Regression Metrics

You learned about regression metrics a few chapters ago. As a quick recap, the Pearson R^2 and RMSE (root-mean-squared error) are good defaults.

We only briefly covered the mathematical definition of R^2 previously, but will delve into it more now. Let x_i represent predictions and y_i represent labels. Let $\bar{x}$ and $\bar{y}$ represent the mean of the predicted values and the labels, respectively. Then the Pearson R (note the lack of square) is

$$R = \frac{\Sigma_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\Sigma_{i=1}^{N}(x_i - \bar{x})^2}\sqrt{\Sigma_{i=1}^{N}(y_i - \bar{y})^2}}$$

This equation can be rewritten as

$$R = \frac{\text{cov}(x, y)}{\sigma(x)\sigma(y)}$$

where cov represents the covariance and σ represents the standard deviation. Intuitively, the Pearson R measures the joint fluctuations of the predictions and labels from their means normalized by their respective ranges of fluctuations. If predictions and labels differ, these fluctuations will happen at different points and will tend to cancel, making R^2 smaller. If predictions and labels tend to agree, the fluctuations will happen together and make R^2 larger. We note that R^2 is limited to a range between 0 and 1.

The RMSE measures the absolute quantity of the error between the predictions and the true quantities. It stands for root-mean-squared error, which is roughly analogous to the absolute value of the error between the true quantity and the predicted quantity. Mathematically, the RMSE is defined as follows (using the same notation as before):

$$\text{RMSE} = \sqrt{\frac{\Sigma_{i=1}^{N}(x_i - y_i)^2}{N}}$$

Hyperparameter Optimization Algorithms

As we mentioned earlier in the chapter, hyperparameter optimization methods are learning algorithms for finding values of the hyperparameters that optimize the chosen metric on the validation set. In general, this objective function cannot be differentiated, so any optimization method must by necessity be a black box. In this section, we will show you some simple black-box learning algorithms for choosing

hyperparameter values. We will use the Tox21 dataset from Chapter 4 as a case study to demonstrate these black-box optimization methods. The Tox21 dataset is small enough to make experimentation easy, but complex enough that hyperparameter optimization isn't trivial.

We note before setting off that none of these black-box algorithms works perfectly. As you will soon see, in practice, much human input is required to optimize hyperparameters.

Can't Hyperparameter Optimization Be Automated?

One of the long-running dreams of machine learning has been to automate the process of selecting model hyperparameters. Projects such as the "automated statistician" and others have sought to remove some of the drudgery from the hyperparameter selection process and make model construction more easily available to non-experts. However, in practice, there has typically been a steep cost in performance for the added convenience.

In recent years, there has been a surge of work focused on improving the algorithmic foundations of model tuning. Gaussian processes, evolutionary algorithms, and reinforcement learning have all been used to learn model hyperparameters and architectures with very limited human input. Recent work has shown that with large amounts of computing power, these algorithms can exceed expert performance in model tuning! But the overhead is severe, with dozens to hundreds of times greater computational power required.

For now, automatic model tuning is still not practical. All algorithms we cover in this section require significant manual tuning However, as hardware quality improves, we anticipate that hyperparameter learning will become increasingly automated. In the near term, we recommend strongly that all practitioners master the intricacies of hyperparameter tuning. A strong ability to hyperparameter tune is the skill that separates the expert from the novice.

Setting Up a Baseline

The first step in hyperparameter tuning is finding a *baseline*. A baseline is performance achievable by a robust (non–deep learning usually) algorithm. In general, random forests are a superb choice for setting baselines. As shown in Figure 5-3, random forests are an ensemble method that train many decision tree models on subsets of the input data and input features. These individual trees then vote on the outcome.