专知

【导读】这是一篇完全手把手进行机器学习项目构建的教程，包含：1. 数据清理和格式化 2. 探索性数据分析 3. 特征工程和特征选择 4. 在性能指标上比较几种机器学习模型 5. 对最佳模型执行超参数调整 6. 在测试集合中评估最佳模型 7. 解释模型结果 8. 得出结论。在第一篇文章中，我们对数据进行了清理和结构化，进行了探索性的数据分析，开发了一组用于我们模型的特征，并建立了一个基准（baseline）来衡量性能。在本文中，我们将讨论如何实现和比较Python中的几种机器学习模型，执行超参数优化，对最佳模型进行优选，并对测试集上的最终模型进行评估。

【干货】Python机器学习项目实战1（附代码）

作者 | William Koehrsen
编译 | 专知
参与 | Mandy, Xiaowen

用python完成一个完整的机器学习项目：第二部分
——Model Selection, Hyperparameter Tuning, and Evaluation

集合解决问题所需的所有机器学习程序可能是一项艰巨的任务。在本系列文章中，我们将通过使用真实数据集实现机器学习工作流程，以了解各个技术是如何结合在一起的。

此项目的完整代码在GitHub上，并附上与本文相对应的参考笔记。您可以随意使用、共享和修改代码!

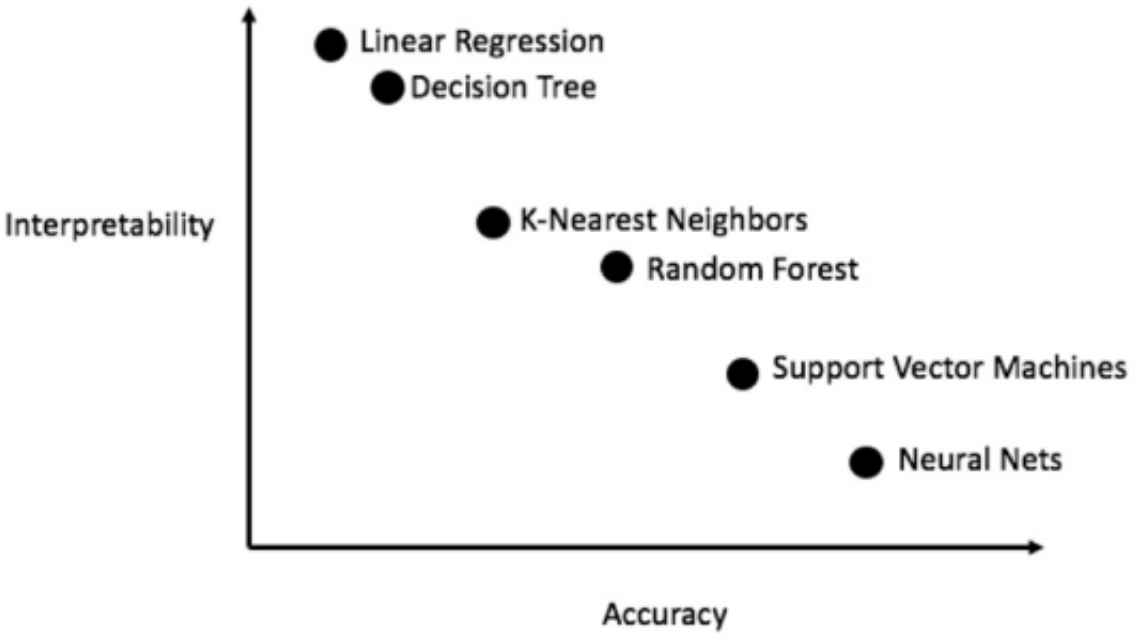
第二部分代码：
https://github.com/WillKoehrsen/machine-learning-project-walkthrough/blob/master/Machine%20Learning%20Project%20Part%202.ipynb

模型评估和选择（Model Evaluation and Selection）

作为一个提醒，我们正在研究一个有监督的回归任务（a supervised regression task）：利用纽约市建筑能源数据（New York City building energy data），我们想开发一个模型，可以预测建筑物的能源之星得分（Energy Star Score）。我们关注的重点是预测的准确性和模型的可解释性。

有大量的机器学习模型可供选择，这会让你难以决定从哪里开始。虽然有些图表试图告诉你要使用哪种算法，但我更愿意多尝试几种算法，并查看哪种算法效果最好！机器学习仍然是一个主要由经验（实验）而不是理论结果驱动的领域，事先知道哪种模型最好，几乎是不可能的。

一般来说，从简单的可解释模型（如线性回归）开始是一个好办法，如果性能不足，继而转向更复杂但通常更准确的模型。下图显示了一种版本的准确性与可解释性之间的权衡：



我们将评估五种不同的模型：

- **线性回归（Linear Regression）**
- **K最近邻回归（K-Nearest Neighbors Regression）**
- **随机森林回归（Random Forest Regression）**
- **梯度增强回归（Gradient Boosted Regression）**
- **支持向量机回归（Support Vector Machine Regression）**

在这篇文章中，我们将着重于实现这些方法而不是背后的理论。对于想要了解背景知识的人来说，我强烈推荐阅读“An Introduction to Statistical Learning “【1】，或者“Hands-On Machine Learning with Scikit-Learn and TensorFlow”【2】。这两本书都很好地解释了理论，并展示了如何分别有效地使用R和Python中的方法。

- **填补缺失值（Imputing Missing Values）**

虽然我们在处理数据时丢失了超过50％缺失值的列，但仍有不少观察结果丢失。 机器学习模型无法处理任何缺失值，因此我们必须把它们填充进去，这是一个称为Imputing的过程【3】。

首先，我们将读入所有数据并提醒自己这些数据是什么样子的：

```
import pandas as pd
import numpy as np
# Read in data into dataframes
train_features = pd.read_csv('data/training_features.csv')
test_features = pd.read_csv('data/testing_features.csv')
train_labels = pd.read_csv('data/training_labels.csv')
test_labels = pd.read_csv('data/testing_labels.csv')
Training Feature Size:  (6622, 64)
Testing Feature Size:   (2839, 64)
Training Labels Size:   (6622, 1)
Testing Labels Size:    (2839, 1)
```

Year Built	Number of Buildings - Self-reported	Occupancy	Site EUI (kBtu/ft²)	Weather Normalized Site Electricity Intensity (kWh/ft²)	Weather Normalized Site Natural Gas Intensity (therms/ft²)	Water Intensity (All Water Sources) (gal/ft²)	Latitude	Longitude	Community Board	Census Tract	log_Direct GHG Emissions (Metric Tons CO2e)	log_Water Intensity (All Water Sources) (gal/ft²)	Borough_Staten Island
1950	1	100	126.0	5.2	1.2	99.41	NaN	NaN	NaN	NaN	6.088818	4.599253	0
1926	1	100	95.4	4.7	0.9	NaN	40.835496	-73.887745	3.0	161.0	5.384036	NaN	0
1954	1	100	40.4	3.8	0.3	NaN	40.663206	-73.949469	9.0	329.0	5.017280	NaN	0
1992	1	100	157.1	16.9	1.1	NaN	40.622968	-74.078742	1.0	27.0	6.510653	NaN	1
1927	1	100	62.3	3.5	0.0	28.65	40.782421	-73.972622	7.0	165.0	6.123589	3.355153	0
1929	1	90	52.9	9.7	0.2	4.80	40.725136	-74.004436	2.0	37.0	5.516649	1.568616	0
1942	1	100	66.8	3.0	0.6	67.14	40.637833	-73.973045	12.0	490.0	5.426271	4.206780	0
1938	1	100	78.4	5.7	NaN	30.73	40.776035	-73.964418	8.0	142.0	6.067036	3.425239	0
1959	1	100	63.0	3.4	0.5	41.96	NaN	NaN	NaN	NaN	6.170447	3.736717	0
1941	1	100	97.8	4.3	0.8	86.88	NaN	NaN	NaN	NaN	5.680655	4.464526	0
1922	1	100	55.4	4.5	0.0	NaN	40.762510	-73.970065	5.0	11203.0	1.335001	NaN	0

显示NaN的值都代表缺少观察结果。虽然有很多方法可以填补缺失数据，但我们将使用一种相对简单的方法，即中值插补法（ median imputation ）。这将使用该列的中值（ the median value ）替换列中的所有缺失值。

在下面的代码中，我们创建了一个Scikit-Learn Imputer对象，并将strategy设置为median。 然后我们在训练数据上训练这个对象（使用imputer.fit），并用它来填充训练和测试数据中的缺失值（使用imputer.transform）。 这意味着测试数据中的缺失值将填入训练数据的相应中值。

（我们必须这样做插补（ imputation ），而不是对所有数据进行训练，这样做是为了避免测试数据泄漏问题（ test data leakage ），测试数据集中的信息会溢出到训练数据中。）

```
# Create an imputer object with a median filling strategy
imputer = Imputer(strategy='median')
# Train on the training features
imputer.fit(train_features)
# Transform both training data and testing data
X = imputer.transform(train_features)
X_test = imputer.transform(test_features)
Missing values in training features:  0
Missing values in testing features:   0
```

现在所有的特征都具有真实的，有限的值，没有缺失值。

- **特征缩放（Feature Scaling）**

缩放（ Scaling ）是指改变特征范围的一般过程。这是必要的，因为特征是以不同单位测量的，因此涵盖了不同的范围。比如支持向量机（ svm ）和考虑到观测之间的距离度量的K-近邻的方法，这两种方法显著地受到特征范围的影响，Scaling使得他们可以学习（ learn ）。尽管线性回归和随机森林等方法实际上并不需要特征缩放，但在比较多种算法

时采取这一步骤仍然是最佳做法。

我们将每个特征缩放至0-1之间。通过获取每个特征值，减去特征中的最小值并除以最大值减去最小值（范围）来完成。这种缩放通常称为归一化（normalization），或者标准化（standardization）。

虽然这个过程很容易通过人工计算实现，但我们可以在Scikit-Learn中使用MinMaxScaler对象来完成。此方法的代码与imputation相同，除了使用scaler来代替imputer！再次，我们确保仅使用训练数据进行训练，之后再转换所有数据。

```
# Create the scaler object with a range of 0-1
scaler = MinMaxScaler(feature_range=(0, 1))
# Fit on the training data
scaler.fit(X)
# Transform both the training and testing data
X = scaler.transform(X)
X_test = scaler.transform(X_test)
```

现在，每个特征的最小值为0，最大值为1。缺少值插补（Missing value imputation）和特征缩放（feature scaling）是几乎所有机器学习流程所需的两个步骤，因此很有必要理解它们的工作原理！

- **在Scikit-Learn中实现机器学习模型（Implementing Machine Learning Models in Scikit-Learn）**

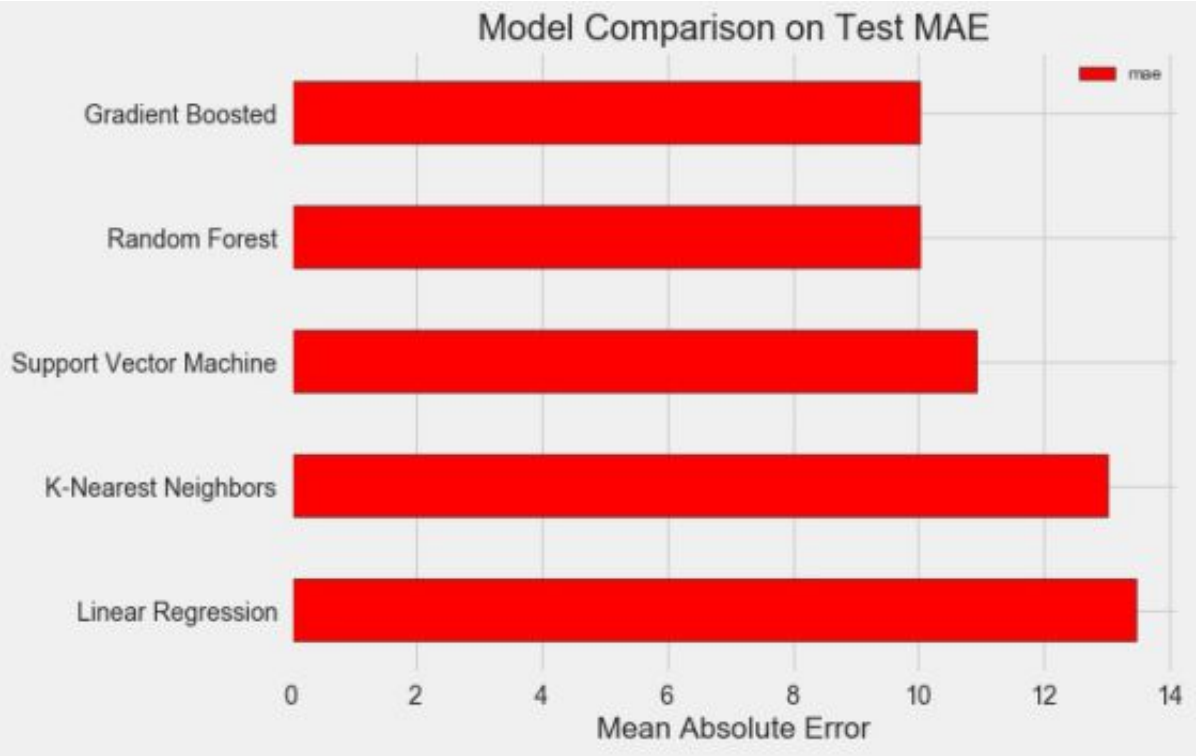
在我们花费大量时间对数据进行清理和格式化之后，实际创建，训练和预测模型相对简单。我们将在Python中使用Scikit-Learn库，它有着很好的说明文档和一致的模型构建语法。 一旦你知道如何在Scikit-Learn中创建一个模型，你就可以快速实现各种算法。

我们通过一个例子来说明如何进行模型的创建，训练（.fit）和测试（.predict），使用Gradient Boosting Regressor：

```
from sklearn.ensemble import GradientBoostingRegressor

# Create the model
gradient_boosted = GradientBoostingRegressor()

# Fit the model on the training data
gradient_boosted.fit(X, y)
```

```
# Make predictions on the test data
predictions = gradient_boosted.predict(X_test)

# Evaluate the model
mae = np.mean(abs(predictions - y_test))

print('Gradient Boosted Performance on the test set: MAE = %0.4f' % mae)
Gradient Boosted Performance on the test set: MAE = 10.0132
```

模型的创建，训练和测试都是一条线！构建其他模型，我们使用相同的语法，只改变算法的名称。 结果如下：



为了使这些数字正确，用目标的中值计算的朴素baseline是24.5。显然，机器学习适用于我们的问题，因为相比于baseline有显著的提高!

梯度增强回归(MAE = 10.013)略优于随机森林(MAE = 10.014)。这些结果并不完全公平，因为我们主要使用了超参数的默认值。特别是在支持向量机等模型中，性能高度依赖于这些设置。最终，从这些比较的结果中，我们将选择梯度增强回归（ gradient boosted regressor ）模型来进行模型优化。

模型优化的超参数调整（Hyperparameter Tuning for Model Optimization）

在机器学习中，在我们选择了一个模型后，我们可以通过调整模型超参数来对我们的问题进行优化。

首先，**超参数是什么，它们与参数有什么不同？【4】**

• **模型超参数**（hyperparameters）被认为是机器学习算法的最好设置，该算法是由数据科学家在训练之前设置的。例如，随机森林中树木的数量，或者k -最近邻居算法中使用的邻居数。

• **模型参数**（parameters）是模型在训练过程中学习的内容，例如线性回归中的权重。

通过改变模型中欠拟合和过拟合的平衡来控制影响模型性能的超参数。当我们的模型不够复杂（它没有足够的自由度）来学习从特征到目标的映射时，就是欠拟合（Underfitting）。一个欠拟合的模型有很高的偏置（bias），我们可以改变我们的模型使其更复杂来纠正。

过拟合（Overfitting）是当我们的模型基本上拟合了所有训练数据点的时候。过拟合模型具有很高的方差（variance），我们可以通过正则化（regularization）来限制模型的复杂性来纠正。欠拟合和过拟合模型都不能很好地适用于测试数据（test data）。

•选择正确的超参数的问题在于，对于每个机器学习问题，最优集都会有所不同！因此，找到最佳设置的唯一方法是在每个新数据集上尝试一些设置。幸运的是，Scikit-Learn有许多方法可以让我们高效地评估超参数。此外，Epistasis Lab的TPOT等项目正试图使用遗传编程(genetic programming)等方法优化超参数搜索。在这个项目中，我们将使用Scikit-Learning来完成这一任务，但是我们将继续关注自动ML场景中的更多工作。

- **随机搜索与交叉验证（Random Search with Cross Validation）**

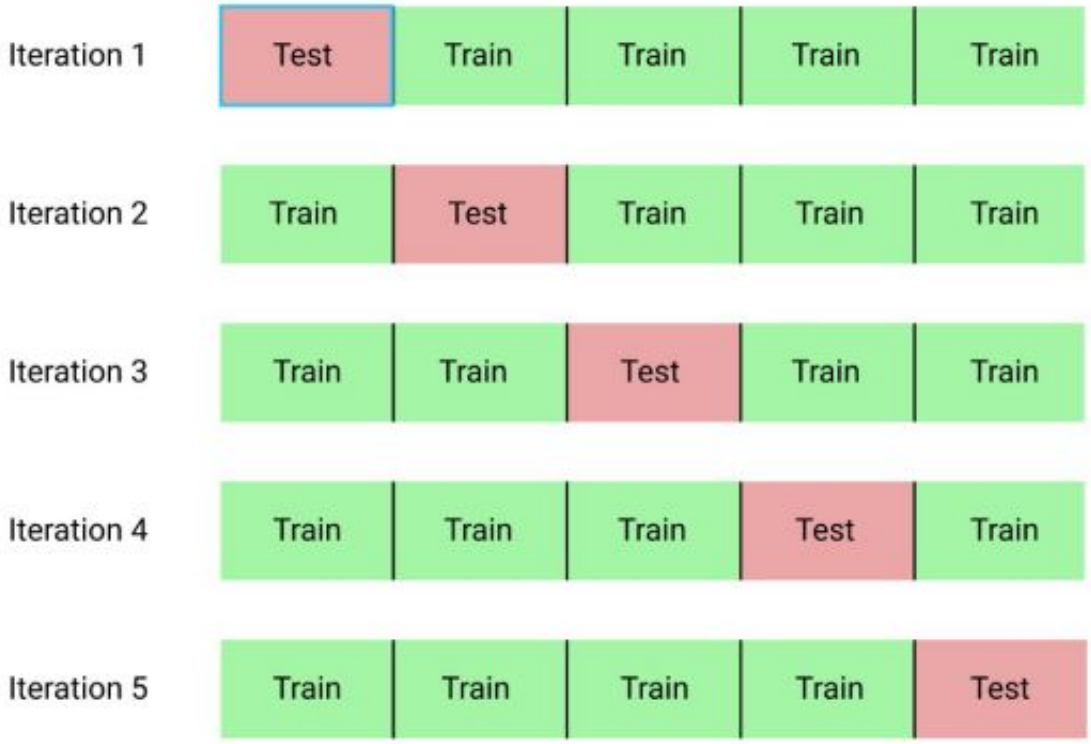
我们将要实现的特定超参数调整方法称为随机搜索和交叉验证：

• **随机搜索（Random Search）**是指我们用来选择超参数的技术。 我们定义一个网格，然后随机抽样不同的组合，而不是网格搜索（grid search），我们会彻底地尝试每一个组合。（令人惊讶的是，随机搜索的结果几乎和网格搜索一样，但大大缩短了运行时间。）

• **交叉验证（Cross Validation）**是我们用来评估所选超参数组合的技术。 我们使用K折交叉验证，而不是将训练集分成单独的训练集和验证集，这会减少我们可以使用的训练数

据量。交叉验证涉及到将训练数据分成K个folds，然后经历一个迭代过程，在这个迭代过程中，我们首先训练前K-1个folds的数据，然后在第K个fold的数据上进行评估表现。我们重复这个过程K次，在K-fold交叉验证结束时，我们将每个K次迭代的平均误差作为最终的性能度量。

K = 5的K-fold交叉验证的过程如下所示：



使用交叉验证执行随机搜索的整个过程是：

1. **建立一个待评估超参数网格**
2. **随机抽样一组超参数**
3. **用选定的组合创建一个模型**
4. **使用K-fold交叉验证评估模型**
5. **确定哪些超参数运行得最好**

当然，我们实际上并没有手动做这件事，而是让Scikit-Learn的RandomizedSearchCV处理所有的工作！

- **Gradient Boosted Methods（梯度增强方法）**

由于我们将使用梯度增强回归模型，我应该至少给出一点背景知识！这个模型是一个集成方法，这意味着它是由许多弱的学习器（weak learners）构建的，在这个例子中是决策树

（individual decision trees）。诸如随机森林之类的bagging算法会对弱的学习器进行并行训练，并让他们投票（vote）进行预测，但像梯度增强这样的方法，将会依次训练学习器，并且每个学习器都“集中了”前面的所有错误。

增强方法近年来越来越流行，并且频繁赢得机器学习竞赛。梯度增强方法（Gradient Boosting Method）是一种特殊的实现，它使用梯度下降，基于前面的残差进行连续训练来最小化成本函数。scikit-learn实现的梯度增强通常被认为效率低于XGBoost等其他库，但它对于我们的小数据集来说足够好，并且相当准确。

- **回归超参数调整（Back to Hyperparameter Tuning）**

在Gradient Boosted Regressor中有许多超参数可以调整（tune），您可以查看Scikit-Learn文档以了解详细信息。在这个项目中，我们将优化以下超参数：
• loss：用来最小化的损失函数。
• n_estimators：弱学习器(决策树)的使用数量。
• max_depth：每个决策树的最大深度。
• min_samples_leaf：决策树的叶节点所需的最小示例数量。
• min_samples_split：拆分决策树节点所需的最小示例数量。
• max_features：用于拆分节点的最大特征数。

我不确定是否有人真正了解所有这些超参数之间的相互作用，找到最佳组合的唯一方法就是try！

在下面的代码中，我们构建一个超参数网格，创建一个RandomizedSearchCV对象，并在超过25个不同的超参数组合中使用4-fold交叉验证来执行超参数搜索：

```
# Loss function to be optimized
loss = ['ls', 'lad', 'huber']

# Number of trees used in the boosting process
n_estimators = [100, 500, 900, 1100, 1500]

# Maximum depth of each tree
max_depth = [2, 3, 5, 10, 15]

# Minimum number of samples per leaf
min_samples_leaf = [1, 2, 4, 6, 8]

# Minimum number of samples to split a node
min_samples_split = [2, 4, 6, 10]
```

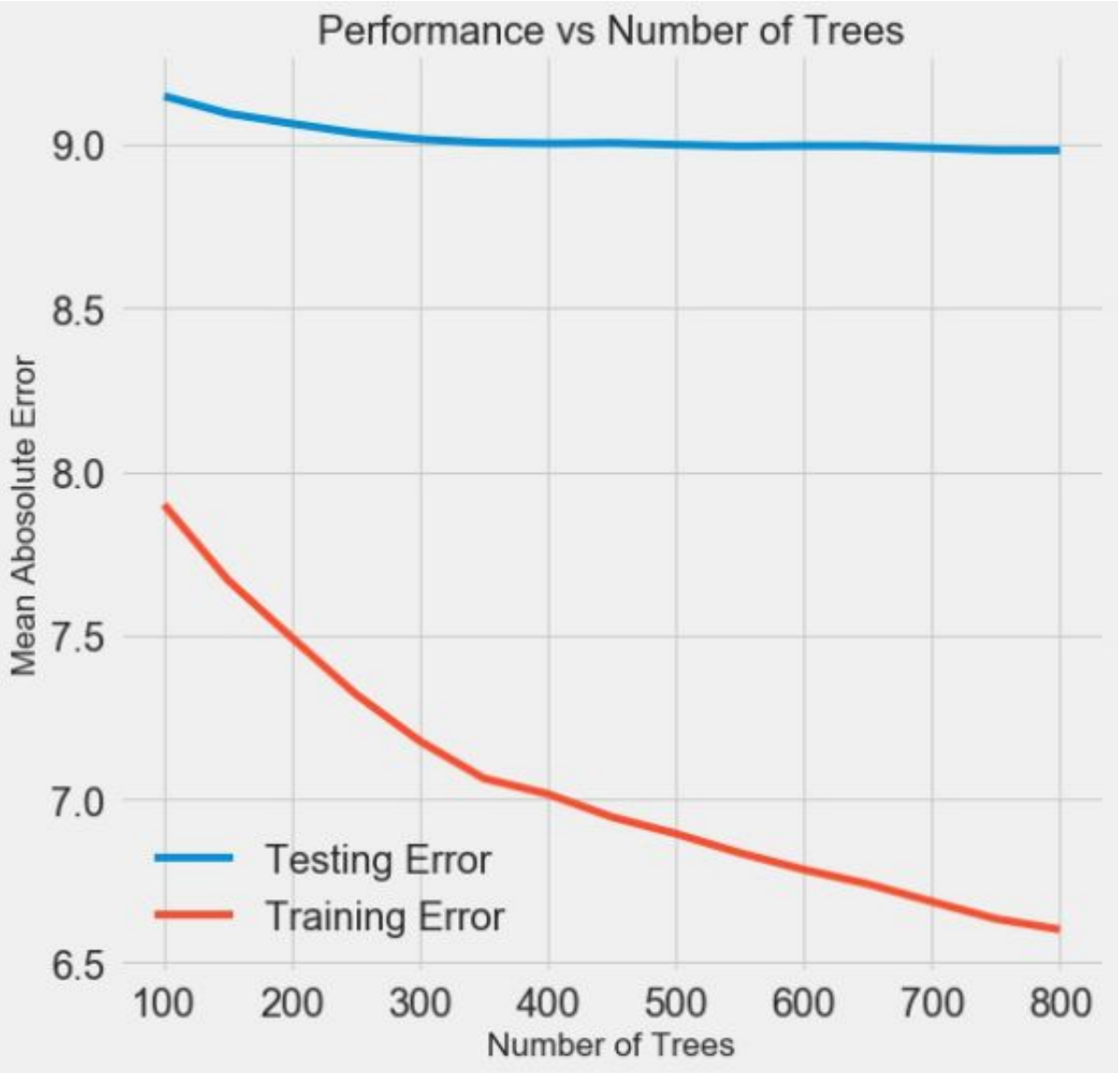
```
# Maximum number of features to consider for making splits
max_features = ['auto', 'sqrt', 'log2', None]

# Define the grid of hyperparameters to search
hyperparameter_grid = {'loss': loss,
                       'n_estimators': n_estimators,
                       'max_depth': max_depth,
                       'min_samples_leaf': min_samples_leaf,
                       'min_samples_split': min_samples_split,
                       'max_features': max_features}

# Create the model to use for hyperparameter tuning
model = GradientBoostingRegressor(random_state = 42)

# Set up the random search with 4-fold cross validation
random_cv = RandomizedSearchCV(estimator=model,
                                           param_distributions=hyperparameter_grid,
                                           cv=4, n_iter=25,
                                           scoring = 'neg_mean_absolute_error',
                                           n_jobs = -1, verbose = 1,
                                           return_train_score = True,
                                           random_state=42)

# Fit on the training data
random_cv.fit(X, y)
```

在执行搜索之后，我们可以检查RandomizedSearchCV对象以找到最佳模型:

```
# Find the best combination of settings
random_cv.best_estimator_
GradientBoostingRegressor(loss='lad', max_depth=5,
                                      max_features=None,
                                      min_samples_leaf=6,
                                      min_samples_split=6,
                                      n_estimators=500)
```

然后，我们可以使用这些结果来执行网格搜索，方法是选择接近这些最优值的网格参数。但是，进一步调整不太可能显著改善我们的模型。作为一般规则，适当的特征工程（feature engineering）对模型性能的影响要比最广泛的超参数调整更大。这是机器学习中的收益递减法则：特征工程在大多数情况下会对实现目标有很大的帮助，而超参数调整通常只会带来很小的收益。

我们可以尝试的一个实验是改变估计器（决策树）的数量，同时保持其他超参数稳定。这直接让我们观察到这个特定设置的效果。具体实现方法请参阅笔记，但结果显示如下：



随着模型使用的树的数量增加，训练和测试的误差都会减少。 但是，训练误差比测试误差下降得快得多，我们可以看到我们的模型过拟合了：它在训练数据上表现非常好，但在测试集上无法达到相同的性能。

我们总是期望在测试集上的表现或多或少会有所下降，但是测试集和训练集的差距太显著则表明过拟合。我们可以通过获取更多训练数据来解决过度拟合问题，或者通过超参数来降低模型的复杂性。在这种情况下，我们将超参数保留在原来的位置，但我鼓励大家尝试任何可行的方法来减少过拟合。

对于最终模型，我们将使用800个估计器（estimators），因为这得到了在交叉验证中的最低误差。现在，测试这个模型的时间到了!

在测试集进行评估（Evaluating on the Test Set）

作为负责任的机器学习工程师，我们确保不让我们的模型在任何训练点上看到测试集的数据。 因此，我们可以使用测试集的表现作为我们的模型在现实世界中部署时的表现。

对测试集进行预测并计算性能是相对简单的。在这里，我们将比较默认的梯度增强回归器（the default Gradient Boosted Regressor）与调参之后的模型的性能：

```
# Make predictions on the test set using default and final model
default_pred = default_model.predict(X_test)
final_pred = final_model.predict(X_test)
Default model performance on the test set: MAE = 10.0118.
Final model performance on the test set:   MAE = 9.0446.
```

超参数调整将模型的准确率提高了约10%。这取决于使用情况，10%可能是一个巨大的改进，但它需要大量的时间投入！

我们也可以使用Jupyter Notebooks中的%timeit命令来计算训练这两个模型（默认模型和调参后模型）需要多长时间。首先是默认模型：

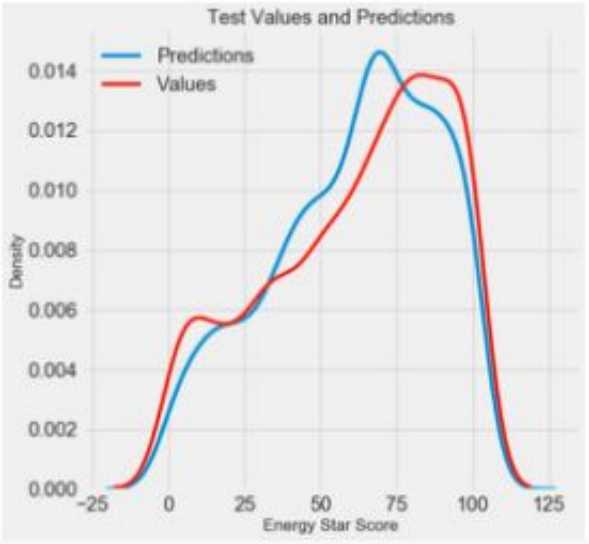
```
%%timeit -n 1 -r 5
default_model.fit(X, y)
1.09 s ± 153 ms per loop (mean ± std. dev. of 5 runs, 1 loop each)
```

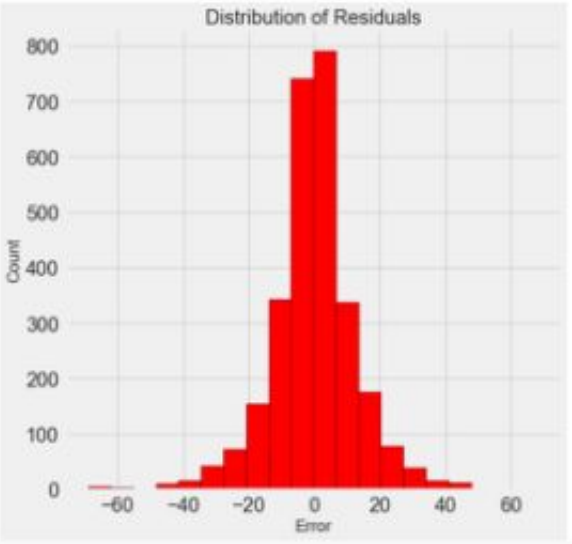
1秒的训练时间似乎挺合理的。最终的调参后的模型并没有那么快：

```
%%timeit -n 1 -r 5
final_model.fit(X, y)
12.1 s ± 1.33 s per loop (mean ± std. dev. of 5 runs, 1 loop each)
```

这证明了机器学习的一个基本特点：它总是一种权衡的游戏。我们必须不断平衡准确性与可解释性，偏差与方差，准确性与运行时间等。这些将最终取决于所要解决的问题。在我们的例子中，相对而言，运行时间增加了12倍是很大的，但从绝对意义上讲，它并没有那么重要。

一旦我们有了最终的预测值，我们就可以看看他们是否表现出明显的偏差。左边是预测值和实际值的密度图，右边是残差的直方图：





模型预测似乎遵循实际值的分布，尽管密度的峰值出现在训练集的中值（66）附近，而非密度的真实峰值（接近100）。 残差几乎是正态分布，尽管在模型预测值远低于真实值的情况下，我们会看到一些大的负值。我们将在下一篇文章中深入探讨模型的结果。

总结（Conclusions）

在本文中，我们介绍了机器学习工作流程中的几个步骤：

- 缺失值的插补和特征的缩放
- 评估和比较几种机器学习模型
- 使用随机网格搜索和交叉验证进行超参数调整
- 评估测试集上的最佳模型

这项工作的结果表明，机器学习适用于我们的任务（使用可用数据预测建筑物的Energy Star Score）。使用梯度增强回归法，我们可以预测测试集的分数在真实值的9.1分以内。此外，我们还看到，超参数的调优可以提高模型的性能，但是需要相当大的时间成本。这是我们在开发机器学习解决方案时必须考虑的众多权衡之一。

在下一篇文章中，我们将着眼于我们创建的黑盒子，并试图理解我们的模型如何进行预测。 我们也将确定影响Energy Star Score的最大因素。虽然我知道我们的模型是准确的，但我们想知道它为什么会做出预测，以及它告诉我们什么问题！

原文链接：
https://towardsdatascience.com/a-complete-machine-learning-project-walk-through-in-python-part-two-300f1f8147e2

代码链接：

https://github.com/WillKoehrsen/machine-learning-project-walkthrough/blob/master/Machine%20Learning%20Project%20Part%202.ipynb

-END-

专·知

人工智能领域主题知识资料查看与加入专知人工智能服务群：

【专知AI服务计划】专知AI知识技术服务会员群加入与人工智能领域26个主题知识资料全集获取

[点击上面图片加入会员]

请PC登录www.zhuanzhi.ai或者点击阅读原文，注册登录专知，获取更多AI知识资料！

请加专知小助手微信（扫一扫如下二维码添加），加入专知主题群（请备注主题类型：AI、NLP、CV、 KG等）交流~

请关注专知公众号，获取人工智能的专业知识！

点击“阅读原文”，使用专知

阅读原文