目录

Boosting（元算法） 1
Ada Boost(适应性 Boost) 3
渐变树增强 8
Xgboost 9
TensorFlow 10

第 5 章

监督学习：高级算法

这是监督学习过程的第二部分。 您现在正在研究监督学习生态系统中更先进的机器学习算法。

提示：这些算法通常用于工业化的机器学习生态系统，因为它们解决了许多常见的问题，您将在示例和它们的解决方案中看到这些问题。

Boosting（元算法）

机器学习术语“Boosting”是指一系列算法，这些算法使弱学习者适应强学习者。 当特征不能产生足够强的学习者来学习

算法时，这些方法是有用的。Boosting 算法使用一组低精度分类器来创建一个高精度分类器。

基本概念（图 5-1）是将机器学习链连接成一个处理链，以获得改进的结果。

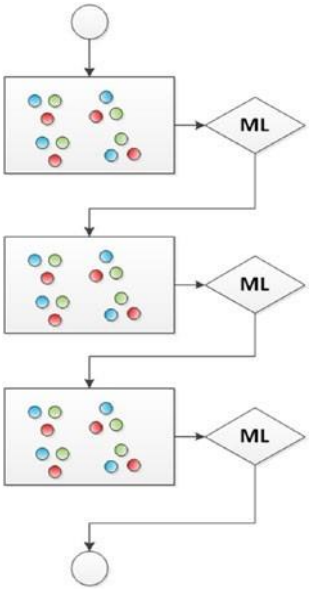

图 5-1。ML Boosting 概念图

这是一个复杂的介绍。 我将快速解释弱学习的概念。

“弱学习”是一种ML算法（用于回归/分类），它提供的精度略优于随机猜测。

Ada Boost(适应性 Boost)

Ada Boost 或 Adaptive Boosting 是一种使数据科学家能够增强数据处理结果以获得更好的结果的方法。

Ada Boost 分类器是一个元估计器，它首先在原始数据集上拟合分类器，然后在同一数据集上拟合分类器的附加副本，但调整错误分类实例的权重，使后续分类器更多地关注困难的情况。

例:

在您的 Jupyter 软件中打开第 005 章示例 001A.ipynb。

使用 Part A，您将激活所需的核心库。

以下三个导入对于您需要执行的工作非常重要。

from sklearn.ensemble import AdaBoostClassifier

Ada Boost 分类器使用 Ada Boost-SAMME 算法集。

这个具体的例子是使用 SAMME 离散升压算法。

你需要的下一个结构是你想要使用的机器学习算法。

您正在使用支持向量分类(SVC)算法。

from sklearn.svm import SVC

数据工程使用标准定标器来准备特征。

from sklearn.preprocessing import StandardScaler

变压器通过删除均值和缩放到单位方差来标准化特征。

既然你有了基本的工具，我将向你展示如何构建一个 boosting（提升型）的解决方案。B 部分和 C 部分提供您需要的数据。 我建议我们用我的 roses 上的数据。在 D 部分中，您使用变压器执行所需特征的缩放。

transformer = StandardScaler(copy=True, with_mean=True, with_std=False)

这将转换数据以支持更好的培训过程。

变压器结果为:

Features: 3

Samples: 420

Scale: None

Mean: [5.82341667 3.07140952 1.22077619]

variance: None

在 PartE，我们创建了一个用于 AdaBoost 的 SVC、

```
svc=SVC(max_iter=5000,
        gamma='auto',
        class_weight='balanced',
        probability=True,
        kernel='linear',
        random_state=0,
        verbose=False)
```

这个 SVC 将被 AdaBoost 使用。

在 F 部分中，您将执行一个 boost 周期:

```
clf1=Ada Boost Classifier((algorithm='SAMME',
                          n_estimators=1,
                          base_estimatorn_estimators=svc,
                          learning_raten_estimators=1,
                          random_staten_estimators=0)
clf1.fit(X_train_scale, y_train)
score1 = clf1.score(X_test_scale,y_test)
```

评分结果：67.22%

在 G 部分中，您执行 5 个周期，得分为：67.7778%-改进!

在 H 部分中，您执行 10 个周期，得分为：73.3333%-改进!

在 I 部分中，您执行 20 个周期，得分为：73.3333%-没有改进?

警告：提升只会提高到一个点，然后你只是浪费时钟周期而没有有效的增益。

您已经成功地将 67.222%提高到 73.3333%，这已经改进了 9.09%。

注意：机器学习可以是本书中的任何一种机器学习算法。

你可以关闭你已经成功完成的笔记本，你的第一个 boost 例子。

在Jupyter软件中打开示例：第05章示例001B.ipynb。

我们将完成另一个示例，但是这个示例使用不同的基本机器学习算法和定标器。

请参阅 D 部分被称为健壮的 Scaler 的定标器。 这个定标器使用对异常值具有鲁棒性的统计数据来转换特征，因为它删除中值并根据分位数范围对数据进行缩放。

在 I 部分中，您将使用 1000 周期的 boost。

E 部分是您可以使用额外的树分类器作为基本机器学习过程的点。

提示：要使 Jupyter 笔记本显示 1000 个周期，您需要执行此修改：

```
%javascript
IPython.OutputArea.prototype._should_scroll=function（lines）{
return false;
}
```

正如您在 J 部分中所观察到的，您实现了从 70.270%到 97.200%的分数改进，因此提高了 38.324！ 这就是 boost 的力量。

你可以关闭这个笔记本。

接下来，我们将研究 Boost 解决方案的值域结果。

打开示例 Jupyter Notebook：第 5 章示例 001C.ipynb

执行完整的笔记本。

随着解决方案的推进，您实现了 69.440%到 82.220%的改进。

我想让你看看在 boost1 到 5 之间缺乏改进。

如果我们绘制预测值域，您将观察到值域中有一个主要的值移位。

您将得到以下四个结果（图 5-2 和图 5-3）。

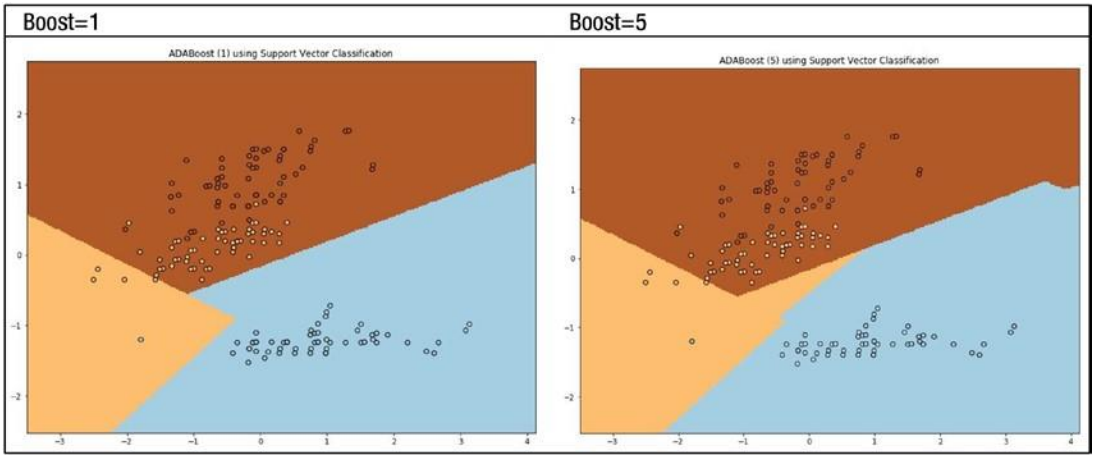

图 5-2。Ada Boost 改变了值域

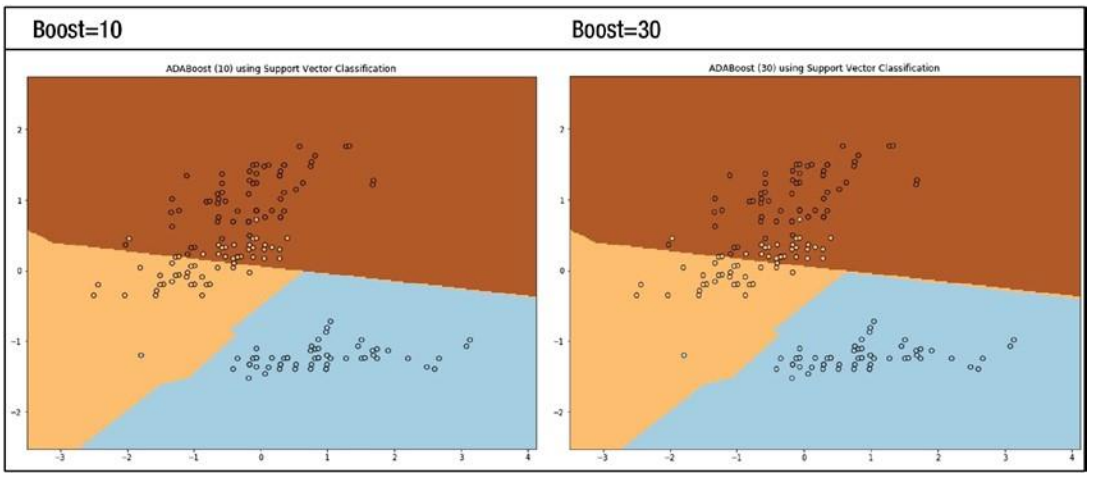

图 5-3。 AdaBoost 不会更改值域

警告：由于域边界上缺少测试数据，该移位不会出现在分数中，因为您没有在这些边缘边界中进行测试。

始终检查您的数据是否覆盖了整个值域！

您可以关闭 notebook。做得好，您现在已经可以执行 boosting 了。

渐变树增强

梯度提升是回归和分类解决方案的机器学习技术，以弱预测模型（特征决策树）组合的形式生成预测模型。

例子：

打开 Jupyter Notebook 的示例：Jupyter 软件中的第 005 章，示例 002.ipynb

核心引擎为：

from sklearn.ensemble import GradientBoostingClassifier

H 部分中的以下命令激活了 Boosting：

clf = GradientBoostingClassifier(** params)

你的结果将如 Figure 5-4 所示。

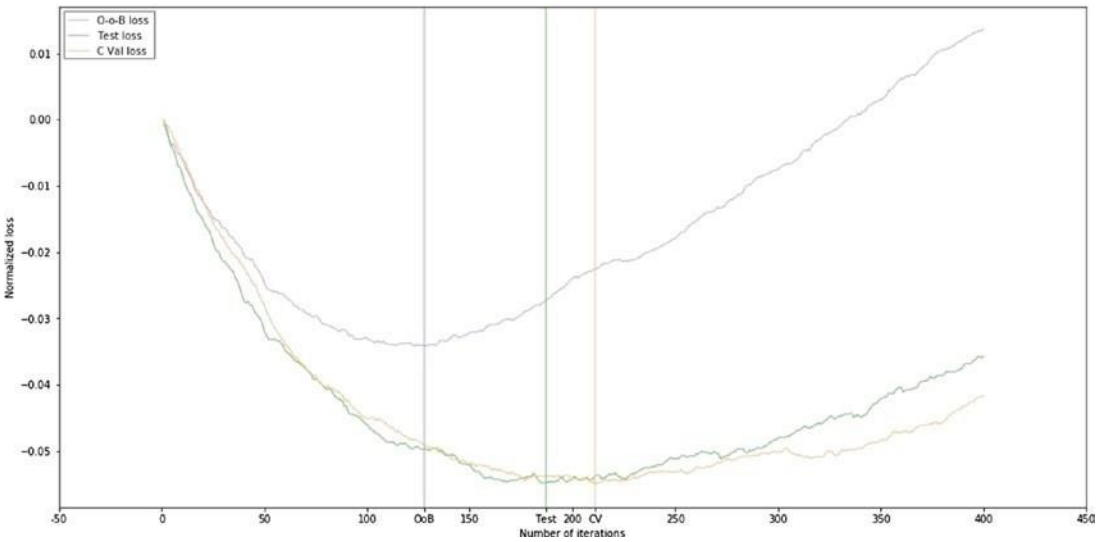

图 5-4。 渐变树增强

您现在可以关闭 Notebook。

Xgboost

XGBoost 提供了一个梯度提升框架。它作为机器学习库非常成功。

该过程支持分布式处理框架 Apache Hadoop、Apache Spark 和 Apache Flink。这使得此框架具有高度可扩展性，也是因此而使用最多的库之一。

Python 库是使用以下方法安装的：

conda install -c conda-forge xgboost

或

conda install -c anaconda py-xgboost （这一个是我的个人选择）

例子：

打开 Jupyter 软件中 Jupyter Notebook 的示例 ：Chapter 005 示例 003.ipynb。

在 A 部分中，加载 xgboost 引擎：

from xgboost import XGBClassifier

在 C 部分中，您通过以下方法使用它：

```
xc = XGBClassifier(max_depth=12,
                   learning_rate=0.05,
                   n_estimators=1968,
                   nthread=8)
```

您的结果：

精度： 82.05128%

您可以使用 E 部分打开机器学习的输出。

您现在可以关闭笔记本。

这是算法示例结束的提示。

现在，您应该能够理解如何使用过程增强算法来帮助机器学习去改进模型的培训。模型训练是 IML 建模部分的最重要的部分。

接下来，我们将研究一个正在出现的算法，即 TensorFlow。

TensorFlow

TensorFlow 是一个开源软件库，用于跨一系列任务进行数据流编程。它是一个象征性的数学库，也用于机器学习应用程序，如神经网络。

TensorFlow 由 Google Brain 团队于 2015 年 11 月开发。TensorFlow 可以在多个 CPU 和 GPU 上运行。

提示：谷歌的 The Machine Learning Crash Course（MLCC）培训课程将使您能够获得对 TensorFlow 更好的基本准备 .

如果你想阅读更多关于背景， 我建议你访问： https://www tensorflow.org/learn

提示：TensorFlow 社区是一个强大而发达的群体。https://www.tensorflow.org/tutorials

提示：作为 TensorFlow 是由谷歌大脑团队开发的。我建议使用：https://colab.research.google.com/notebooks/welcome.ipynb

您只需上传示例即可使用它们。

注意：在本书中我已经用我自己的 TensorFlow。

你可以通过下列语句将 TensorFlow 库安装到 Python 中：

conda install -c conda-forge tensorflow

我将向您展示使用 TensorFlow 进行机器学习的基本构建基块。

TensorFlow 在将数据处理成有意义的信息方面具有巨大的能力。

打开 Jupyter 软件中的示例 Jupyter Notebook：Chapter 005 示例 004.ipynb

注意：您将直接使用 TensorFlow。

本章的稍后部分，我将向您展示如何使用 Keras 作为接口。

在 A 部分中，您将注意到 TensorFlow 接口将 TensorFlow 引入为 tf。

在 B 部分中，您将执行一些基本数学内容，以此向您介绍基本概念。

声明常量如下：a = tf.constant(5)。这是完成 a =5 的。

因此，让我们执行以下计算：

a=5.000, b=10.000, c=20.000, d=12.000, e=89.000

x=5.000 + 10.000 + 20.000

Addition with constants: x = 35

y=5.000 x 10.000 x 20.000

Multiplication with constants: y = 1000

z=2.000 ^ 12.000

Power with constants: z = 4096

s=sqrt(89.000)

Square root with constants: s = 9

我将向您展示如何使用 TensorFlow 处理矩阵。

创建产生 1x3 矩阵的常量操作。该操作作为节点添加到默认图中。构造函数返回的值表示常量操作的输出。

Create Constant that produces a 1x4 matrix.

matrix1 = tf.constant([[10., 11., 12.,13.]])

Create another Constant that produces a 4x1 matrix.

matrix2 = tf.constant([[14.],[15.],[16.],[17.]])

创建以 ″ 矩阵 1″和″矩阵 2″作为输入的 Matmul 操作。返回的值″product″表示矩阵乘法的结果。

product = tf.matmul(matrix1, matrix2)

为了运行 matmul 操作，我们调用 ‘run()’ 方法，传递 ″product″。

操作的输出以 ″结果″ 作为 numpy ‘ndarray’ 对象返回。

with tf.Session() as sess:

result = sess.run(product)

print(result)

您的结果：

[[718.]]

调查 G 部分计算以测试系统的功能：更改 i=1000 并看看它的厉害吧！

恭喜您能够执行基本的 TensorFlow 处理。