目录

第 7 章

无监督学习：神经网络工具包

下一个无监督学习技术使得无监督学习过程更进了一步。

我很荣幸向您介绍自动编码器。它们代表着未来我们将如何发现和探索工业化世界中形成的海量数据湖，这些数据湖是由我们每天生成的数据形成的。

神经网络自编码器

自编码器是一种人工神经网络，用于以无监督的方式学习高效的数据编码。自编码器的目的是学习出一组数据的特征；通常被用于降维操作。近年来，自编码器概念已广泛应用于生成模型的学习。在当前商业世界中，一些最著名的 AI 成就就包括了在深度神经网络内部的使用稀疏的自编码器。它是一种用于数据集的非常有用的技术。

打开代码：Chapter-007-001-Neural-NetworksAutoencoder-01.ipynb

这些示例为您提供了一个简单的自编码器，该代码解释了该过程的一些规则，以便您可以将它用作其他自编码问题的基础。

Your input is: [[0 0 1], [0 1 1], [1 0 1], [1 1 1]]

Your required output is: [[0], [0], [1], [1]]

Output after Training: [[0.00966553], [0.00786406], [0.99359009],[0.99211908]]

Output after Stepper function: [[0.], [0.], [1.], [1.]]

基本原则是在步进函数（stepper 函数）之后调用 sigmoid 函数来以获得所需的结果。这些自编码器可以很容易地转移到集成电路（IC）系统中，以帮助边缘处理和传感器问题。

您已经成功地将三个输入减少（自编码）到一个输出.

下一步要研究的是 Chapter-007-002-Neural-Networks-Autoencoder-02.ipynb

此自编码器使用两层来处理输入。请运行程序以观察这一过程。您正在取得巨大进步，现在可以成功执行两个自编码器解决方案

在第 11, 12, 13,14 和 15 章中，我将指导您进一步学习几个示例，并说明如何使用自编码器模型的工业化版本来解决实际问题。

现在，我将指导您了解一项对解决现实方案有用的新技术。接下来，我们将讨论 GANS

生成式对抗网络 （GAN）

生成时对抗网络 （GANs） 是用于无监督机器学习的 AI 算法，它由两个神经网络系统在零和游戏的框架中相互挑战实现。相互竞争的 ML 流程可产生一个快速的合作学习解决方案，轻松将解决方案演变为最佳解决方案。

因此，让我们来讨论零和游戏：

在博弈论和经济理论中，零和博弈是一种情况的数学表示。这种情况为：其中每个参与者的得到或者损失的效用与其他参与者损失或收益的效用完全平衡。如果将参与者的总收益相加并减去总损失，则它们结果为零。

因此，与所爱的人分享一杯咖啡，其中一个人喝了更多的咖啡会减少另一个人喝到的咖啡量，即使每个人同样重视咖啡结果也是零和游戏，换句话说，大家都是幸福的！

接下来 ，我将介绍 PyTorch。

PyTorch 是基于 Python 的科学计算包，它利用图形处理单元的力量来协助处理。

基于 Theano 引擎的 TensorFlow 和使用 Torch 框架的 PyTorch 之间的基本区别是这些框架在硬件中定义计算图形的方式。因此，在多数情况下，torch 框架的速度与 TensorFlow 一样快，但 torch 对于循环神经网络的速度更快。众所周知，TensorFlow 上的 Keras 是较慢的，但它却赢得了比赛。主要原因是，与许多其他目前流行的深度学习框

架相比，TensorFlow 提供了出色的功能和服务。现在，TensorFlow 的 Keras 集成使深度学习部署变得更加容易。因此，许多新项目都使用 TensorFlow 的 Keras 为标准。

您将需要 torch 库供 cpu 使用：

```
conda install -c pytorch pytorch-cpu
```

打开代码：Chapter-007-003-GAN-01.ipynb，然后运行。

您的结果：

```
Using data [Data and variances]
0: D: 0.7086962461471558/0.6972255110740662 G: 0.6868060827255249
(Real:[3.7825968647003174, 1.2327905192370257], Fake:
[0.09708623103797435,0.009256606334190703])
5000: D: 0.6805199980735779/0.6616045236587524 G: 0.8273075819015503
(Real: [4.117388695478439, 1.274467621457445], Fake:
[4.077913434505462,1.0943474129771693])
10000: D: 0.37435632944107056/0.8348380327224731 G: 0.4132739007472992
(Real: [3.9343803453445436, 1.3165514626757275], Fake:
[3.7064030361175537,1.3515258142353932])
15000: D: 0.024186545982956886/2.411825656890869 G: 0.9955140948295593
(Real: [4.019607482552528, 1.3916774260541982], Fake:
[4.100582712888718,1.3490263347503852])
20000: D: 0.8105971813201904/0.08567392826080322 G: 1.3057440519332886
(Real: [3.8651703649759295, 1.291156659436279], Fake:
[4.00323578953743,1.2979101588489939])
25000: D: 0.00010073692101286724/1.0811676474986598e-05
G:12.61683464050293
(Real: [4.023330308198929, 1.1151899002588046], Fake:[14.387419891357421,
0.6251815240638156])
30000: D: 9.059946023626253e-06/-5.972999872483342e-14 G:
26.81680679321289
(Real: [4.0317733579874035, 1.3068207970685468], Fake:
[37.9575638961792,0.9236522286831534])
35000: D: 2.3841761276344187e-07/-9.841016890277388e-13
G:27.20176887512207
```

(Real: [4.068414183855057, 1.3843235577079283], Fake:[62.59007144927978, 1.5635452696977525])

40000: D: 0.0002785713004413992/-1.000088900582341e-12 G: 27.63102149963379

(Real: [4.057045288085938, 1.1723868704134213], Fake: [68.872311668396,8.993019130772057])

45000: D: 2.5033971269294852e-06/-1.000088900582341e-12 G:27.63102149963379

(Real: [3.9334994852542877, 1.190026090616771], Fake:[69.51191040039062, 8.09731438960914])

您正在取得巨大进展，您现在可以成功执行 GAN 解决方案。

在第 11、12、13、14，15 章中，我将指导您学习几个示例，说明如何使用工业化的 GAN 版本来解决实际问题。

注意 生成式对抗网络是一种无监督学习算法，目前正在以指数化的速度发展。它正在医疗保健、金融和智能城市领域不断产生新的知识。

有趣的事实 一个 GAN 系统在 2018 年生成了一幅《爱德蒙·贝拉米的肖像》，卖了 43.25 万美元。这证明机器学习可以轻松学习，甚至掌握和人类一样的交易。

卷积神经网络 （CNN）

在机器学习中，卷积神经网络（CNN 或 ConvNet）是一类深入、前馈的人工神经网络，最常用于分析视觉图像。CNN 使用多层感知器的变体，旨在达到最少的预处理。

ConvNet 或 CNNs 是神经网络的一个类别，已经证明在图像识别和分类等领域非常用。除了应用在机器人和自动驾驶汽车中的动力视觉外，ConvNets 还成功地识别了人脸、物体和交通标志。

打开示例代码：Chapter-007-004A-CNN-01.ipynb 此过程使用传统的 MNIST 数据集生成 CNN。

执行笔记本以获得所需的数据集。

在数据目录中应该有两个名为"Train-28x28_CNN_text.txt 和 28x28_CNN_text.txt 文件。

现在打开代码：Chapter-007-004B-CNN-01.ipynb

A 部分加载所需的数据。

使用独热结构实现检测数字的转换：

label: 0 in one-hot representation: [1 0 0 0 0 0 0 0 0 0]
label: 1 in one-hot representation: 0[0 1 0 0 0 0 0 0 0 0]
label: 2 in one-hot representation: [0 0 1 0 0 0 0 0 0 0]
label: 3 in one-hot representation: [0 0 0 1 0 0 0 0 0 0]
label: 4 in one-hot representation: [0 0 0 0 1 0 0 0 0 0]
label: 5 in one-hot representation: [0 0 0 0 0 1 0 0 0 0]
label: 6 in one-hot representation: [0 0 0 0 0 0 1 0 0 0]
label: 7 in one-hot representation: [0 0 0 0 0 0 0 1 0 0]
label: 8 in one-hot representation: [0 0 0 0 0 0 0 0 1 0]
label: 9 in one-hot representation: [0 0 0 0 0 0 0 0 0 1]

在 B 部分中创建 sigmoid 函数。在 C 部分中，创建神经网络类。

以下命令激活神经网络：

```
ANN = NeuralNetwork(no_of_in_nodes = image_pixels,
                    no_of_out_nodes = 10,
                    no_of_hidden_nodes = 100,
                    learning_rate = 0.1)
```

在 D 部分中训练神经网络。

在 MNIST 数据的 60,000 个图片上完成训练。

在 E 部分中，您可以根据 MNIST 数据集中的 10,000 张图像测试这个神经网络

F 部分显示您得到的结果：

```
Accuracy Train => 0.949
Accuracy Test ==> 0.9477
=======================================================
[[5797 1 43 24 17 46 42 15 15 22]
 [ 1 6633 62 38 13 32 21 62 109 14]
 [ 5 23 5479 43 4 8 3 45 8 2]
 [ 4 23 104 5779 1 87 1 19 82 57]
 [ 10 10 45 7 5505 31 7 48 34 64]
 [ 5 6 5 44 0 4992 43 1 12 5]
 [ 29 2 61 18 67 82 5778 8 36 4]
```

[0 6 40 44 4 9 0 5820 4 28]
[51 11 93 52 4 52 23 10 5427 22]
[21 27 26 82 227 82 0 237 124 5731]]
==========

Digit: 0 Precision: 0.9787 Recall: 0.9626
Digit: 1 Precision: 0.9838 Recall: 0.9496
Digit: 2 Precision: 0.9196 Recall: 0.9749
Digit: 3 Precision: 0.9426 Recall: 0.9386
Digit: 4 Precision: 0.9423 Recall: 0.9556
Digit: 5 Precision: 0.9209 Recall: 0.9763
Digit: 6 Precision: 0.9763 Recall: 0.9495
Digit: 7 Precision: 0.929 Recall: 0.9773
Digit: 8 Precision: 0.9275 Recall: 0.9446
Digit: 9 Precision: 0.9634 Recall: 0.874

G 部分令 epoch=5 并进行测试。

Epoch: 0
Accuracy Train=> 0.9457
Accuracy Test==> 0.9462
Epoch: 1
Accuracy Train=> 0.9607
Accuracy Test==> 0.956
Epoch: 2
Chapter 7 Unsupervised Learning: Neural Network Toolkits
213
Accuracy Train=> 0.9684
Accuracy Test==> 0.9603
Epoch: 3
Accuracy Train=> 0.9736
Accuracy Test==> 0.963
Epoch: 4
Accuracy Train=> 0.9726
Accuracy Test==> 0.9613

这表明 CNN 是稳定的，并且达到了 > 94% 的精度。

循环神经网络（RNNs）

循环神经网络 （RNN） 是人工神经网络的一类，其中节点之间的连接沿着结点的序列形成定向图。这允许它显示时间序列的动态行为。与前馈神经网络不同，RNN 可以使用其内部的状态处理输入序列。这让他们能够记住它以前做了什么， 以及他们如何适应这些知识。

这使得它们适用于未分段、连续的手写识别或语音识别等任务。

打开代码：Chapter-007-005-RNN-01.ipynb

E 部分是主引擎并得到以下结果：

```
================================================
Cycle: 0
================================================
Error:[0.3217455]
Pred:[1 0 0 0 1 0 0 1]
True:[1 0 0 0 1 0 0 1]
108 + 29 = 137
================================================
================================================
Cycle: 1000
================================================
Error:[0.19121145]
Pred:[1 0 1 1 1 1 0 1]
True:[1 0 1 1 1 1 0 1]
76 + 113 = 189
================================================
================================================
Cycle: 2000
================================================
Error:[0.12260766]
Pred:[0 0 1 0 1 0 1 0]
True:[0 0 1 0 1 0 1 0]
10 + 32 = 42
================================================
================================================
Cycle: 3000
```

==========
Error:[0.25359378]
Pred:[1 0 0 1 0 0 0 1]
True:[1 0 0 1 0 0 0 1]
108 + 37 = 145
==========
==========
Cycle: 4000
==========
Error:[0.27286803]
Pred:[1 0 0 0 1 0 1 0]
True:[1 0 0 0 1 0 1 0]
123 + 15 = 138
==========
==========
Cycle: 5000
==========
Error:[0.27208995]
Pred:[1 1 0 1 0 0 0 1]
True:[1 1 0 1 0 0 0 1]
83 + 126 = 209
==========
==========
Cycle: 6000
==========
Error:[0.17898913]
Pred:[1 1 0 1 0 1 0 0]
True:[1 1 0 1 0 1 0 0]
112 + 100 = 212
==========
==========
Cycle: 7000
==========
Error:[0.17206773]
Pred:[0 1 0 1 1 0 0 1]

True:[0 1 0 1 1 0 0 1]
79 + 10 = 89
==========
==========
Cycle: 8000
==========
Error:[0.21744059]
Pred:[1 0 1 0 1 1 1 1]
True:[1 0 1 0 1 1 1 1]
82 + 93 = 175
==========
==========
Cycle: 9000
==========
Error:[0.18515105]
Pred:[0 1 0 0 0 1 1 1]
True:[0 1 0 0 0 1 1 1]
51 + 20 = 71
==========
==========
Cycle: 10000
==========
Error:[0.12941404]
Pred:[0 1 1 1 1 1 1 0]
True:[0 1 1 1 1 1 1 0]
76 + 50 = 126
==========
==========
Cycle: 11000
==========
Error:[0.22409284]
Pred:[1 1 1 1 0 0 0 0]
True:[1 1 1 1 0 0 0 0]
118 + 122 = 240
==========

```
================================================
Cycle: 12000
================================================
Error:[0.16455129]
Pred:[1 0 1 0 1 1 0 0]
True:[1 0 1 0 1 1 0 0]
49 + 123 = 172
================================================
================================================
Cycle: 13000
================================================
Error:[0.20210813]
Pred:[1 0 1 1 1 0 0 0]
True:[1 0 1 1 1 0 0 0]
106 + 78 = 184
================================================
================================================
Cycle: 14000
================================================
Error:[0.10140656]
Pred:[0 1 0 1 1 0 1 1]
True:[0 1 0 1 1 0 1 1]
18 + 73 = 91
================================================
================================================
Cycle: 15000
================================================
Error:[0.17551984]
Pred:[1 0 0 0 1 0 0 0]
True:[1 0 0 0 1 0 0 0]
111 + 25 = 136
================================================
```

你能观察状态是如何从一个周期转移到下一个周期的吗？

请注意　在现代图像识别和描述工作中使用的 LSDM（长期短期记忆）神经网络，通过将 RNN 与 Convnet 结合在一起进行识别和分类。

谱双聚类算法

双聚类、块聚类、联合聚类、或双模式聚类是同时对聚类矩阵的行和列进行数据挖掘技术。

数据使用函数 make_checkerboard 生成，然后随机处理并传递到谱双聚类算法中。重新排列洗牌后的矩阵的行和列，以显示算法找到的双集群。

打开代码：Chapter-007-006-Spectral-Biclustering-01.ipynb。

核心引擎是：

```
model = SpectralBiclustering(n_clusters=n_clusters, method='log', random_state=1968)
```

您的处理结果如图 7-1 和 7-2 所示。

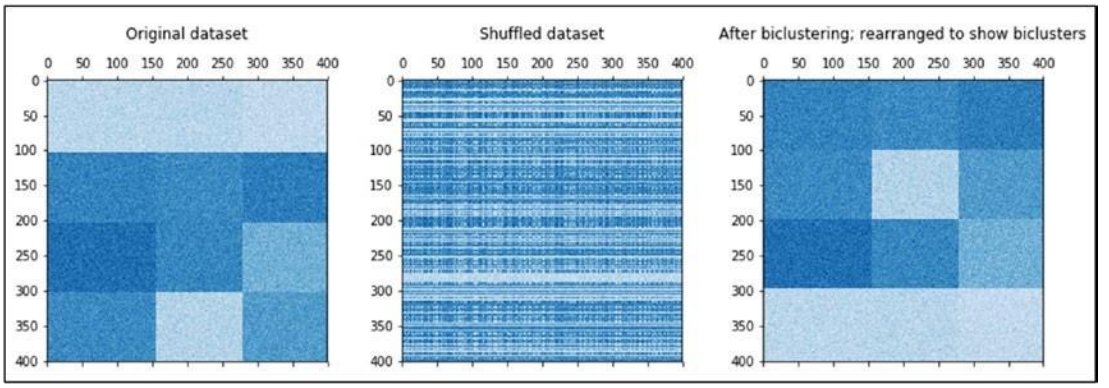

图 7-1。双集群结果

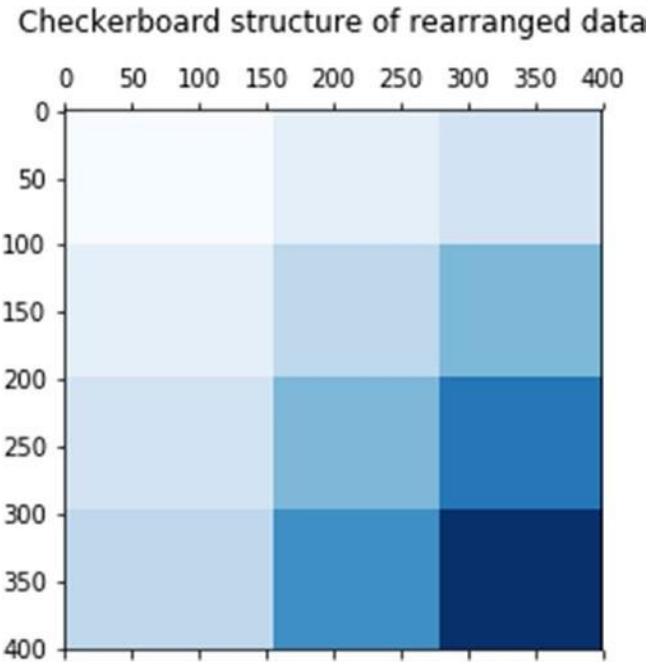

图 7-2。 双聚类最终结果

您可以清楚地看到集群之间的分类，这在执行图像分类时非常有用。

BIRCH 聚类算法

BIRCH 算法（利用层次方法的平衡迭代规约和聚类）是一种无监督的数据处理算法，主要用于在大型数据集上完成层次聚类。

我们的示例比较了 BIRCH（有没有全局聚类步骤）和 MiniBatchKMeans 在相同数据集上的时间，这一数据集由 make_blobs 生成，包括 100,000 个样本以及两个特征。

如果 n_clusters 设置为"None"，则数据从 100,000 个样本减少到 158 个集群。这可视为在最终（全局）聚类步骤之前的一个预处理，比如可以提前将这 158 个集群减小到 100 个集群。

打开 Chapter-007-007-Birch-clustering-01.ipynb。

结果如图 7-3 所示。

Birch without global clustering as the final step took 2.08 seconds

n_clusters : 158

Birch with global clustering as the final step took 2.06 seconds
n_clusters : 100
Time taken to run MiniBatchKMeans 2.62 seconds

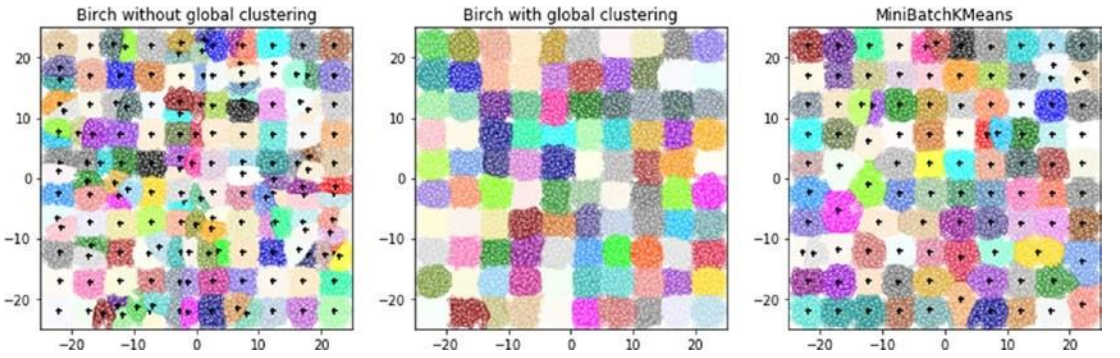

图 7-3。 BIRCH 聚类结果

机器学习工具包

机器学习工具包是一组用于数据分析的方法，通过支持一组无缝协作的预调技术来自动化分析模型构建。

Scikit-learn

Scikit-learning（以前的 scikits.learn）是 Python 的一个开源机器学习库。它具有各种分类、回归和聚类算法，包括支持向量机、随机森林、梯度提升、K-Means 和 DBSCAN。

你可能在本书的前一部分使用过这个库

安装：

```
conda install -c conda-forge scikit-learn
```

这是一个使用 scipy 中的 face 数据集的例子：Chapter-007-008-scikit-learn-01.ipynb

Keras

Keras 是一个用 Python 编写的开源神经网络库。它可以在 TensorFlow、微软认知工具包（Microsoft Cognitive Toolkit）或 Theano 上运行。它旨在实现快速使用深度神经网络，并且专注于用户友好、模块化和扩展性。

安装：

conda install -c conda-forge keras

pip install tensorflow

conda install -c conda-forge tensorflow

打开代码：Chapter-007-009A-Keras-01.ipynb

提示 使用 TensorFlow 上的 Keras 会工作得更好，它可以展示 Keras 和 TensorFlow 的力量供您使用。

您的结果：

0 -> 0.9875 => 1 => 1 # Yes

1 -> 0.0170 => 0 => 0 # Yes

2 -> 0.9954 => 1 => 1 # Yes

3 -> 0.0006 => 0 => 0 # Yes

4 -> 0.9954 => 1 => 1 # Yes

5 -> 0.0113 => 0 => 0 # Yes

6 -> 0.9773 => 1 => 1 # Yes

7 -> 0.0070 => 0 => 0 # Yes

8 -> 0.0000 => 0 => 1 # No

9 -> 0.9984 => 1 => 1 # Yes

您可以观察到，这种 Keras 和 TensorFlow 组合是对 IML 生态系统产生重大影响的一种模式。

XGBoost

Python 的可扩展、可移植和分布式梯度提升（GBDT、GBRT 或 GBM）库包括了常见的机器学习解决方案。

XGBoost 是一个增强的分布式梯度提升库，设计高效、灵活且可移植。它在梯度提升框架下实现机器学习算法。 XGBoost 提供并行树提升（也称为 GBDT、GBM），可快速准确地解决多个数据科学问题。类似的代码在主流分布式环境（Hadoop、SGE 和 MPI）上运行，可以解决数十亿条记录上的问题。

安装库：

conda install -c conda-forge xgboost

打开代码：Chapter-007-010- xgboost-01.ipynb

结果：

0 -> [9.0, 102.0, 76.0, 37.0, 0.0, 32.9, 0.665, 46.0] => 0 => 1 # No

1 -> [9.0, 119.0, 80.0, 35.0, 0.0, 29.0, 0.263, 29.0] => 1 => 1 # Yes

2 -> [7.0, 159.0, 64.0, 0.0, 0.0, 27.4, 0.294, 40.0] => 1 => 0 # No

3 -> [7.0, 106.0, 92.0, 18.0, 0.0, 22.7, 0.235, 48.0] => 0 => 0 # Yes

4 -> [2.0, 71.0, 70.0, 27.0, 0.0, 28.0, 0.586, 22.0] => 0 => 0 # Yes

---

提示 XgBoost 目前是在快速获得数据集的结果方面享有很好的声誉的 ML 算法之一。我个人经常使用它。

在更广泛的领域中， XgBoost 是任何数据科学实践中使用最多的算法之一。这是您将使用的算法之一！

StatsModels

StatsModels 是一个 Python 模块，它提供用于估计许多不同统计模型以及进行统计测试和统计数据挖掘的类和功能。

安装库：

conda install -c conda-forge statsmodels

打开代码： Chapter-007-011-statsmodels-01.ipynb

执行此代码以查看此库的各种功能。

提示 statsModel 库为大多数统计人员进入机器学习生态系统提供了过渡阶段。

LightGBM

LightGBM 是一个梯度提升框架，使用基于树的学习算法。

安装库：

conda install -c conda-forge lightgbm

打开示例代码：Chapter-007-012-LightGBM-01.ipynb

执行代码。

结果：

```
Confusion Matrix

[[64 4]

 [ 5 27]]

====================

Accuracy: 0.91
```

该库总体表现良好，我预计它在未来五年内将进一步演变为您应该知道的十种算法之一。

CatBoost

决策树上的梯度提升库。

"CatBoost"名称来自两个词"类别（Category）"和"提升（Boosting）"。

安装库：

```
conda install -c conda-forge catboost
```

打开示例代码：Chapter-007-013-CatBoost-01.ipynb

执行这个代码。

结果：

```
bestTest = 0.8358208955
bestIteration = 428
```

这证明该算法在 500 个周期中的第 428 个周期中找到了最佳结果。这几乎节省了 20%的过程 。

这是机器学习工具包的结尾。掌握这些知识将为您将来处理新的数据提供良好的服务。

接下来是什么？

您现在已成功完成无监督机器学习的第 2 部分。

本章中最主要的两个内容是学习如何使用 TensorFlow 和 XGBoost。

使用某种形式的神经网络算法现在是一种必须掌握的技能。因此，请确保您了解最后两章，因为这是机器学习工作的提升最大的地方。

您现在已经了解了最常见的机器学习类型。下一章是第 3 部分，我将向您介绍深度学习算法。