TRAINING A MODEL (FOR SUPERVISED LEARNING)

Holdout method

In case of supervised learning, a model is trained using the labelled input data. However, how can we understand the performance of the model? The test data may not be available immediately. Also, the label value of the test data is not known. That is the reason why a part of the input data is held back (that is how the name holdout originates) for evaluation of the model. This subset of the input data is used as the test data for evaluating the performance of a trained model. In general 70%–80% of the input data (which is obviously labelled) is used for model training. The remaining 20%–30% is used as test data for validation of the performance of the model. However, a different proportion of dividing the input data into training and test data is also acceptable. To make sure that the data in both the buckets are similar in nature, the division is done randomly. Random numbers are used to assign data items to the partitions. This method of partitioning the input data into two parts – training and test data.

which is by holding back a part of the input data for validating the trained model is known as holdout method

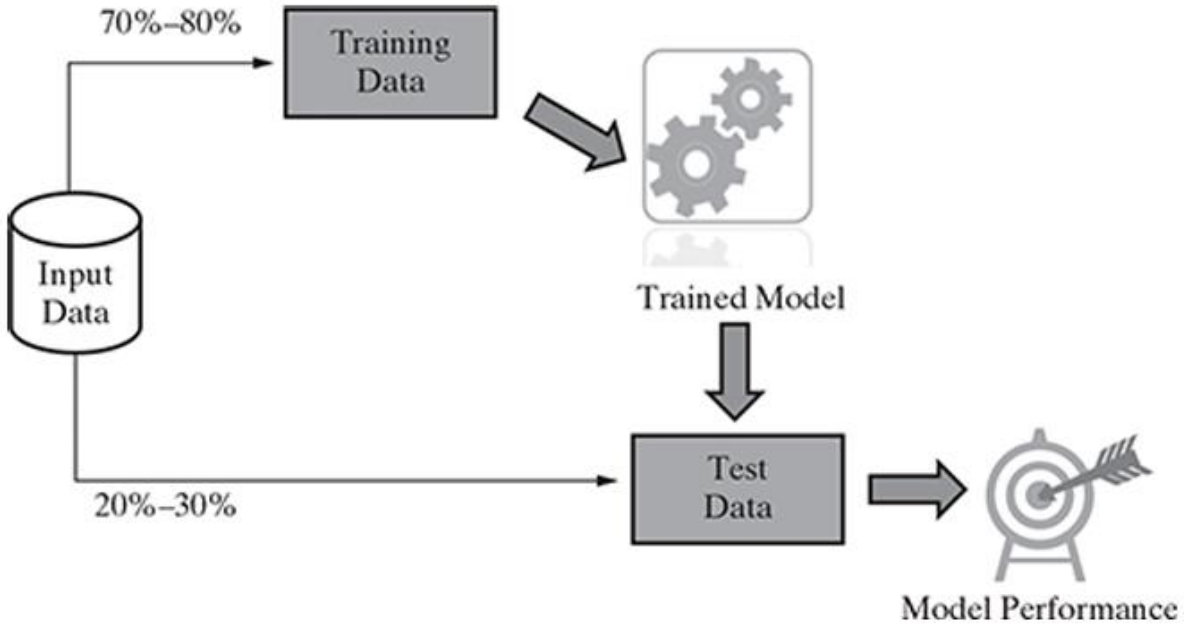

FIG. 3.1 Holdout method

Once the model is trained using the training data, the labels of the test data are predicted using the model's target function. Then the predicted value is compared with the actual value of the label. This is possible because the test data is a part of the input data with known labels. The performance of the model is in general measured by the accuracy of prediction of the label value,

In certain cases, the input data is partitioned into three portions – a training and a test data, and a third validation data. The validation data is used in place of test data, for measuring the model performance. It is used in iterations and to refine the model in each iteration. The test data is used only for once, after the model is refined and finalized, to measure and report the final performance of the model as a reference for future learning efforts.

K-fold Cross-validation method

Holdout method employing stratified random sampling approach still heads into issues in certain specific situations. Especially, the smaller data sets may have the challenge to divide the data of some of the classes proportionally amongst training and test data sets. A special variant of holdout method, called repeated holdout, is sometimes employed to ensure the randomness of the composed data sets. In repeated holdout, several random holdouts are used to measure the model performance. In the end, the average of all performances is taken. As multiple holdouts have been drawn, the training and test data (and also validation data, in case it is drawn) are more likely to contain representative data from all classes and resemble the original input data closely. This process of repeated holdout is the basis of k-fold cross validation technique. In k-fold cross-validation, the data set is divided into k-completely distinct or non-overlapping random partitions called folds. Figure 3.2 depicts an overall approach for k-fold cross-validation.

The value of 'k' in k-fold cross-validation can be set to any number. However, there are two approaches which are extremely popular:

1. 10-fold cross-validation (10-fold CV)

2. Leave-one-out cross-validation (LOOCV)

10-fold cross-validation is by far the most popular approach. In this approach, for each of the 10-folds, each comprising of approximately 10% of the data, one of the folds is used as the test data for validating model performance trained based on the remaining 9 folds (or 90% of the data). This is repeated 10 times, once for each of the 10 folds being used as the test data and the remaining folds as the training data.The average performance across all folds is being reported. Figure 3.3 depicts the detailed approach of selecting the 'k' folds in k-fold cross-validation. As can be observed in the figure, each of the circles resembles a

record in the input data set whereas the different colors indicate the different classes that the records belong to.The entire data set is broken into ‘k’ folds – out of which one fold is selected in each iteration as the test data set. The fold selected as test data set in each of the ‘k’ iterations is different. Also, note that though in figure 3.3 the circles resemble the records in the input data set, the contiguous circles represented as folds do not mean that they are subsequent records in the data set. This is more a virtual representation and not a physical representation. As already mentioned, the records in a fold are drawn by using random sampling technique.

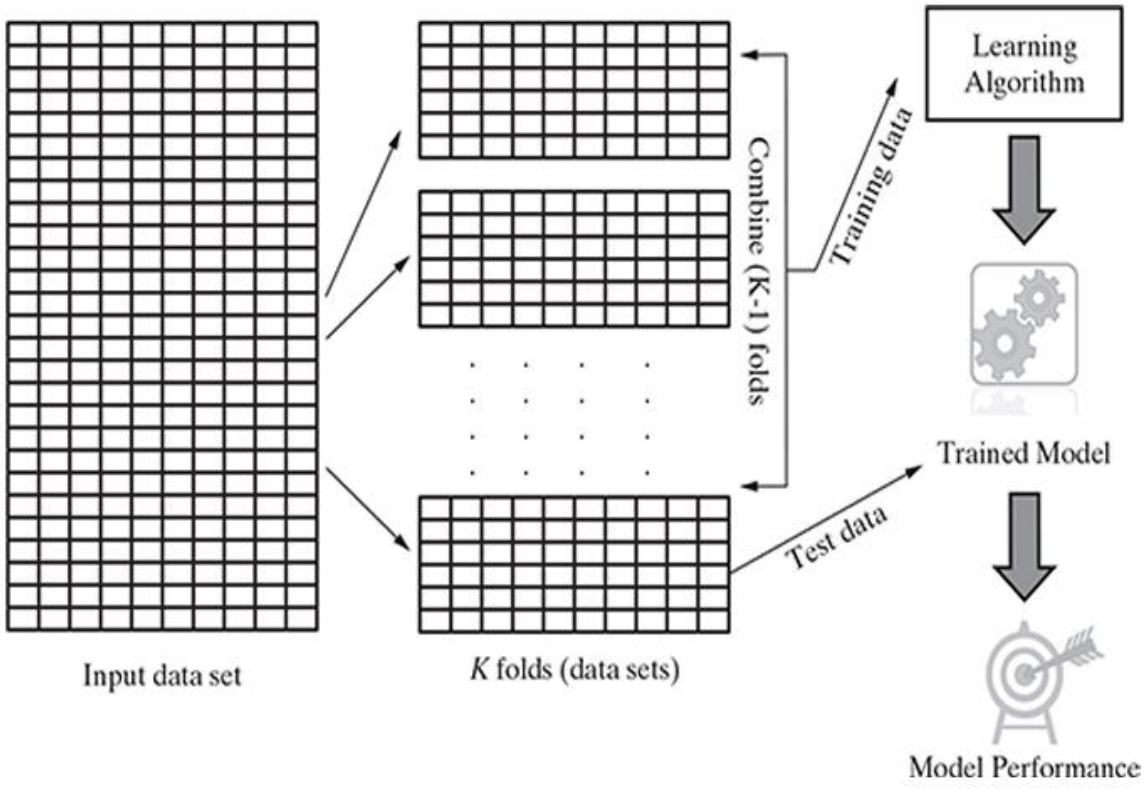

FIG. 3.2 Overall approach for *K*-fold cross-validation

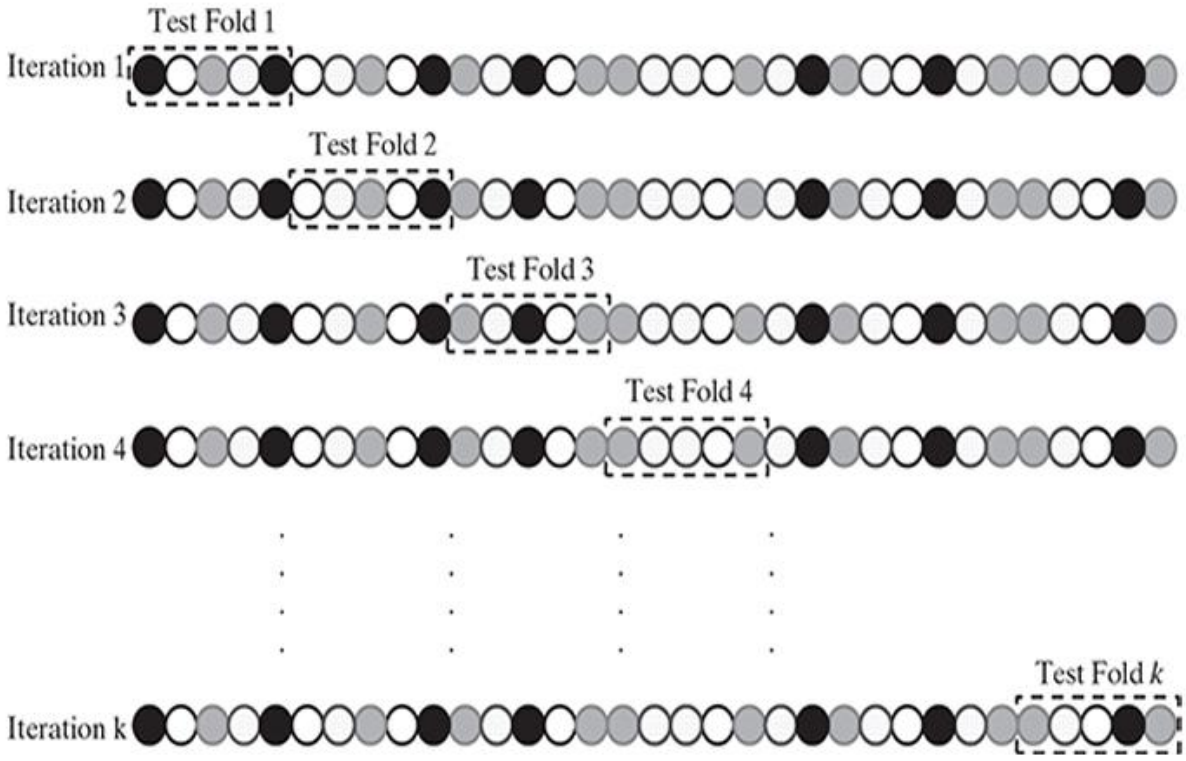

FIG. 3.3 Detailed approach for fold selection

Leave-one-out cross-validation (LOOCV) is an extreme case of k-fold cross-validation using one record or data instance at a time as a test data. This is done to maximize the count of data used to train the model. It is obvious that the number of iterations for which it has to be run is equal to the total number of data in the input data set. Hence, obviously, it is computationally very expensive and not used much in practice.

Bootstrap sampling

Bootstrap sampling or simply bootstrapping is a popular way to identify training and test data sets from the input data set. It uses the technique of Simple Random Sampling with Replacement (SRSWR), which is a well-known technique in sampling theory for drawing random samples. We have seen earlier that k-fold cross-validation divides the data into separate partitions – say 10 partitions in case of 10-fold cross validation. Then it uses data instances from partition as test data and the remaining partitions as training data. Unlike this approach adopted in case of k-fold cross- validation, bootstrapping randomly picks data instances from the input data set, with the possibility of the same data instance to be picked multiple times. This essentially means that from the input data set having 'n' data instances, bootstrapping can create one or more training data sets having 'n' data instances, some of

the data instances being repeated multiple times. Figure 3.4 briefly presents the approach followed in bootstrap sampling.

This technique is particularly useful in case of input data sets of small size, i.e. having very less number of data instances.

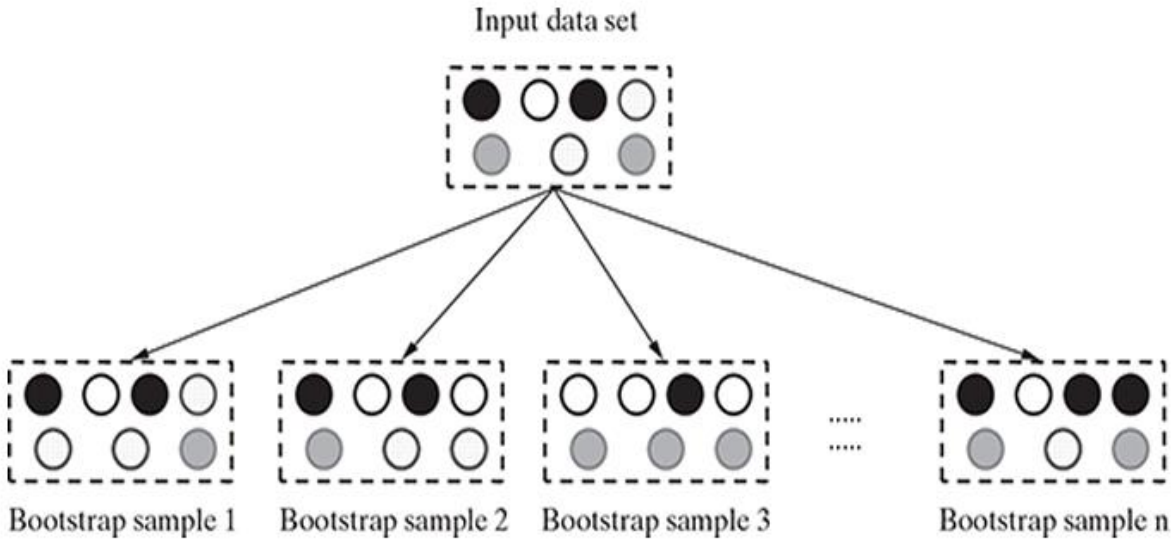

FIG. 3.4 Bootstrap sampling

CROSS-VALIDATION	BOOTSTRAPPING
It is a special variant of holdout method, called repeated holdout. Hence uses stratified random sampling approach (without replacement). Data set is divided into 'k' random partitions, with each partition containing approximately $\frac{n}{k}$ number of unique data elements, where 'n' is the total number of data elements and 'k' is the total number of folds.	It uses the technique of Simple Random Sampling with Replacement (SRSWR). So the same data instance may be picked up multiple times in a sample.
The number of possible training/test data samples that can be drawn using this technique is finite.	In this technique, since elements can be repeated in the sample, possible number of training/test data samples is unlimited.

Lazy vs. Eager learner

Eager learning follows the general principles of machine learning – it tries to construct a generalized, input-independent target function during the model training phase. It follows the typical steps of machine learning, i.e. abstraction and generalization and comes up zswith a trained model at the end of the learning phase. Hence, when the test data comes in for classification, the eager learner is ready with the model and doesn't need to refer back to the training data. Eager learners take more time in the learning phase than the lazy learners. Some of the algorithms which adopt eager learning approach include Decision Tree, Support Vector Machine, Neural Network, etc.

Lazy learning, on the other hand, completely skips the abstraction and generalization processes, as explained in context of a typical machine learning process. In that respect, strictly speaking, lazy learner doesn't 'learn' anything. It uses the training data in exact, and uses the knowledge to classify the unlabelled test data. Since lazy learning uses training data as-is, it is also known as rote learning (i.e. memorization technique based on repetition). Due to its heavy dependency on the given training data instance, it is also known as instance learning. They are also called non-parametric learning. Lazy learners take very little time in training because not much of training actually happens. However, it takes quite some time in classification as for each tuple of test data, a comparison-based assignment of label happens. One of the most popular algorithm for lazy learning is k-nearest neighbor.