Chapter 3
Machine Learning Basics

Machine learning has changed many industries, including healthcare. The most fundamental concepts in machine learning include (1) *supervised learning* that has been used to develop risk prediction models for target diseases and (2) *unsupervised learning* that has been applied to discover unknown disease subtypes. Both supervised and unsupervised learning expect to model various patient features as demographic features, including age, gender and ethnicity, and past diagnosis features (e.g., ICD codes). The key difference is the presence of labels in supervised learning and the absence of labels in unsupervised learning. A label is a gold standard for a target of interest, such as a patient's readmission status for training a readmission predictive model.

As we will describe in later chapters, most of the deep learning successes are in supervised learning. In contrast, the potential for unsupervised learning is immense due to the availability of a large amount of unlabeled data. This chapter will present the predictive model pipeline, basic models for supervised and unsupervised learning, and various model evaluation metrics. Table 3.1 defines notations used in this chapter.

3.1 Predictive Modeling Pipeline

A predictive modeling pipeline is a process of building prediction models from observational data. And predictive modeling pipelines are common use cases for supervised learning. As shown in Fig. 3.1, such a pipeline is not a single algorithm but a sequence of computational steps involving the following steps:

1. We first define the **prediction target**. For example, we may want to predict a future diagnosis of heart failure. An appropriate prediction target should be important for the application and feasible to achieve, given the available data.

C. Xiao, J. Sun, *Introduction to Deep Learning for Healthcare*,
https://doi.org/10.1007/978-3-030-82184-5_3

Table 3.1 Notation table

Notation	Definition
$\boldsymbol{x} \in \mathbb{R}^M$	M-dimensional feature vector
$\boldsymbol{x}_+$	Index set of data points of positive class
$\boldsymbol{x}_-$	Index set of data points of negative class
$y_i \in \{1, 2, \ldots, K\}$	Label for data point i
$(\boldsymbol{x}_i, y_i)$	A data point for supervised learning
$\boldsymbol{x}_i$	A data point for unsupervised learning
N	Number of data points
$\boldsymbol{X} \in \mathbb{R}^{M \times N}$ $\boldsymbol{y} \in \mathbb{R}^N$	Feature matrix and label vector
$\boldsymbol{w}$	weight vectors

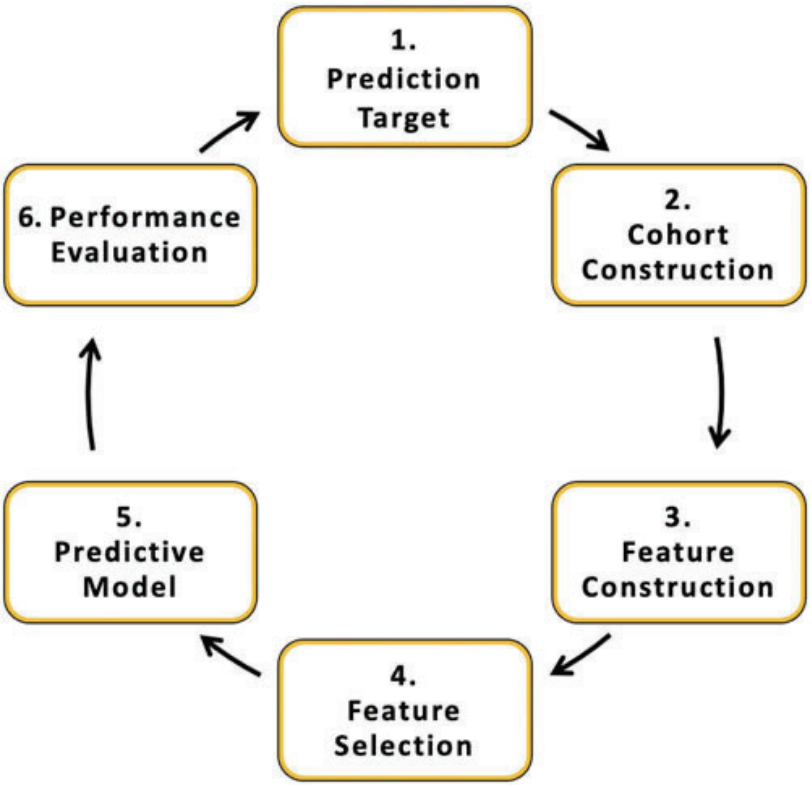

Fig. 3.1 Predictive modeling pipeline

2. We then need to **construct the patient cohort** for this study. For example, we may include all patients with an age greater than 45 for a heart failure study. There are many reasons why cohort construction is needed when building healthcare predictive models: (1) there might be the financial cost associated with acquiring the dataset based on the cohort; (2) we may want to build the model for a specific group of patients instead of a general population; (3) the full set of all patients may have various data quality issues.
3. Next, we will **construct all the features** from the data and **select those relevant features** for predicting the target. In a traditional machine learning pipeline, we

often have to consider both feature construction and selection steps. With the rise of deep learning, the features are often created and implicitly selected by the multiple layers of neural networks.
4. After that we can build the **predictive model** which can be either classification (i.e., discrete labels such as heart failure or not) and regression (i.e., continuous output such as length of stay).
5. Finally we need to **evaluate the model performance** and iterate.

3.2 Supervised Learning

We will start with the problem of disease classification as one major supervised learning task in healthcare applications: given a set of patients and their associated patient data, assign each patient with a discrete label $y \in \mathcal{Y}$, where $\mathcal{Y}$ is the set of possible diseases. This problem has many applications, from studying electroencephalography (EEG) time series for seizure detection to analyzing electronic health records (EHR) for predicting heart failure diagnosis. Supervised learning tasks such as disease classification are also a building block throughout many complex deep learning architectures, which we will discuss later.

Supervised learning expects an input of a set of N data points; each data point consists of m input features $\boldsymbol{x}$ (also known as variables or predictors) and a label $y \in \mathcal{Y}$ (also known as a response, a target or an outcome). Supervised learning aims to learn a mapping from features $\boldsymbol{x}$ to a label y based on the observed data points. If the labels are continuous (e.g., hospital cost), the supervised learning problem is called a *regression* problem. And if the labels are discrete variables (e.g., mortality status), the problem is called a *classification* problem. In this chapter, the label y are categorical values of K classes (i.e., $y \in \{1, 2, \ldots, K\}$).

3.2.1 Logistic Regression

Logistic regression is one popular binary classification model.[1] Logistic regression is actually the simplest neural network model, which is also known as the perceptron. Next, let us explain logistic regression with a healthcare example.

Suppose we want to predict the heart failure onset of patients based on their health-related features. In this example, the label $y = 1$ if the patient has heart failure and $y = 0$ if the patient does not have heart failure. Each patient has a M-dimensional feature vector $\boldsymbol{x}$ representing demographic features, various lab tests,

[1]Maybe a confusing name as logistic regression is for classification not for regression. But the naming choice will become meaningful after we explain the mathematical construction.

and other disease diagnoses. The classification task is to determine whether a patient will have heart failure based on this M-dimensional feature vector $\boldsymbol{x}$.

Mathematically logistic regression models the probability of heart failure onset $y = 1$ given input features $\boldsymbol{x}$, denoted by $P(y = 1|\boldsymbol{x})$. Then the classification is performed by comparing $P(y = 1|\boldsymbol{x})$ with a threshold (e.g., 0.5). If $P(y = 1|\boldsymbol{x})$ is greater than the threshold, we predict the patient will have heart failure; otherwise, the patient will not.

One building block of logistic regression is the log-odds or logit function. The odds are the quantity that measures the relative probability of label presence and label absence as

$$\frac{P(y = 1|\boldsymbol{x})}{1 - P(y = 1|\boldsymbol{x})}.$$

The lower the odds, the lower probability of the given label. Sometimes we prefer to use log-odds (natural logarithm transformation of odds), also known as the logit function.

$$logit(\boldsymbol{x}) = \log\left(\frac{P(y = 1|\boldsymbol{x})}{1 - P(y = 1|\boldsymbol{x})}\right).$$

Now instead of modeling probability of heart failure label given input feature $P(y = 1|\boldsymbol{x})$ directly, it is easier to model its logit function as a linear regression over $\boldsymbol{x}$:

$$\log\left(\frac{P(y = 1|\boldsymbol{x})}{1 - P(y = 1|\boldsymbol{x})}\right) = \boldsymbol{w}^T\boldsymbol{x} + b \tag{3.1}$$

where $\boldsymbol{w}$ is the weight vector, b is the offset variable. Equation (3.1) is why logistic regression is named logistic regression.

After taking exponential to both sides and some simple transformation, we will have the following formula.

$$P(y = 1|\boldsymbol{x}) = \frac{e^{\boldsymbol{w}^T\boldsymbol{x}+b}}{1 + e^{\boldsymbol{w}^T\boldsymbol{x}+b}} \tag{3.2}$$

With the formulation in Eq. (3.2), the logistic regression will always output values between 0 and 1, which is desirable as a probability estimate.

Let us denote $P(y = 1|\boldsymbol{x})$ as $P(\boldsymbol{x})$ for brevity. Now learning the logistic regression model means to estimate the parameters $\boldsymbol{w}$ and b on the training data. We often use maximum likelihood estimation (MLE) to find the parameters. The idea is to estimate $\boldsymbol{w}$ and b so that the prediction $\hat{P}(\boldsymbol{x}_i)$ to data point i in the training data is as close as possible to actual observed values (in this case either 0 or 1). Let $\boldsymbol{x}_+$ be the set of indices for data points that belong to the positive class (i.e., with heart failure), and $\boldsymbol{x}_-$ one for data points that belong to the negative class (i.e., without heart failure), the likelihood function used in the MLE is given by Eq. (3.3).

$$\mathcal{L}(\mathbf{w}, b) = \prod_{a_+ \in \mathbf{x}_+} P(\mathbf{x}_{a_+}) \prod_{a_- \in \mathbf{x}_-} (1 - P(\mathbf{x}_{a_-})) \tag{3.3}$$

If we take the logarithm to the MLE, we will get the following formula for log-likelihood in Eq. (3.4).

$$\log(\mathcal{L}(\mathbf{w}, b)) = \sum_{i=1}^{N} [y_i \log P(\boldsymbol{x}_i) + (1 - y_i) \log(1 - P(\boldsymbol{x}_i))] \tag{3.4}$$

Note that since either y_i or $1 - y_i$ is zero, only one of two probability terms (either $\log P(\boldsymbol{x}_i)$ or $\log(1 - P(\boldsymbol{x}_i))$) will be added.

Multiplying a negative sign to have a minimization problem, what we have now is the negative log-likelihood, also known as (binary) cross-entropy loss.

$$J(\mathbf{w}, b) = -\sum_{i=1}^{N} [y_i \log P(\boldsymbol{x}_i) + (1 - y_i) \log(1 - P(\boldsymbol{x}_i))] \tag{3.5}$$

To maximize the log-likelihood is the same as to minimize the cross-entropy loss. We can use the gradient descent method to find the optimal $\boldsymbol{w}$ and b.

3.2.2 Softmax Regression

We sometimes want to classify data points into more than two classes. For example, given brain image data from patients that are suspected of having Alzheimer's disease (AD), the diagnoses outcomes include (1) normal, (2) mild cognitive impairment (MCI), and (3) AD. In that case, we will use multinomial logistic regression, also called softmax regression to model this problem.

Assuming we have K classes, the goal is to estimate the probability of the class label taking on each of the K possible categories $P(y = k|\boldsymbol{x})$ for $k = 1, \cdots, K$. Thus, we will output a K-dimensional vector representing the estimated probabilities for all K classes. The probability that data point i is in class a can be modeled by Eq. (3.6).

$$P(y_i = a|\boldsymbol{x}_i) = \frac{e^{\boldsymbol{w}_a^T \boldsymbol{x}_i + b_a}}{\sum_{k=1}^{K} e^{\boldsymbol{w}_k^T \boldsymbol{x}_i + b_k}} \tag{3.6}$$

where $\boldsymbol{w}_a$ is the weight for a-th class, $\boldsymbol{x}_i$ is the feature vector for data point i, $\boldsymbol{w}_a$ and b_a are the weight vector and the offset for class a, respectively. To learn parameters for softmax regression, we often optimize the following average cross-entropy loss over all N training data points:

$$J(w) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} I(y_i = k)\log(P(y_i = k|\boldsymbol{x}_i))$$

where K is the number of label classes (e.g., 3 classes in AD classification), $I(y_i = k)$ is binary indicator (0 or 1) if k is the class for data point i. And $P(y_i = k|\boldsymbol{x}_i)$ is the predicted probability that data point i is of class k.

3.2.3 Gradient Descent

Gradient descent (GD) is an iterative learning approach to find the optimal parameters based on data. For example, for softmax regression parameter estimation, we can use GD by computing the derivatives

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w})$$

and update the weights in the opposite direction of the gradient like the following rule

$$\boldsymbol{w} \coloneqq \boldsymbol{w} - \eta \nabla_{w_k} J(\boldsymbol{w})$$

for each class $k \in \{1, \cdots, K\}$ and η is the learning rate, which is an important hyperparameter that needs to be adjusted. The gradient computation and weight update are iteratively performed until some stopping criterion is met (e.g., maximum number of iterations reached). Here ∇ is a differentiation operator which transforms a function $J(\boldsymbol{w})$ into its gradient vector along each feature dimension x_i. For example, $\boldsymbol{x} = [x_1, x_2, x_3]$, then the gradient vector is

$$\nabla J(\boldsymbol{w}) = \left\langle \frac{\partial J(\boldsymbol{w})}{\partial x_1}, \frac{\partial J(\boldsymbol{w})}{\partial x_2}, \frac{\partial J(\boldsymbol{w})}{\partial x_3} \right\rangle$$

3.2.4 Stochastic and Minibatch Gradient Descent

The gradient descent is a method to optimize an objective function $g(\boldsymbol{\theta})$ parameterized by model parameters $\boldsymbol{\theta} \in R^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta})$ with respect to the parameters. The full gradient can be very expensive to compute on a large data set because it has to process all data points. Several gradient descent variants reduce the computational cost, e.g., stochastic gradient descent (SGD) and mini-batch gradient descent.

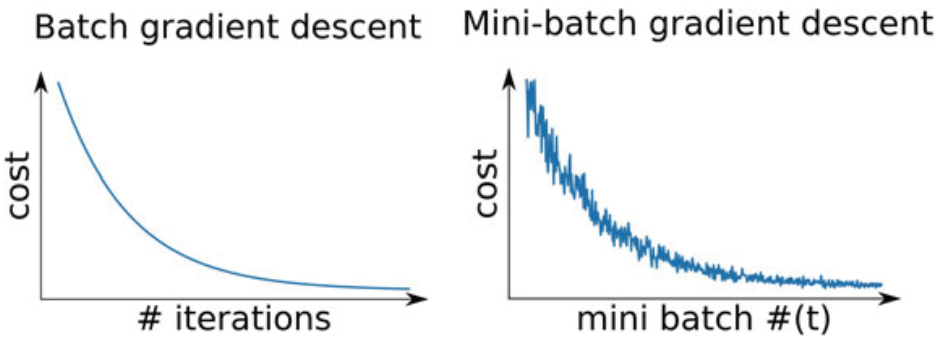

Fig. 3.2 Objective function changes for batch gradient descent and mini-batch gradient descent

The SGD performs parameter updating for every single data point in the training set. Given data point $\boldsymbol{x}_i$ with label y_i, the SGD does the following update:

$$\theta = \theta - \eta \nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}; < \boldsymbol{x}_i, y_i >) \tag{3.7}$$

where η is the learning rate and $g(\boldsymbol{\theta}; < \boldsymbol{x}_i, y_i >)$ is the objective function evaluated on one data point $< \boldsymbol{x}_i, y_i >$. By updating one data point at a time, SGD is computationally more efficient. However, since SGD updates based on one data point at a time, it can have a much higher variance that causes the objective function to fluctuate. Such behaviors can cause SGD to deviate from the true optimum. There are several ways to alleviate this issue. For example, we can slowly decrease the learning rate as it empirically shows SGD would have similar convergence behavior as batch gradient descent (Fig. 3.2).

The mini-batch approach inherits benefits from both GD and SGD. It computes the gradient over small batches of training data points. The mini-batch n is a hyperparameter, and the mini-batch gradient descent does the following update.

$$\theta = \theta - \eta \nabla_{\boldsymbol{\theta}} g(\theta; < \boldsymbol{x}_i, y_i >, \ldots, < \boldsymbol{x}_{i+n-1}, y_{i+n-1} >) \tag{3.8}$$

where $< \boldsymbol{x}_i, y_i >, \ldots, < \boldsymbol{x}_{i+n-1}, y_{i+n-1} >$ are the n data points in a batch. Here the gradient is iteratively computed using batches of data points. Via such a mini-batch update, we reduce the variance of the parameter updates and solve the unstable convergence issue seen by SGD.

3.3 Unsupervised Learning

In many healthcare applications, labels are not available. In such cases, we resort to unsupervised learning models. Unsupervised learning models are not used for classifying (or predicting) towards a known label y. Rather we discover patterns or clusters about the input data points $\boldsymbol{x}$. Next, we briefly introduce some popular unsupervised learning methods.

3.3.1 Principal Component Analysis

Suppose we want to study N data points of M features represented by a matrix $\boldsymbol{X} \in \mathbb{R}^{N \times M}$. The number of features M can be large in many healthcare datasets. For example, there are 68,000 ICD-9 codes where each code can be a separate binary feature. Principal component analysis (PCA) can be applied to reduce the data dimensionality from M to a much lower dimension R. More specifically, PCA is a linear transformation:

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{W} \tag{3.9}$$

where $\boldsymbol{X}$ is the original data matrix, $\boldsymbol{Y} \in \mathbb{R}^{N \times R}$ is the low-dimensional representation after PCA, $\boldsymbol{W} \in \mathbb{R}^{M \times R}$ is the orthogonal projection matrix. The objective of PCA is to minimize the reconstruction error:

$$\min_{\boldsymbol{W}} \|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{W}\boldsymbol{W}^\top\|^2$$

where $\boldsymbol{X}\boldsymbol{W}\boldsymbol{W}^\top = \boldsymbol{Y}\boldsymbol{W}^\top$ is the reconstruction matrix. The solution of PCA relates to another matrix factorization named singular value decomposition (SVD).

$$\boldsymbol{X} \approx \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{W}^\top$$

where $\boldsymbol{U} \in \mathbb{R}^{M \times R}$ and $\boldsymbol{W} \in \mathbb{R}^{N \times R}$ are orthogonal matrices[2] that contain left and right singular vectors and $\boldsymbol{\Sigma} \in \mathbb{R}^{R \times R}$. Connecting to PCA, if $\boldsymbol{X}$ is the high-dimensional data matrix, the low-dimensional representation $\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{\Sigma} = \boldsymbol{X}\boldsymbol{W}$.

In practice, PCA can be used as a feature extraction method for generating features from high-dimensional data such as neuroimaging data. For example, neuroimaging data such as Magnetic Resonance Imaging (MRI) or functional MRI includes many voxels, whose high dimensionality brings many challenges for diagnostic classification tasks. If we consider brain voxels as a raw feature, we can apply PCA to generate low-dimensional features to support downstream classification tasks. For instance, [86] showed that PCA features combining with a support vector machine (SVM) classifier provided good discriminative power in early diagnosis of Alzheimer's disease.

To summarize, as an unsupervised learning method, PCA provides low-dimensional linear representation to approximate the original high-dimensional features. In fact, PCA can be achieved by a neural network via autoencoders with linear activation. In later chapters, we can see more details about how neural networks expand the idea of PCA to low-dimensional nonlinear embedding using methods such as autoencoder.

[2]This means $\boldsymbol{U}\top\boldsymbol{U} = \boldsymbol{I}$ where $\boldsymbol{I}$ is the identity matrix.

3.3.2 Clustering

Besides dimensionality reduction, clustering is another major topic in unsupervised learning. Clustering methods aim to find homogeneous groups (or clusters) from a dataset, such that similar points are within the same cluster but dissimilar points in different clusters. For example, researchers have applied a clustering algorithm on EHR data to find disease subtypes of type II diabetes patient group into three clusters [97].

One popular clustering method is K-means, which tries to group data into K clusters where users specify the number K. The clustering assignments are achieved by minimizing the sum of distances (e.g., Euclidean distances) between data points and the corresponding cluster centroid (or the mean vectors). The K-means method is described in the following procedure:

Algorithm 1 The K-means algorithm

Input: (1) data points $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N$; (2) number of clusters K
Until convergence
DO
    Initialize K centers.
    WHILE not converged, **DO**
        Assign each $\boldsymbol{x}_i$ to the closest center $\arg\min_{k \in 1,2,\ldots,K} \|\boldsymbol{x}_i - \boldsymbol{\mu}_k\|$;
        Compute $\mathcal{L}$ for this observation $(\boldsymbol{x}, t)$;
        Update K centers $\boldsymbol{\mu}_k := \frac{1}{|S_i|} \sum_{i \in S_i} \boldsymbol{x}_i$.
RETURN
        Disjoint clusters $S_1, S_2, \ldots, S_K$.

Following the aforementioned procedure, we finish clustering all samples. Figure 3.3 provides a visualization of the iterative clustering procedure, where we apply k-means clustering over a set of points with cluster number $K = 2$.

3.4 Evaluation Metrics

In this section, we will introduce some common performance measures and evaluation strategies in machine learning.

3.4.1 Evaluation Metrics for Regression Tasks

The mean squared error (MSE) is the most basic performance metric for regression models. The formulation of MSE is given in Eq. (3.10).

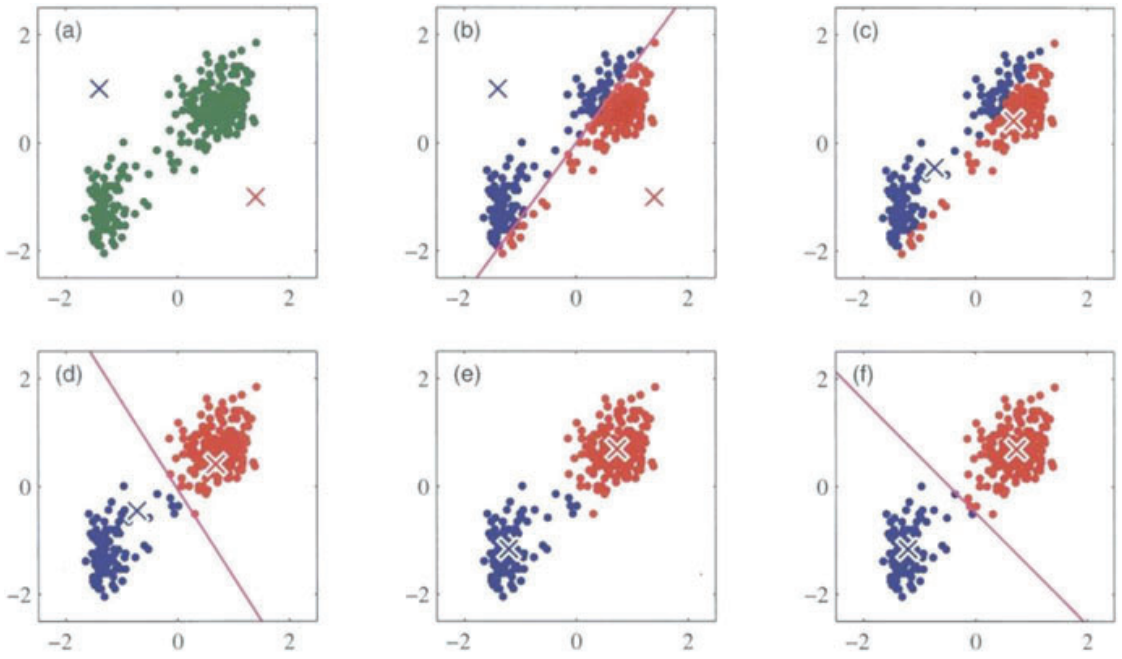

Fig. 3.3 k-Means clustering over a set of points with $K = 2$. From left to right, top to bottom, we firstly initialize two cluster centers with a blue cross and a red cross. After a few iterations, all blue (red) points are assigned to the blue (red) cluster, completing the K-means clustering procedure

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f(\boldsymbol{x}_i) - y_i)^2 \tag{3.10}$$

where $f(\boldsymbol{x}_i)$ is the predicted value for the i-th data point. Small MSE means the prediction is close to the true observation on average. Thus the model has a good fit. We can take the squared root of MSE to obtain another popular metric called root mean squared error (RMSE):

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(f(\boldsymbol{x}_i) - y_i)^2} \tag{3.11}$$

RMSE and MSE are commonly used in neural network parameter tuning. For example, the authors in [150] built a feedforward neural network model to find prognostic ischemic heart disease patterns from magnetocardiography (MCG) data. In training the model, RMSE was calculated as the evaluation metric to help to choose the model parameters, such as the number of nodes in the hidden layer and the number of learning epochs (see Fig. 3.4). Hyperparameters leading to the lowest RMSE was then chosen in the final model.

Another measure for regression problem is the coefficient of determination (also called as R^2) that measures the correlation between the predicted values $\{f(\boldsymbol{x}_i)\}$ and actual observations $\{y_i\}$. The R^2 is computed using Eq. (3.11).

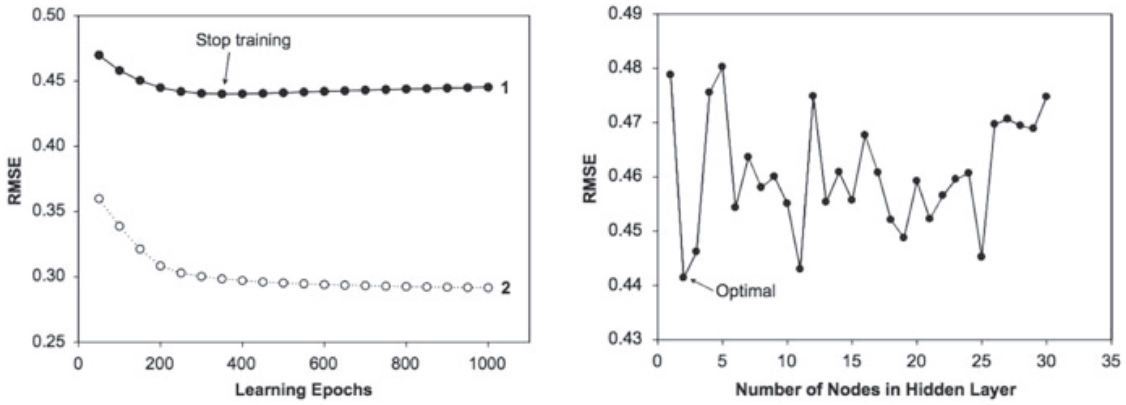

Fig. 3.4 In [150], to train the neural network model, RMSE was used as the evaluation metric in parameter tuning, including the number of learning epochs and the number of nodes in the hidden layer. Parameters exhibiting the lowest RMSE were chosen for the final model

$$R^2 = 1 - \frac{\sum_{i=1}^{N} (f(\boldsymbol{x}_i) - y_i)^2}{\sum_{i=1}^{N} (\bar{y} - y_i)^2} \tag{3.12}$$

where $\bar{y} = \frac{1}{N} \sum_i^N y_i$ is the sample mean of the target observation y_i.

The R^2 measures the "squared correlation" between the observation y_i and the estimation $f(\boldsymbol{x}_i)$. If R^2 is close to 1, then the model's estimations closely mirror true observed outcomes. If R^2 is close to 0 (or even negative), it means the estimation is far from the outcomes. Note that R^2 can become negative, in which case the model fit is worse than predicting the simple average of y_i regardless of the value of $\boldsymbol{x}_i$.

3.4.2 Evaluation Metrics for Classification Tasks

For binary classification, the label y_i can be either 1 or 0. For example, $y_i = 1$ indicates patient i is a heart failure patient (*case*) and $y_i = 0$ indicates a patient without heart failure (*control*). There are many performance metrics for classification models. For binary classification, prediction scores can be real values (e.g., probability risk scores between 0 and 1) or binary values.

Binary Prediction for Classification

If the predictions are binary values, we can construct a 2-by-2 confusion matrix to quantify all the possibilities between predictions and labels. In particular, we count the following four numbers: the number of case patients that are correctly

Table 3.2 Confusion matrix and performance metrics for classification

Total	Actual cases	Actual controls	Accuracy = (TP+TN)/Total
Predicted cases	True positive (TP)	False positive (FP)	Precision
Predicted controls	False negative (FN)	True negative (TN)	= TP/(TP+FP)
	Recall=Sensitivity =TP/(TP+FN)	Specificity=1-FPR =TN/(FP+TN)	False Positive Rate FPR=FP/(FP+TN)

predicted as cases is the true positive (TP); the number of case patients that are wrongly predicted as controls is the false negative (FN); the number of control patients that are correctly predicted as controls is the true negative (TN); and the number of control patients that are wrongly predicted as cases is the false positive (FP) (Table 3.2).

A few important performance metrics can be derived from the confusion matrix, including accuracy, precision—also known as positive predictive value (PPV), recall—also known as sensitivity, false positive rate, specificity, and F1 score:

Accuracy is the fraction of correct predictions over the total population, or formally:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

However, accuracy does not differentiate true positives (TP) or true negatives (TN). If there are many more controls (e.g., patients without the disease) than cases (patients with the disease), the high accuracy can be trivially achieved by classifying everyone as controls (or negatives). Other metrics address this class imbalance challenge indirectly, such as precision and recall, by focusing on the positive class.

Precision or positive predictive value (PPV) is the fraction of correct case predictions over all case predictions:

$$precision = \frac{TP}{TP + FP}. \tag{3.13}$$

While recall, also known as sensitivity and true positive rate (TPR), is the fraction of cases that are correctly predicted as cases

$$recall = \frac{TP}{TP + FN}. \tag{3.14}$$

Since precision and recall are often a trade-off, the F1 score is a popular measure that combines them by treating false positives and false negatives as equally important. More specifically, the F1 score is defined as the harmonic mean of precision and recall, given by the following formula:

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}. \tag{3.15}$$

A more detailed explanation about the F1 score can be found at [130].

Specificity is also called the true negative rate, which measures the proportion of controls (negative samples) that are correctly predicted as controls:

$$specificity = \frac{TN}{FP + TN}$$

A related measure is the false positive rate (FPR), which is 1-specificity.

Real-Value Prediction for Classification

Most classifiers output a soft prediction score x between 0 and 1 instead of binary values. In those cases, one often has to define a classification threshold θ. That is, the classifiers output 1 if $x \geq \theta$, otherwise output 0. However, the right classification threshold value is often unknown. To avoid choosing values of θ, one can evaluate the average performance across different θ.

We will introduce the **area under the Receiver Operating Characteristic curve**, called ROC-AUC or AUROC. Before describing ROC-AUC, we need to understand the Receiver Operating Characteristic curve (ROC), which has the true positive rate (TPR) or recall or sensitivity as the y-axis and the false positive rate (FPR) or 1-specificity as the x-axis. The ROC-AUC is commonly used in model comparison and can be interpreted as the probability that the classifier will assign a higher score to a randomly chosen positive example than a randomly chosen negative example. A model with higher ROC-AUC is considered a better model. The advantage of ROC-AUC over accuracy is that it does not require choosing a classification threshold. In this manner, ROC-AUC is more robust against class imbalance.

A similar metric is an **area under the precision-recall curve**, called PR-AUC or AUPR. The precision-recall (PR) curve uses recall as x-axis and precision as y-axis. The area under the precision-recall curve has a very similar interpretation as the ROC-AUC but has a different visual representation of the curves. Looking at PR curves can expose differences between algorithms that are not apparent in ROC curves. For example, Fig. 3.5 shows performance comparison using ROC and PR curves. The figure is from [167] where the authors built machine learning models to predict adverse drug reactions using drugs' molecular structure data. In the performance comparison, results indicated the proposed model (denoted as "lda-dummy" in both figures) that incorporates medical ontology achieved the best performance in both ROC-AUC and PR-AUC.

The goal in ROC space is to be in the upper-left-hand corner, and when we look at the ROC curves in Fig. 3.5 the baseline model "lasso" appears to be fairly close to the best model "lda-dummy". In PR space, the goal is to be in the upper-right-hand

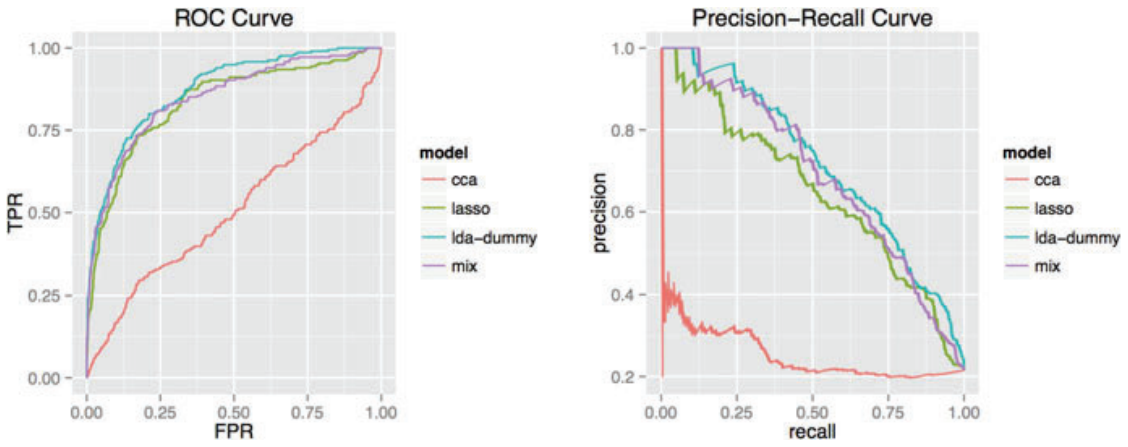

Fig. 3.5 In [167] where the authors built machine learning models to predict adverse drug reactions using drugs' molecular structure data. In the performance comparison, results indicated the model that incorporate medical ontology achieved the best performance in both ROC-AUC and PR-AUC

corner, and the PR curves in Fig. 3.5 show that there is still a significant gap between the performance of "lasso" and "lda-dummy".

Similar effects have been observed and discussed in [35], where the authors provided the following insights: "this difference exists because in this domain the number of negative examples greatly exceeds the number of positives examples". Consequently, a large change in the number of false positives can lead to a small change in the false positive rate used in ROC analysis. Precision, on the other hand, by comparing false positives to true positives rather than true negatives, captures the effect of a large number of negative examples on the algorithm's performance". In healthcare applications, since we often have many negative samples (e.g., controls), PR curve, and PR-AUC can be more meaningful to ROC-AUC.

Multi-Class Classification

Beyond binary classification, such as heart failure prediction, there are many scenarios we have multiple classes such as disease prediction, where each disease is a separate class. Thus each class becomes a separate binary classification task, which leads to a separate set of performance metrics such as precision and recall. When evaluating the multi-class classification's overall performance, people often have to average those performance metrics from different classes. There are two general approaches for computing average performance: **macro-average** or **micro-average**.

Suppose we have K classes, let TP_k, FP_k, TN_k, FN_k denote the number of data points that belong to true positive prediction, false positive prediction, true negative prediction, and false negative prediction in class k respectively ($k \in \{1, \cdots, K\}$). For each class k, we can compute aforementioned metrics such as F1 score using (3.15), denoted as $F1_k(TP_k, FP_k, TN_k, FN_k)$ since F1 score is a

function of the four values in the confusion matrix. Then the macro-F1 is a simple averaging across all K classes given by Eq. (3.16).

$$\text{macro-F1} = \frac{1}{K}\sum_{k=1}^{K} F1_k(TP_k, FP_k, TN_k, FN_k). \tag{3.16}$$

Micro-averaged metrics are computed differently. We will first sum up the counts of true positives, false positives, true negative, and false negatives of the model for different classes to construct a new confusion matrix where we have

$$\begin{aligned} TP &= \frac{1}{K}\sum_{k=1}^{K} TP_k, \quad FP = \frac{1}{K}\sum_{k=1}^{K} FP_k \\ TN &= \frac{1}{K}\sum_{k=1}^{K} TN_k, \quad FN = \frac{1}{K}\sum_{k=1}^{K} FN_k \end{aligned} \tag{3.17}$$

Then we can use Eqs. (3.13) and (3.14) to calculate micro-precision and micro-recall. Then by plugging in both measures into Eq. (3.15) we can find micro-F1, which is also a function of the new measures produced in Eq. (3.17):

$$\text{micro-F1} = F1(TP, FP, TN, FN).$$

We can follow the same procedure to compute other micro-averaged and macro-averaged measures. In general, macro-averaging gives each class equal weight, whereas micro-averaging gives each data point equal weight. As a result, the micro-averaged metric is a measure of effectiveness in the large classes. To observe performances on small classes, macro-averaged metrics will be needed.

3.4.3 Evaluation Metrics for Clustering Tasks

A clustering algorithm usually aims at achieving high intra-cluster similarity (i.e., similarity within a group) and low inter-cluster similarity (i.e., similarity across groups). Here the similarity is usually represented by distance measures such as Euclidean distance and cosine distance.

One popular metric for clustering tasks is **silhouette coefficient**, which measures how similar each data point is to its own cluster compared to other clusters. The silhouette coefficient is computed as follows. Let $d_0(i)$ be the average distance between a data point i and all other data within the same cluster, let $d_1(i)$ be the lowest average distance of data point i to all data points in any other cluster where i does not belong to, silhouette coefficient is given by Eq. (3.18).

$$s(i) = \frac{d_1(i) - d_0(i)}{\max\{d_1(i), d_0(i)\}} \tag{3.18}$$

The range of $s(i)$ is $[-1, 1]$. A silhouette coefficient close to 1 indicates that the data point is appropriately clustered. And a silhouette coefficient close to -1 means the data point is wrongly clustered. Then the overall clustering performance can be the average silhouette coefficient of all points.

If ground truth clustering assignments are known, we can also use other metrics such as **rand index**, **mutual information** and **normalized mutual information** to evaluate the cluster results. Given C the clustering assignments, and C^* the ground-truth clustering assignments, we can define the following metrics to evaluate the clustering assignment C:

- **Rand index (RI)** defines as $\frac{a+b}{n(n-1)/2}$ where a is the number of data point pairs that belong to the same cluster in C and C^*, b is the number of data point pairs that belong to different clusters in C and C^*, and n is the number of data points. The rand index is a score between 0 and 1, where 0 indicates C and C^* do not agree on any clustering assignments, and 1 indicates C and C^* agree completely.
- **Mutual information (MI)** is another way to measure clustering quality. Mutual information is a concept from information theory, which measures the mutual dependence of two random variables.

  $$MI(C, C^*) = \sum_{x \in C} \sum_{y \in C^*} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

  where x is a particular cluster with clustering assignment C and y is a particular ground-truth cluster in C^*, and $p(x, y)$ computes the probability of a data point in both cluster x and y.
- **Normalized mutual information (NMI)** is a normalized version of MI between 0 (no mutual information) and 1 (perfect correlation).

  $$NMI(C, C^*) = \frac{MI(C, C^*)}{\sqrt{H(C)H(C^*)}}$$

  where $MI(C, C^*)$ is the mutual information, $H(C)$ and $H(C^*)$ are the entropy measure. For example, $H(C) = \sum_{x \in C} p(x) \log p(x)$ where x is a cluster in C and $p(x)$ is the proportion of the cluster x.

3.4.4 Evaluation Strategy

A cross-validation strategy is commonly used to evaluate the performance of regression and classification models. The general idea is to split the entire dataset into training and testing sets iteratively. We then use the testing set to evaluate the

model's performance constructed from the training set. A common practice is to have most data used as the training set while a small portion is used as the test set (e.g., 90-10 split).

A popular variant is called K-fold cross-validation. We randomly split the entire data set into K non-overlapping partitions or folds of equal size in that strategy. When performing the split, one should ensure a non-overlapping partition to avoid information leaking. For example, to predict heart failure risk on patients where multiple visits of the same patient can occur, it is important to partition the patients' data so that no patients are shared across partitions. Each time we use onefold as the test set and the remaining folds as the training set. We learn a model using the training data and compute the model's performance measures on the testing data. We iterate K times until all folds have been used as a test set once. We then calculate the average performance measures across the K folds, which becomes the overall model performance. Another well-known strategy is called leave-one-out cross-validation (LOOCV) is, in fact, a special form of k-fold cross-validation, where each partition contains only one data point. LOOCV is typically only used for small datasets due to its expensive computation cost.

In deep learning, the data sets become huge, and the model training and parameter tuning become expensive. It becomes computationally too expensive to perform cross-validation, mainly due to model hyperparameters tuning. Instead, the evaluation strategy is shifted toward a single random split of the large data set into three partitions: training, validation, and test. A typical setup can be 80% training, 10% validation, and 10% test. First, we iteratively train the models with different hyperparameters (e.g., number of layers and number of neurons) on the training set and check the model performance on the validation set to decide the best hyperparameters setting. Then we use the best hyperparameters setting to train another model on the combination of training and validation set and check the model performance on the test set. The final performance on the test set will be used as the estimate of the true model performance.

3.5 Exercises

1. If you have a classification problem on 500 10-dimensional patient records, what algorithms would you try first? What algorithms would you try last?
2. If you have to cluster a large patient dataset (e.g., one billion data points), what algorithms would you use? what steps would you try to speed up the process?
3. What are the steps in a clinical predictive modeling pipeline?
4. How do you know if a prediction target is possible?
5. Which of the following are standard/good practice for building clinical predictive models?

   (a) Cross-validation are most commonly used for evaluating deep learning models.

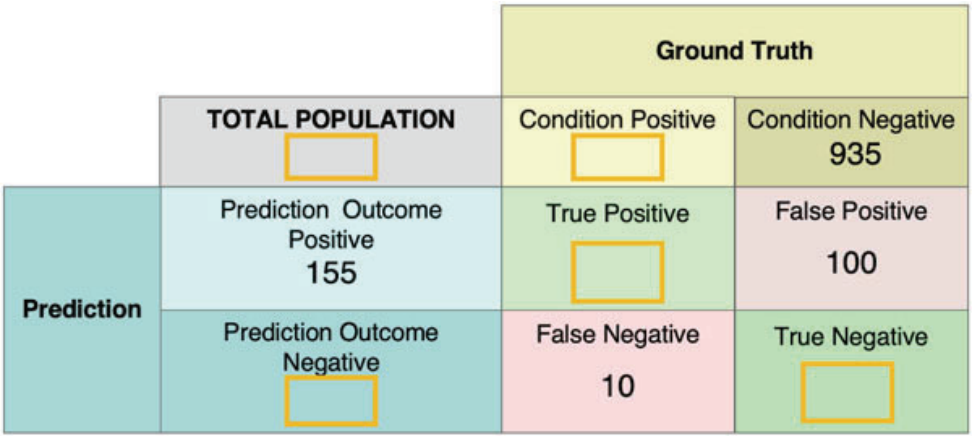

Fig. 3.6 Classification evaluation exercise

(b) For training deep learning models, it is important to keep validation and test sets large.
(c) Validation and Test sets can be small but should contain realistic samples with high-quality labels.
(d) Training data can be large and flexible, even with potentially noisy data.

6. What is the time complexity of K-means algorithm given n is the number of points, k is the number of clusters, d is the dimensionality of each point, and i is the number of clustering iterations?
7. Calculate the following statistics: Total Population, Condition Positive, True Positive, Prediction Outcome Negative, True Negative (Fig. 3.6).
8. Continue with the previous question, calculate True Positive Rate, False Positive Rate, False Negative Rate, True Negative Rate.