目录

CHAPTER 3

机器学习中的分类器

本章将介绍机器学习中的几种分类算法。这包括 kNN（k 近邻）算法、逻辑回归（尽管它叫做分类器）、决策树、随机森林、SVMs 和贝叶斯分类器等算法。对算法的强调旨在向您介绍机器学习，其中包括依赖于 scikit-learn 的基于树的代码示例。本章的后半部分包含标准数据集的基于 Keras 的代码示例。

由于空间限制，本章不介绍其他著名的算法，如 LDA（Linear Discriminant Analysis）和 kMeans 算法（用于无监督学习和聚类）。但是，有许多关于这些算法和其他算法的在线教程。

牢记这些要点，本章第一部分简要讨论了 introduction 中提到的分类器。本章的第二部分概述了激活函数，如果您决定了解深度神经网络，这将非常有用。在本节中，您将了解如何以及为什么要在神经网络中使用它们。本节还包含用于激活函数的 TensorFlow API 的列表，然后是对于他们的一些优点的说明。

第三部分介绍逻辑回归，它依赖于 sigmoid 函数，该函数也用于 RNNs（循环神经网络）和 LSTMs（长期短期记忆）。本章的第四部分包含涉及逻辑回归和 MNIST 数据集的代码示例。

分类器是三种主要算法类型之一：回归算法（如第 4 章中的线性回归）、分类算法（本章讨论）和聚类算法（如 kMeans，本书未讨论）。

另一点：与激活函数相关的部分涉及对神经网络中隐藏层的基本了解。根据您的了解程度，在进入本节之前，阅读一些准备材料可能会有很大收益（有许多文章可再网上找到）。

什么是分类？

给定一个数据集，该数据集包含其类别的观测值，分类的目的是确定新数据所属的类别。类指类别，也称为目标或标签（target，label）。例如，电子邮件服务提供商中的垃圾邮件检测涉及二值分类（只有两个类别）。 MNIST 数据集包含一组图像，其中每个图像都是一个数字，这意味着有 10 个标签。分类中的一些应用包括信用审批、医疗诊断和目标营销。

什么是分类器？

在上一章中，您了解到线性回归将监督学习与数值数据结合使用：目标是训练一个可以进行数值（连续型）预测的模型（例如，明天的股票价格、系统的温度、气压等）。相比之下，分类器将监督学习与各种数据类别结合使用：目标是训练一个可以进行分类预测的模型。

例如，假设数据集中的每一行都是特定的葡萄酒，并且每一列都与特定的葡萄酒特征（单宁、酸度等）相关。进一步假设数据集中有五类葡萄酒： 为了简单起见，让我们将它们标记为 A、B、C、D 和 E。给定一个新的数据点（即新数据行），此数据集的分类器将尝试确定此葡萄酒的标签。

本章中某些分类器可以执行分类任务并进行数字预测（即它们可用于回归和分类）。

常用分类器

此处列出了一些机器学习的最流行的分类器（没有特定顺序）：

- 线性分类器
- kNN
- 逻辑回归
- 决策树
- 随机森林
- SVMs
- 贝叶斯分类器
- CNNs（深度学习）

请记住，不同的分类器具有不同的优点和缺点，这通常涉及复杂性和准确性之间的权衡，类似于 AI 以外的算法。

在深度学习的情况下，CNN（卷积神经网络）执行图像分类，这使得它们成为分类器（它们还可用于音频和文本处理）。

以下各节简要介绍了这些机器学习分类器。

二值分类与多类别分类

二值分类器使用具有两个类别的数据集，而多类别分类器（有时称为多词分类器）区分两个及以上个类别。随机森林分类器和朴素贝叶斯分类器支持多个类别，而 SVM 和线性分类器只能用作二值分类器（但存在 SVM 的多类别扩展）。

此外，还有基于二值分类器的多分类技术：一对多（OvA）和一对一（OvO）。

OvA 技术（也称为"one-versus-the-rest"）涉及多个二值分类器，这些分类器的数量等于类别的数量。例如，如果数据集有五个类，则 OvA 使用五个二值分类器，每个分类器都检测

五个类中的一个。要对此数据点进行分类，请选择输出最高分的二值分类器。

OvO 技术也涉及多个二值分类器，但在这种情况下，二值分类器用于一对类别的训练。例如，如果类是 A、B、C、D 和 E，则需要十个二进制分类器：一个用于 A 和 B，一个用于 A 和 C，一个用于 A 和 D，等等，直到最后一个二值分类器用于 D 和 E。

通常，如果有 n 个类，则需要 n*（n-1）/2 个二值分类器。尽管 OvO 技术比 OvA 技术（例如，20 个类需要 190 个二值分类器）需要更多的二值分类器（例如，20 个类只需要 20 个二值分类器），但 OvO 技术的优点是，每个二值分类器仅对数据集中与两个选定类相关的部分进行训练。

多标签分类

多标签分类涉及在数据集为实例分配多个标签。因此，多标签分类扩展了多类别分类（在上一节中讨论过），后者涉及将单个标签分配给一个实例，其中一个标签可以有很多值。下面是一篇涉及多标签分类 的文章，其中有包含基于 Keras 的代码：

https://medium.com/@vijayabhaskar96/multi-label-image-classification-tutorial-with-keras-imagedatagenerator-cd541f8eaf24

您还可以在网上搜索涉及 SKLearn 或 PyTorch 的多标签分类任务的文章。

什么是线性分类器？

线性分类器将数据集分隔为两个类。线性分类器是 2D 平面上的线、三维空间的平面和更高维的超平面（平面的泛化）。

线性分类器通常是最快的分类器，因此当分类速度非常重要时，通常使用线性分类器。当输入向量是稀疏的（即大部分为零值）或维度数量较大时，线性分类器通常工作的很好。

什么是 knn？

kNN（k 近邻）算法是一种分类算法。简而言之，彼此靠近的数据点被归类为同一类。引入新点时，它将添加到它的邻居（最近）的大多数的类中。例如，假设 k 等于 3，并引入了一个新的数据点。看看它最近的三个邻居的类别：假设他们是 A，A 和 B。然后以其中的多数为新数据点的类别，所以新数据点被标记为 A 类的数据点。

kNN 算法本质上是一种启发式算法，而不是一种具有复杂数学基础的技术，但它仍然是一个有效和有用的算法。

如果要使用简单算法，或者如果您认为数据集的特征高度非结构化，请尝试 kNN 算法。尽管它非常简单，但 kNN 算法可以产生高度非线性的决定。可以在搜索相似项的搜索应用程序中使用 kNN。

通过用向量表示项（item），然后使用适当的距离指标（如欧氏距离）比较向量，来测量项之间的相似性。

kNN 搜索的一些具体示例包括查找语义上相似的文档。

如何处理 knn 中的平局投票

k 取奇数不太可能导致平局投票的结果，但并不是不可能。例如，假设 k 等于 7，并且当引入一个新的数据点时，其七个最近的邻居属于[A，B，A，B，A，B，C]。如您所看到的，没有多数票，因为 A 类有 3 分，B 类有 3 分，C 类有 1 分。

有几种技术处理 kNN 中的平局：

- 将更高权重分配给更近的点
- 增加 k 的值，直到确定获胜者

- 降低 k 的值，直到确定获胜者
- 随机选择一个类

即使将 k 减至 1，仍有可能产生平局投票：可能有两个点与新点的距离相同，因此您需要一种机制来决定选择哪个点作为邻居。

如果 A 类和 B 类是平局，则随机选择 A 类或 B 类。另一个变种是跟踪平局投票，并交替循环，以确保更均匀的分配。

什么是决策树？

决策树是另一种类型的分类算法，它涉及树状结构。在泛型树中，数据点的位置由简单的逻辑条件决定。提供一个简单的示例，假设数据集包含一组表示人员年龄的数字，并且我们还要假设第一个数字是 50。此数字被选为树的根，小于 50 的所有数字都添加到树的左子树上，而大于 50 的所有数字都添加到树的右子树上。

例如，假设数字序列为 {50、25、70、40}。然后，我们可以构建一个树：50 是根节点;25 是 50 的左孩子;70 是 50 的右孩子;40 是 20 的右孩子。我们将添加到此数据集的每个新值需要进行处理，以确定在树中的节点的哪个方向（左或右）上。

列表 3.1 展示了 sklearn_tree2.py 的内容，这个程序定义了欧几里德平面中的一组二维平面上的点及其标签，然后预测欧几里德平面中其他几个点的标签（即类）。

列表 3.1： sklearn_tree2.py

```
from sklearn import tree

# X = pairs of 2D points and Y = the class of each
   point
X = [[0, 0], [1, 1], [2,2]]
Y = [0, 1, 1]

tree_clf = tree.DecisionTreeClassifier()
tree_clf = tree_clf.fit(X, Y)

#predict the class of samples:
print("predict class of [-1., -1.]:")
print(tree_clf.predict([[-1., -1.]]))

print("predict class of [2., 2.]:")
print(tree_clf.predict([[2., 2.]]))

# the percentage of training samples of the same
   class
# in a leaf note equals the probability of each
   class
print("probability of each class in [2.,2.]:")
print(tree_clf.predict_proba([[2., 2.]]))
```

列表 3.1 从 sklearn 导入树，然后初始化数组 X 和 Y。接下来，变量 tree_clf 作为决策树的实例进行初始化，然后通过调用 fit()方法和 x，y 来训练它。

运行列表 3.3 中的代码，你会看到以下输出：

```
predict class of [-1., -1.]:

[0]

predict class of [2., 2.]:

[1]

probability of each class in [2.,2.]:
```

```
[[0.  1.]]
```

点 [-1，-1] 和 [2，2] 分别被 0 和 1 正确标记，这可能正是我们预期的值。

列表 3.2 显示 sklearn_tree3.py 的内容，它扩展了列表 3.1 中的代码（修改以粗体显示）。它添加了第三个标签，并且预测欧几里德平面中三个点而不是两个点的标签。

列表 3.2： sklearn_tree3.py

```
from sklearn import tree

# X = pairs of 2D points and Y = the class of each
   point
X = [[0, 0], [1, 1], [2,2]]
Y = [0, 1, 1]

tree_clf = tree.DecisionTreeClassifier()
tree_clf = tree_clf.fit(X, Y)

#predict the class of samples:
print("predict class of [-1., -1.]:")
print(tree_clf.predict([[-1., -1.]]))

print("predict class of [2., 2.]:")
print(tree_clf.predict([[2., 2.]]))

# the percentage of training samples of the same
   class
# in a leaf note equals the probability of each
   class
print("probability of each class in [2.,2.]:")
print(tree_clf.predict_proba([[2., 2.]]))
```

现在执行列表 3.2 中的代码，你会看到以下输出：

```
predict class of [-1., -1.]:
[0]
predict class of [0.8, 0.8]:
[1]
predict class of [2., 2.]:
[2]
probability of each class in [2.,2.]:
[[0. 0. 1.]]
```

如您所见，点 [-1，-1]、[0.8、0.8] 和 [2，2] 分别被值 0、1 和 2 标记，这很可能也是您所期望的。

列表 3.3 显示数据集 partial_wine.csv 的一部分，其中包含两个特征列和一个标签列（有三个类）。此数据集的总行数为 178。

列表 3.3： partial_wine.csv

```
Alcohol, Malic acid, class
14.23,1.71,1
13.2,1.78,1
13.16,2.36,1
14.37,1.95,1
13.24,2.59,1
14.2,1.76,1
```

列表 3.4 显示了 tree_classifier.py 的内容，该程序使用决策树来训练数据集 partial_wine.csv 上的模型。

列表 3.4： tree_classifier.py

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('partial_wine.csv')
```

```
X = dataset.iloc[:, [0, 1]].values
y = dataset.iloc[:, 2].values

# split the dataset into a training set and a test set
from sklearn.model_selection import train_test_
   split
X_train, X_test, y_train, y_test = train_test_
   split(X, y, test_size = 0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# ====> INSERT YOUR CLASSIFIER CODE HERE <====
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy
   ',random_state=0)
classifier.fit(X_train, y_train)
# ====> INSERT YOUR CLASSIFIER CODE HERE <====

# predict the test set results
y_pred = classifier.predict(X_test)

# generate the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("confusion matrix:")
print(cm)
```

列表 3.4 包含一些 import 语句，然后用 CSV 文件的内容初始化数据类型为 dataframe 的 dataset。接下来，变量使用数据集的前两列（以及所有行）初始化 x，使用数据集的第三列（和所有行）初始化 y。

接下来，使用 75/25 的比例划分 x 和 y，同时用它们来初始化 X_train，X_test，y_train 和 y_test。注意变量 sc（这是

StandardScalar 类的实例）对变量 X_train 和 X_test 执行缩放操作。

列表 3.4 中粗体显示的代码块是其中我们创建决策树分类器类的实例，然后使用变量 X_train 和 X_test 训练它。.

列表 3.4 的下一部分用一组 X_test 的预测值初始化 y_pred。列表 3.4 的最后一部分根据 y_test 和 y_pred 中的数据创建混淆矩阵。

混淆矩阵的所有对角线元素都是正确的预测（如真正的正和真正的负）;所有其他单元格都包含一个数值，该数值指定不正确的预测（如 FP 和 FN）。

现在运行列表 3.4 中的代码，你会看到输出以下的混淆矩阵，其中有 36 个正确的预测和 9 个不正确的预测（正确率为 80%）：

```
confusion matrix:

[[13  1  2]

 [ 0 17  4]

 [ 1  1  6]]

from sklearn.metrics import confusion_matrix
```

前面 3x3 矩阵中共有 45 个项，对角线上是正确的标签。因此，正确率为 36/45 = 0.80。