Python机器学习基础教程

Stephen CUI

February 17, 2023

Chapter 1

模型评估与改进

为了评估我们的监督模型，我们使用 `train_test_split` 函数将数据集划分为训练集和测试集，在训练集上调用 fit 方法来构建模型，并且在测试集上用 score 方法来评估这个模型——对于分类问题而言，就是计算正确分类的样本所占的比例。

请记住，之所以将数据划分为训练集和测试集，是因为我们想要度量模型对前所未见的新数据的泛化性能。我们对模型在训练集上的拟合效果不感兴趣，而是想知道模型对于训练过程中没有见过的数据的预测能力。

本章我们将从两个方面进行模型评估。我们首先介绍交叉验证，然后讨论评估分类和回归性能的方法，其中前者是一种更可靠的评估泛化性能的方法，后者是在默认度量（score方法给出的精度和 R^2）之外的方法。

我们还将讨论网格搜索，这是一种调节监督模型参数以获得最佳泛化性能的有效方法。

1.1 交叉验证

交叉验证（cross-validation）是一种评估泛化性能的统计学方法，它比单次划分训练集和测试集的方法更加稳定、全面。在交叉验证中，数据被多次划分，并且需要训练多个模型。最常用的交叉验证是 k 折交叉验证（k-fold cross-validation），其中 k 是由用户指定的数字，通常取 5 或 10。在执行 5 折交叉验证时，首先将数据划分为（大致）相等的 5 部分，每一部分叫作折（fold）。接下来训练一系列模型。使用第 1 折作为测试集、其他折（2 5）作为训练集来训练第一个模型。利用 2 5 折中的数据来构建模型，然后在 1 折上评估精度。之后构建另一个模型，这次使用 2 折作为测试集，1、3、4、5 折中的数据作为训练集。利用 3、4、5 折作为测试集继续重复这一过程。对于将数据划分为训练集和测试集的这 5 次划分，每一次都要计算精度。最后我们得到了 5 个精度值。

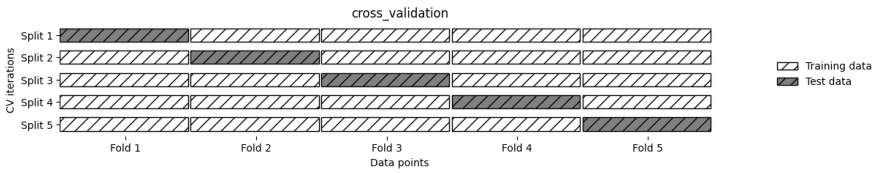

Figure 1.1: Data splitting in five-fold cross-validation

1.1.1 scikit-learn中的交叉验证

scikit-learn 是利用 `model_selection` 模块中的 `cross_val_score` 函数来实现交叉验证的。`cross_val_score` 函数的参数是我们想要评估的模型、训练数据与真实标签。默认情况下，`cross_val_score` 执行 5 折交叉验证，返回 5 个精度值。可以通过修改 cv 参数来改变折数。

总结交叉验证精度的一种常用方法是计算平均值。

1.1.2 交叉验证的优点

1. 减少实验的随机性：如果使用交叉验证，每个样例都会刚好在测试集中出现一次：每个样例位于一个折中，而每个折都在测试集中出现一次。因此，模型需要对数据集中所有样本的泛化能力都很好，才能让所有的交叉验证得分（及其平均值）都很高。
2. 提供模型对训练集选择的敏感性信息：假设对于 iris 数据集，我们观察到精度在 90% 到 100% 之间。这是一个不小的范围，它告诉我们将模型应用于新数据时在最坏情况和最好情况下的可能表现。
3. 对数据的使用更加高效：在使用 5 折交叉验证时，在每次迭代中我们可以使用 4/5（80%）的数据来拟合模型。在使用 10 折交叉验证时，我们可以使用 9/10（90%）的数据来拟合模型。更多的数据通常可以得到更为精确的模型。

交叉验证的主要缺点是增加了计算成本。现在我们要训练 k 个模型而不是单个模型，所以交叉验证的速度要比数据的单次划分大约慢 k 倍。

> 重要的是要记住，交叉验证不是一种构建可应用于新数据的模型的方法。交叉验证不会返回一个模型。在调用 `cross_val_score` 时，内部会构建多个模型，但交叉验证的目的只是评估给定算法在特定数据集上训练后的泛化性能好坏。

1.1.3 分层k折交叉验证和其他策略

将数据集划分为 k 折时，从数据的前 k 分之一开始划分，这可能并不总是一个好主意。对于iris数据集，数据的前三分之一是类别 0，中间三分之一是类别 1，最后三分之一是类别 2。想象一下在这个数据集上进行 3 折交叉验证。第 1 折将只包含类别 0，所以在数据的第一次划分中，测试集将只包含类别 0，而训练集只包含类别 1 和 2。由于在 3 次划分中训练集和测试集中的类别都不相同，因此这个数据集上的 3 折交叉验证精度为 0。这没什么帮助，因为我们在 iris 上可以得到比 0% 好得多的精度。

由于简单的 k 折策略在这里失效了，所以 scikit-learn 在分类问题中不使用这种策略，而是使用**分层 k 折交叉验证**（stratified k-fold cross-validation）。在分层交叉验证中，我们划分数据，使每个折中类别之间的比例与整个数据集中的比例相同，如 Figure 1.2 所示。

举个例子，如果 90% 的样本属于类别 A 而 10% 的样本属于类别 B，那么分层交叉验证可以确保，在每个折中 90% 的样本属于类别 A 而 10% 的样本属于类别 B。

对于回归问题，scikit-learn 默认使用标准 k 折交叉验证。也可以尝试让每个折表示回归目标的不同取值，但这并不是一种常用的策略，也会让大多数用户感到意外。

对交叉验证的更多控制

可以利用 cv 参数来调节 `cross_val_score` 所使用的折数。但 scikit-learn允许提供一个**交叉验证分离器**（cross-validation splitter）作为 cv 参数，来对数据划分过程进行更精细的控制。对于大多数使用场景而言，回归问题默认的 k 折交叉验证与分类问题的分层 k 折交叉验证的表现都很好，但有些情况下你

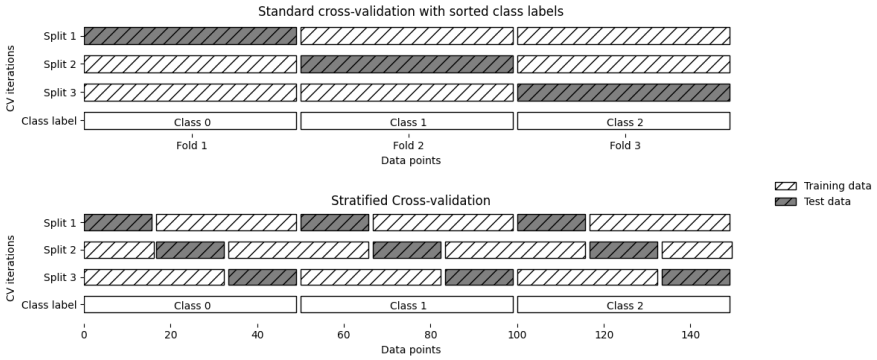

Figure 1.2: Comparison of standard cross-validation and stratified cross-validation

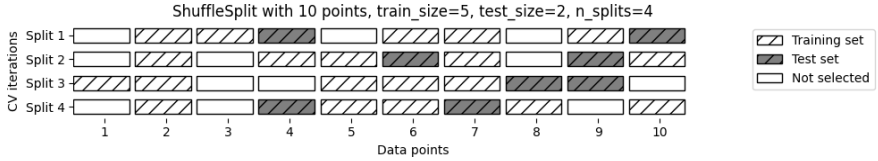

Figure 1.3: Shuffle-split cross-validation

可能希望使用不同的策略。比如说，我们想要在一个分类数据集上使用标准 k 折交叉验证来重现别人的结果。可以将 kfold 分离器对象作为 cv 参数传入 `cross_val_score`。

通过这种方法，我们可以验证，在 iris 数据集上使用 3 折交叉验证（不分层）确实是一个非常糟糕的主意。解决这个问题的另一种方法是将数据打乱来代替分层，以打乱样本按标签的排序。可以通过将 KFold 的 shuffle 参数设为 True 来实现这一点。如果我们将数据打乱，那么还需要固定 `random_state` 以获得可重复的打乱结果。

留一法交叉验证

另一种常用的交叉验证方法是**留一法**（leave-one-out）。你可以将留一法交叉验证看作是每折只包含单个样本的 k 折交叉验证。对于每次划分，你选择单个数据点作为测试集。这种方法可能非常耗时，特别是对于大型数据集来说，但在小型数据集上有时可以给出更好的估计结果。

打乱划分交叉验证

一种非常灵活的交叉验证策略是打乱划分交叉验证（shuffle-split cross-validation）。在打乱划分交叉验证中，每次划分为训练集取样 `train_size` 个点，为测试集取样 `test_size` 个（不相交的）点。将这一划分方法重复 `n_iter` 次。Figure 1.3 显示的是对包含 10 个点的数据集运行 4 次迭代划分，每次的训练集包含 5 个点，测试集包含 2 个点（你可以将 `train_size` 和 `test_size` 设为整数来表示这两个集合的绝对大小，也可以设为浮点数来表示占整个数据集的比例）。

打乱划分交叉验证可以在训练集和测试集大小之外独立控制迭代次数，这有时是很有帮助的。它还允许在每次迭代中仅使用部分数据，这可以通过设置 `train_size` 与 `test_size` 之和不等于 1 来实现。用这种方法对数据进行二次采样可能对大型数据上的试验很有用。

ShuffleSplit 还有一种分层的形式，其名称为 StratifiedShuffleSplit，它可以为分类任务提供更可靠的结果。

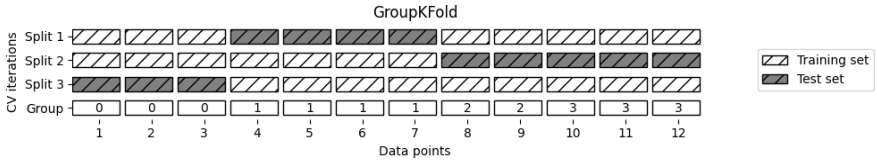

Figure 1.4: Label-dependent splitting with GroupKFold

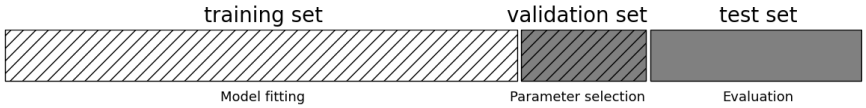

Figure 1.5: A threefold split of data into training set, validation set, and test set

分组交叉验证

GroupKFold is a variation of k-fold which ensures that the same group is not represented in both testing and training sets. 另一种非常常见的交叉验证适用于数据中的分组高度相关时。比如你想构建一个从人脸图片中识别情感的系统，并且收集了 100 个人的照片的数据集，其中每个人都进行了多次拍摄，分别展示了不同的情感。我们的目标是构建一个分类器，能够正确识别未包含在数据集中的人的情感。你可以使用默认的分层交叉验证来度量分类器的性能。但是这样的话，同一个人的照片可能会同时出现在训练集和测试集中。对于分类器而言，检测训练集中出现过的人脸情感比全新的人脸要容易得多。因此，为了准确评估模型对新的人脸的泛化能力，我们必须确保训练集和测试集中包含不同人的图像。

为了实现这一点，我们可以使用 GroupKFold，它以 groups 数组作为参数，可以用来说明照片中对应的是哪个人。这里的 groups 数组表示数据中的分组，在创建训练集和测试集的时候不应该将其分开，也不应该与类别标签弄混。

1.2 网格搜索

在尝试调参之前，重要的是要理解参数的含义。找到一个模型的重要参数（提供最佳泛化性能的参数）的取值是一项棘手的任务，但对于几乎所有模型和数据集来说都是必要的。

考虑一个具有 RBF（径向基函数）核的核 SVM 的例子，它在 SVC 类中实现，它有 2 个重要参数：核宽度 gamma 和正则化参数 C。

1.2.1 简单网格搜索

我们可以实现一个简单的网格搜索，在 2 个参数上使用 for 循环，对每种参数组合分别训练并评估一个分类器

1.2.2 参数过拟合的风险与验证集

我们尝试了许多不同的参数，并选择了在测试集上精度最高的那个，但这个精度不一定能推广到新数据上。由于我们使用测试数据进行调参，所以不能再用它来评估模型的好坏。我们需要一个独立的数据集来进行评估，一个在创建模型时没有用到的数据集。

为了解决这个问题，一种方法是再次划分数据，这样我们得到 3 个数据集：用于构建模型的训练集，用于选择模型参数的验证集（开发集），用于评估所选参数性能的测试集。Figure 1.5 给出了这3个集合的图示。

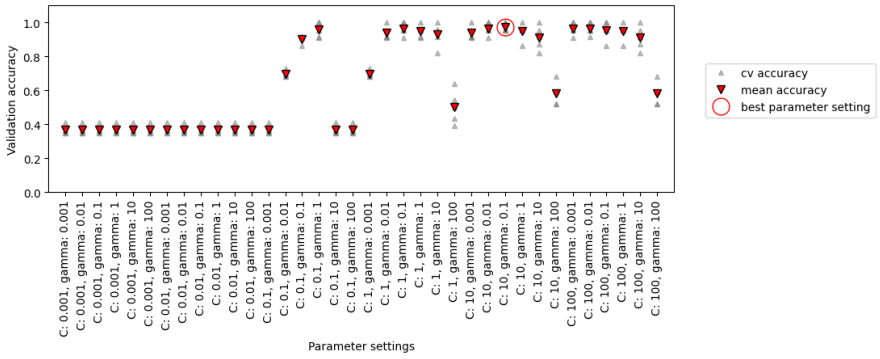

Figure 1.6: Results of grid search with cross-validation

训练集、验证集和测试集之间的区别对于在实践中应用机器学习方法至关重要。任何根据测试集精度所做的选择都会将测试集的信息“泄漏”（leak）到模型中。因此，保留一个单独的测试集是很重要的，它仅用于最终评估。好的做法是利用训练集和验证集的组合完成所有的探索性分析与模型选择，并保留测试集用于最终评估——即使对于探索性可视化也是如此。严格来说，在测试集上对不止一个模型进行评估并选择更好的那个，将会导致对模型精度过于乐观的估计。

1.2.3 带交叉验证的网格搜索

虽然将数据划分为训练集、验证集和测试集的方法（如上所述）是可行的，也相对常用，但这种方法对数据的划分方法相当敏感。为了得到对泛化性能的更好估计，我们可以使用交叉验证来评估每种参数组合的性能，而不是仅将数据单次划分为训练集与验证集。

> 交叉验证是在特定数据集上对给定算法进行评估的一种方法。但它通常与网格搜索等参数搜索方法结合使用。因此，许多人使用交叉验证（cross-validation）这一术语来通俗地指代带交叉验证的网格搜索。

划分数据、运行网格搜索并评估最终参数，这整个过程如 Figure 1.7 所示。

由于带交叉验证的网格搜索是一种常用的调参方法，因此 scikit-learn 提供了 GridSearchCV 类，它以估计器（estimator）的形式实现了这种方法。要使用 GridSearchCV 类，你首先需要用一个字典指定要搜索的参数。然后 GridSearchCV 会执行所有必要的模型拟合。字典的键是我们要调节的参数名称，字典的值是我们想要尝试的参数设置。

创建的 `grid_search` 对象的行为就像是一个分类器，我们可以对它调用标准的 fit、 predict 和 score 方法[1]。

拟合 GridSearchCV 对象不仅会搜索最佳参数，还会利用得到最佳交叉验证性能的参数在整个训练数据集上自动拟合一个新模型。

利用交叉验证选择参数，我们实际上找到了一个在测试集上精度为 97% 的模型。重要的是，我们没有使用测试集来选择参数。我们找到的参数保存在 `best_params_` 属性中，而交叉验证最佳精度（对于这种参数设置，不同划分的平均精度）保存在 `best_score_` 中。

[1]用另一个估计器创建的 scikit-learn 估计器被称为元估计器（meta-estimator）。GridSearchCV 是最常用的元估计器

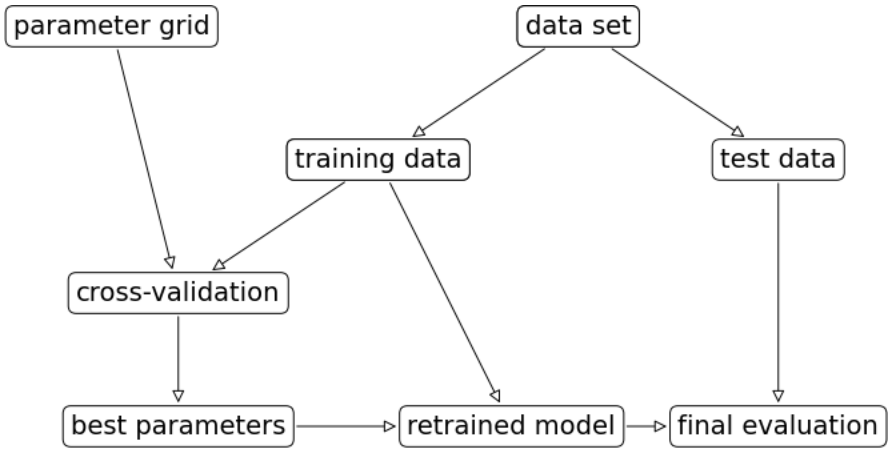

Figure 1.7: Overview of the process of parameter selection and model evaluation with gridsearch

> 同样，注意不要将 `best_score_` 与模型在测试集上调用 score 方法计算得到的泛化性能弄混。使用 score 方法（或者对 predict 方法的输出进行评估）采用的是在整个训练集上训练的模型。而 `best_score_` 属性保存的是交叉验证的平均精度，是在训练集上进行交叉验证得到的。

能够访问实际找到的模型，这有时是很有帮助的，比如查看系数或特征重要性。你可以用 `best_estimator_` 属性来访问最佳参数对应的模型，它是在整个训练集上训练得到的。由于 `grid_search` 本身具有 predict 和 score 方法，所以不需要使用 `best_estimator_` 来进行预测或评估模型。

分析交叉验证的结果

将交叉验证的结果可视化通常有助于理解模型泛化能力对所搜索参数的依赖关系。由于运行网格搜索的计算成本相当高，所以通常最好从相对比较稀疏且较小的网格开始搜索。然后我们可以检查交叉验证网格搜索的结果，可能也会扩展搜索范围。网格搜索的结果可以在 `cv_results_` 属性中找到，它是一个字典，其中保存了搜索的所有内容。

`cv_results_` 中每一行对应一种特定的参数设置。对于每种参数设置，交叉验证所有划分的结果都被记录下来，所有划分的平均值和标准差也被记录下来。由于我们搜索的是一个二维参数网格（C 和 gamma），所以最适合用热图可视化。

可以看到，SVC 对参数设置非常敏感。对于许多种参数设置，精度都在 40% 左右，这是非常糟糕的；对于其他参数设置，精度约为 96%。我们可以从这张图中看出以下几点。首先，我们调节的参数对于获得良好的性能非常重要。这两个参数（C 和 gamma）都很重要，因为调节它们可以将精度从 40% 提高到 96%。此外，在我们选择的参数范围中也可以看到输出发生了显著的变化。同样重要的是要注意，参数的范围要足够大：每个参数的最佳取值不能位于图像的边界上。

如果对于不同的参数设置都看不到精度的变化，也可能是因为这个参数根本不重要。通常最好在开始时尝试非常极端的值，以观察改变参数是否会导致精度发生变化。

基于交叉验证分数来调节参数网格是非常好的，也是探索不同参数的重要性的好方法。但是，你不应该在最终测试集上测试不同的参数范围——前面说过，只有确切知道了想要使用的模型，才能对测试

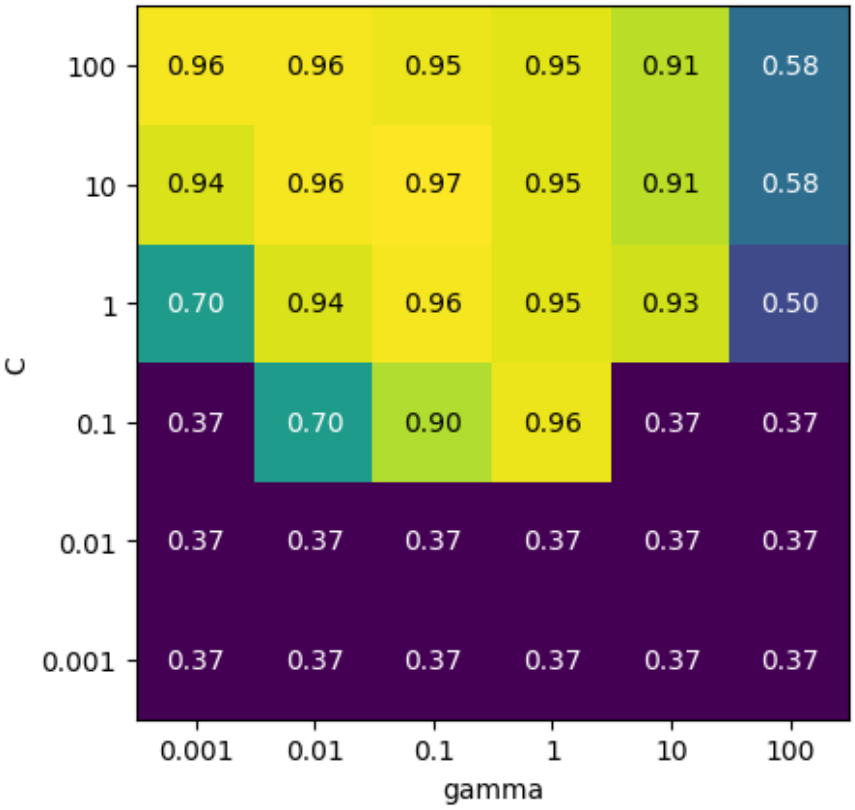

Figure 1.8: Heat map of mean cross-validation score as a function of C and gamma

集进行评估。

在非网格的空间中搜索

在某些情况下，尝试所有参数的所有可能组合（正如 GridSearchCV 所做的那样）并不是一个好主意。例如，SVC 有一个 kernel 参数，根据所选择的 kernel（内核），其他参数也是与之相关的。如果 kernel='linear'，那么模型是线性的，只会用到 C 参数。如果 kernel='rbf'，则需要使用 C 和 gamma 两个参数（但用不到类似 degree 的其他参数）。在这种情况下，搜索 C、gamma 和 kernel 所有可能的组合没有意义：如果 kernel='linear'，那么 gamma 是用不到的，尝试 gamma 的不同取值将会浪费时间。为了处理这种“条件”（conditional）参数，GridSearchCV 的 `param_grid` 可以是字典组成的列表（a list of dictionaries）。列表中的每个字典可扩展为一个独立的网格。

使用不同的交叉验证策略进行网格搜索

与 `cross_val_score` 类似，GridSearchCV 对分类问题默认使用分层 k 折交叉验证，对回归问题默认使用 k 折交叉验证。但是，你可以传入任何交叉验证分离器作为 GridSearchCV 的 cv 参数。

嵌套交叉验证 前面在使用 GridSearchCV 时，我们仍然将数据单次划分为训练集和测试集，这可能会导致结果不稳定，也让我们过于依赖数据的此次划分。我们可以再深入一点，不是只将原始数据一次划分为训练集和测试集，而是使用交叉验证进行多次划分，这就是所谓的**嵌套交叉验证**（nested cross-validation）。在嵌套交叉验证中，有一个外层循环，遍历将数据划分为训练集和测试集的所有划分。对于每种划分都运行一次网格搜索（对于外层循环的每种划分可能会得到不同的最佳参数）。然后，对于每种外层划分，利用最佳参数设置计算得到测试集分数。GridSearchCV虽然做了交叉验证，但是其只是找出最优的参数，使用这个最优参数的模型后仍然要交叉验证，否则只是一次划分数据的结果，随机波动性很大

交叉验证与网格搜索并行 虽然在许多参数上运行网格搜索和在大型数据集上运行网格搜索的计算量可能很大，但令人尴尬的是，这些计算都是并行的（parallel）。这也就是说，在一种交叉验证划分下使用特定参数设置来构建一个模型，与利用其他参数的模型是完全独立的。这使得网格搜索与交叉验证成为多个 CPU 内核或集群上并行化的理想选择。你可以将 `n_jobs` 参数设置为你想使用的 CPU 内核数量，从而在 GridSearchCV 和 `cross_val_score` 中使用多个内核。你可以设置 `n_jobs=-1` 来使用所有可用的内核。

你应该知道，scikit-learn 不允许并行操作的嵌套。因此，如果你在模型（比如随机森林）中使用了 `n_jobs` 选项，那么就不能在 GridSearchCV 使用它来搜索这个模型。如果你的数据集和模型都非常大，那么使用多个内核可能会占用大量内存，你应该在并行构建大型模型时监视内存的使用情况。

还可以在集群内的多台机器上并行运行网格搜索和交叉验证，不过目前scikit-learn 还不支持这一点。但是，如果你不介意编写 for 循环来遍历参数的话，可以使用 IPython 并行框架来进行并行网格搜索。

对于 Spark 用户，还可以使用最新开发的 spark-sklearn 包（应该已经被弃用了），它允许在已经建立好的 Spark 集群上运行网格搜索。

1.3 评估指标与评分

1.3.1 牢记最终目标

在选择指标时，你应该始终牢记机器学习应用的最终目标。在实践中，我们通常不仅对精确的预测感兴趣，还希望将这些预测结果用于更大的决策过程。在选择机器学习指标之前，你应该考虑应用的高级目标，这通常被称为商业指标（business metric）。对于一个机器学习应用，选择特定算法的结果被称为商业影响（business impact）。要想评估某个模型的商业影响，可能需要将它放在真实的生产环境中。在开发的初期阶段调参，仅为了测试就将模型投入生产环境往往是不可行的，因为可能涉及很高的商业风险或个人风险。

1.3.2 二分类指标

错误类型

不平衡数据集

混淆矩阵

考虑不确定性

准确率-召回率曲线

受试者工作特征（ROC）与AUC

1.3.3 多分类指标

1.3.4 回归指标

1.3.5 在模型选择中使用评估指标

1.4 小结