mathbf{x}_i)). \tag{1.4}$$

For classification, one commonly used loss function is the 0-1 loss $c(\mathbf{x}, y, f(\mathbf{x})) \equiv (f(\mathbf{x}_i) \neq y_i)$:

$$\frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i) \neq y_i), \tag{1.5}$$

where $f(\mathbf{x}) \neq y$ is 1 if f predicts a different class than y on x, and 0 otherwise. For regression, one commonly used loss function is the squared loss $c(\mathbf{x}, y, f(\mathbf{x})) \equiv (f(\mathbf{x}_i) - y_i)^2$:

$$\frac{1}{n}\sum_{i=1}^{n}(f(\mathbf{x}_i) - y_i)^2. \tag{1.6}$$

Other loss functions will be discussed as we encounter them later in the book.

It might be tempting to seek the f that minimizes training sample error. However, this strategy is flawed: such an f will tend to *overfit* the particular training sample. That is, it will likely fit itself to the statistical noise in the particular training sample. It will learn more than just the true relationship between $\mathcal{X}$ and $\mathcal{Y}$. Such an overfitted predictor will have small training sample error, but is likely to perform less well on future test data. A sub-area within machine learning called computational learning theory studies the issue of overfitting. It establishes rigorous connections between the training sample error and the true error, using a formal notion of complexity such as the Vapnik-Chervonenkis dimension or Rademacher complexity. We provide a concise discussion in Section 8.1. Informed by computational learning theory, one reasonable training strategy is to seek an f that "almost" minimizes the training sample error, while *regularizing* f so that it is not too complex in a certain sense. Interested readers can find the references in the bibliographical notes.

To estimate f's future performance, one can use a separate sample of labeled instances, called the *test sample*: $\{(\mathbf{x}_j, y_j)\}_{j=n+1}^{n+m} \overset{\text{i.i.d.}}{\sim} P(\mathbf{x}, y)$. A test sample is not used during training, and therefore provides a faithful (unbiased) estimation of future performance.

Definition 1.12. Test sample error. The corresponding test sample error for classification with 0-1 loss is

$$\frac{1}{m}\sum_{j=n+1}^{n+m}(f(\mathbf{x}_j) \neq y_j), \tag{1.7}$$

and for regression with squared loss is

$$\frac{1}{m}\sum_{j=n+1}^{n+m}(f(\mathbf{x}_j) - y_j)^2. \tag{1.8}$$

In the remainder of the book, we focus on classification due to its prevalence in semi-supervised learning research. Most ideas discussed also apply to regression, though.

As a concrete example of a supervised learning method, we now introduce a simple classification algorithm: k-nearest-neighbor (kNN).

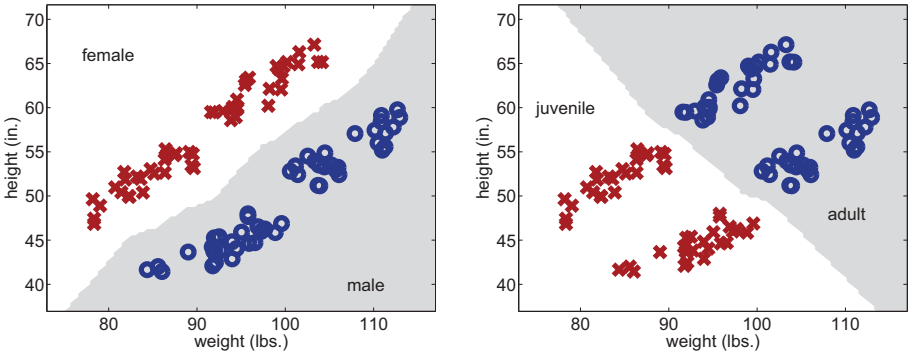
Algorithm 1.13. k-nearest-neighbor classifier.

Input: Training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$*; distance function* $d()$*;*
number of neighbors k*; test instance* $\mathbf{x}^*$

1. Find the k training instances $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_k}$ closest to $\mathbf{x}^*$ under distance $d()$.
2. Output y^* as the majority class of $y_{i_1}, \ldots, y_{i_k}$. Break ties randomly.

Being a D-dimensional feature vector, the test instance $\mathbf{x}^*$ can be viewed as a point in D-dimensional feature space. A classifier assigns a label to each point in the feature space. This divides the feature space into decision regions within which points have the same label. The boundary separating these regions is called the *decision boundary* induced by the classifier.

Example 1.14. Consider two classification tasks involving the little green aliens. In the first task in Figure 1.3(a), the task is gender classification from weight and height. The symbols are training data. Each training instance has a label: female (red cross) or male (blue circle). The decision regions from a 1NN classifier are shown as white and gray. In the second task in Figure 1.3(b), the task is age classification on the same sample of training instances. The training instances now have different labels: juvenile (red cross) or adult (blue circle). Again, the decision regions of 1NN are shown. Notice that, for the same training instances but different classification goals, the decision boundary can be quite different. Naturally, this is a property unique to supervised learning, since unsupervised learning does not use any particular set of labels at all.



(a) classification by gender (b) classification by age

Figure 1.3: Classify by gender or age from a training sample of 100 little green aliens, with 1-nearest-neighbor decision regions shown.

In this chapter, we introduced statistical machine learning as a foundation for the rest of the book. We presented the unsupervised and supervised learning settings, along with concrete examples of each. In the next chapter, we provide an overview of semi-supervised learning, which falls somewhere between these two. Each subsequent chapter will present specific families of semi-supervised learning algorithms.

BIBLIOGRAPHICAL NOTES

There are many excellent books written on statistical machine learning. For example, readers interested in the methodologies can consult the introductory textbook [131], and the comprehensive textbooks [19, 81]. For grounding of machine learning in classic statistics, see [184]. For computational learning theory, see [97, 176] for the Vapnik-Chervonenkis (VC) dimension and Probably Approximately Correct (PAC) learning framework, and Chapter 4 in [153] for an introduction to the Rademacher complexity. For a perspective from information theory, see [119]. For a perspective that views machine learning as an important part of artificial intelligence, see [147].