目录

第 8 章

无监督学习：深度学习

下一组无监督学习技术将为无监督过程引入神经网络。

人工神经网络（ANN）是一种生态系统，通过模仿类似于大脑中的激励过程来得到解决方案，即具有共同结果的一系列激励过程。

深度学习

深度学习是机器学习的一个子领域，使用算法来生成人工神经网络。

TensorFlow

TensorFlow （https://www.tensorflow.org/）是一个使用数据流图进行数值计算的开源软件库。图形中的节点表示数学运算，图形边表示在它们之间流转的多维数组（张量）。灵活的体系结构允许您使用单个"API"将计算部署到桌面、服务器或移动设备中的一个或多个"CPU"或"GPU"。

您可以使用以下任何一项来安装：

conda install -c anaconda tensorflow (I used this one on my laptop)

conda install -c conda-forge tensorflow

conda install -c anaconda tensorflow-gpu (I use this in my

GPU-enabled Desktop Workstation)

您在前几章中已经使用过此处理。

打开示例代码：Chapter-008-001-Tensorflow-01.ipynb

使用 Tensorflow 运行以上代码来求解数学公式 y = m * x + k

结果：y = (0.196810 * x) + 0.201904

处理通过尝试与训练数据集匹配来学习 m 和 b 的值。

打开示例代码：Chapter-008-002-Tensorflow-02.ipynb

警告 在 Mac 上，加载 mnist 数据集时存在证书验证失败的问题。您需要在 Python 安装中执行"/Applications/Python/3.6/ Install/Certificates.command"来解决此问题。

您的结果： Loss:0.149082，accuracy:0.957300。误差为 14.9%，精度 95.7%是一个很好的结果，说明这个过程运行的很好。

PyTorch

PyTorch 是 Python 的开源机器学习库，基于 Torch，用于自然语言处理等应用程序。

PyTorch 是一个优化的 tensor 库，用于使用 GPU 和 CPU 进行深度学习。

PyTorch 通过混合前端、分布式训练以及工具和库生的态系统实现快速、灵活的实验和高效的生产过程。

请参阅：https://pytorch.org/ 以了解更多详细信息。

安装：

conda install pytorch torchvision cudatoolkit=10.0 -c pytorch

如果您只想或只能使用 cpu 则执行：conda install pytorch-cpu torchvision-cpu -c pytorch

现在打开代码：Chapter-008-003-Torch-01.ipynb

我们将通过比较 NumPy 和 Pytorch 向您介绍机器学习技术。

NumPy 提供了一个 n 维数组对象和用于操作这些数组的许多函数。

一方面，NumPy 是科学计算的通用框架;它对于计算图、深度学习或梯度一无所知。另一方面此外，通过使用 NumPy 操作手动实现向前和向后传输，我们可以轻松地用随机数据拟合双层网络。

PyTorch 是 Python 的开源机器学习库，基于支持两种处理类型的 Torch 生态系统：

- 具有强 GPU 加速的张量计算（与 NumPy 进行对比）。
- 基于磁带的自动微分系统构建的神经网络。

因此，我建议我们调查它比只使用 NumPy 的解决方案好多少。

我已经把最大值设置为 1968， 然后两次运行 19680。

Torch（10.753 秒）在 1968 条记录上比 NumPy（14.803 秒）快 1.38 倍。

Torch（104.135 秒）在 19680 条记录上比 NumPy（149.317 秒）快 1.43 倍。

提示 上一例子证明 Torch 平均比 numPy 强 40% 。当您启用 GPUs 时，这个数字会更高。

Theano

Theano 是一个 Python 库，允许您定义，优化和高效地评估涉及多维数组的数学表达式。这是用于评估 CPU 和 GPU 上的数学表达式的优化编译器。该引擎自 2007 年投入使用，因此在公司中得到广泛使用。核心开发者蒙特利尔学习算法研究所（MILA）于 2017 年 9 月通知机器学习社区，主要开发将停止。

此后，我协助多家客户将其 Theano 解决方案转换为 Keras 和 TensorFlow 解决方案。

安装：

conda install -c conda-forge theano

打开示例代码： Chapter-008-004-Theano.ipynb

运行代码。这将向您展示 Theano 引擎如何执行机器学习。

您的结果应如下：

print('Difference: ')

print(result[0])

Difference:

[[1. 0.]

 [-1. -2.]]

print('Absolute Difference: ')

print(result[1])

Absolute Difference:

[[1. 0.]

 [1. 2.]]

print('Squared Difference: ')

print(result[2])

Squared Difference:

[[1. 0.]

 [1. 4.]]

```
print('Powered Difference: ')
print(result[3])
[[ 1. 1. ]
 [-1. 0.25]]
```

提示：我建议您通过这一引擎收集源数据，以确保您了解成功使用此引擎所需的技术。

我还建议您花一些时间研究这一引擎，以便维护现有代码或将代码移植到另一个引擎。

集群比较

在以下示例集群中的比较将测试您对整个聚类过程的理解。

打开示例代码：Chapter-008-005-Compare-clustering-01.ipynb

结果（图 8-1）显示了数据配置文件的最细微的变化如何在同一机器学习算法和超参数上具有不同的结果。

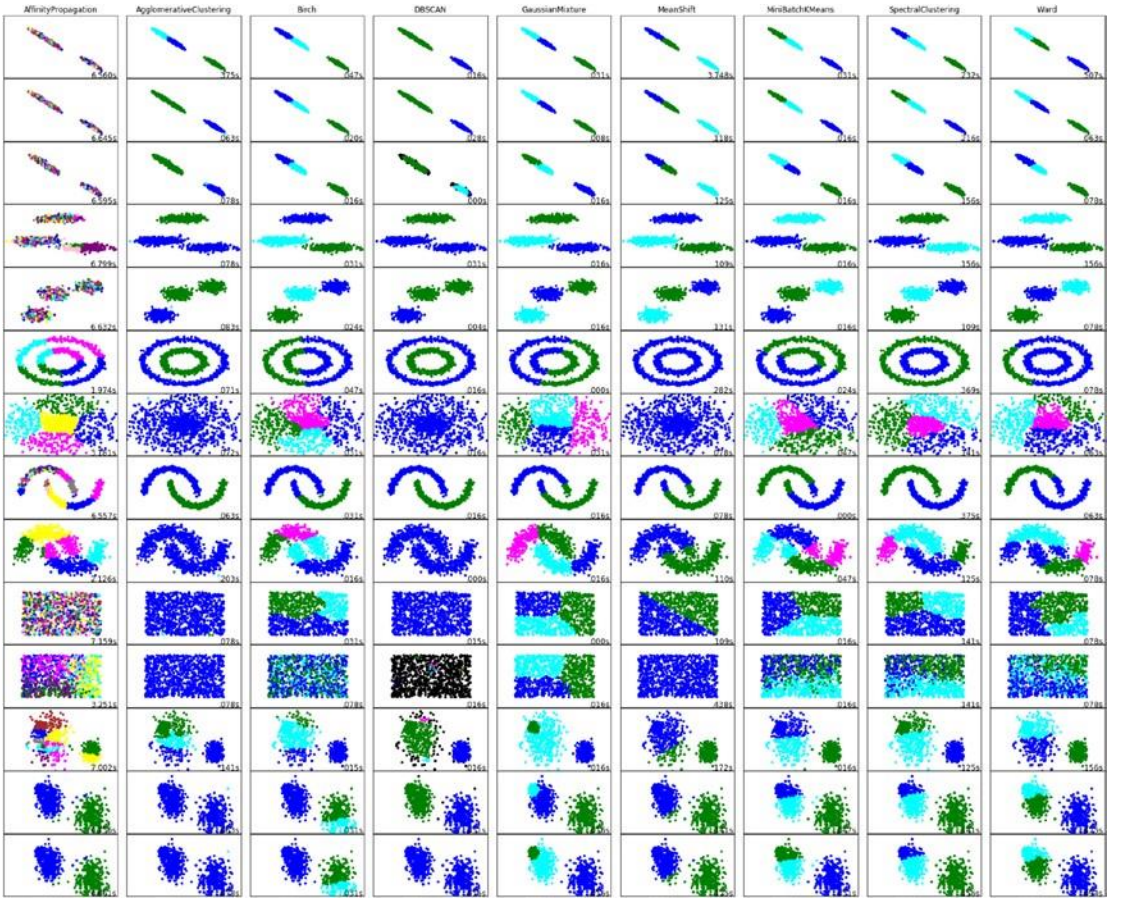

图 8-1. 集群比较结果

提示 通过创建具有不同数据模型的标准测试台来调查机器学习算法及其超参数，以确保您了解机器学习的行为。

结果如图 8-1 所示。

预处理数据集

机器学习系统中耗时最大的过程是是数据预处理，这一步的目的是匹配机器学习解决方案所需的模型或功能。

典型的数据科学和机器学习解决方案需要三到四个人为每位机器学习工程师一小时的工作执行数据准备。

根据我的经验，一个初创公司团队将每周花四天时间准备数据来完成一天的机器学习。

警告 我建议您每天使用轻量级开发方法运行您的 ML，以确保您从 ML 开发中获得正确的结果，同时定期与业务领域人士一起检查结果，以确保我们跟踪您的结果，了解业务变化的最新情况。

您需要了解的下一个比率是，数据预处理将在轻量级任务中消耗 80% 的开发项目。ML 组件通常已经开发，只需要适当的超参数。

因此，在数据工程人员与数据科学人员之间，以 1 比 4 的比例规划您的项目，并预计完成 1 比 9 的工作任务。

现在，让我们开始我们的基本预处理任务。

数据预处理

打开代码：Chapter-008-006-Preprocessing-Data-01.ipynb。 部件 C 是一个缩放处理过程:

此处理过程根据方差和均值标准化数据集。

```
X_scaled = preprocessing.scale(X_train, axis=0, with_mean=True, with_std=False, copy=True )
```

D 部分是一个具有鲁棒性的缩放处理器:

此处理器根据中位数和四分位数标准化数据集。

```
scaler = preprocessing.RobustScaler(with_centering=True, with_scaling=True,
quantile_range=(25.0, 75.0), copy=True ).fit(X_train)
```

sklearn.preprocessing 库有几个有用的预处理引擎。我建议你试用所有引擎以确定哪一个支持你的预处理要求。

Processor	Description
preprocessing.Binarizer([threshold, copy])	Binarize data (set feature values to 0 or 1) according to a threshold.
preprocessing.FunctionTransformer([func, ...])	Constructs a transformer from an arbitrary callable.
preprocessing.KBinsDiscretizer([n_bins, ...])	Bin continuous data into intervals.
preprocessing.KernelCenterer()	Center a kernel matrix.
preprocessing.LabelBinarizer([neg_label, ...])	Binarize labels in a one-vs.-all fashion.
preprocessing.LabelEncoder	Encode labels with value between 0 and n_classes-1.
preprocessing.MultiLabelBinarizer([classes, ...])	Transform between iterable of iterables and a multi-label format.
preprocessing.MaxAbsScaler([copy])	Scale each feature by its maximum absolute value.
preprocessing.MinMaxScaler([feature_range, copy])	Transforms features by scaling each feature to a given range.
preprocessing.Normalizer([norm, copy])	Normalize samples individually to unit norm.
preprocessing.OneHotEncoder([n_values, ...])	Encode categorical integer features as a one-hot numeric array.
preprocessing.OrdinalEncoder([categories, dtype])	Encode categorical features as an integer array.
preprocessing.PolynomialFeatures([degree, ...])	Generate polynomial and interaction features.
preprocessing.PowerTransformer([method, ...])	Apply a power transform featurewise to make data more Gaussian-like.

preprocessing.QuantileTransformer([...])	Transform features using quantiles information.
preprocessing.RobustScaler([with_centering, ...])	Scale features using statistics that are robust to outliers.
preprocessing.StandardScaler([copy, ...])	Standardize features by removing the mean and scaling to unit variance.
preprocessing.add_dummy_feature(X[, value])	Augment data set with an additional dummy feature.
preprocessing.binarize(X[, threshold, copy])	Boolean thresholding of array-like or scipy.sparse matrix.
preprocessing.label_binarize(y, classes[, ...])	Binarize labels in a one-vs.-all fashion.
preprocessing.maxabs_scale(X[, axis, copy])	Scale each feature to the [-1, 1] range without breaking the sparsity.
preprocessing.minmax_scale(X[, ...])	Transforms features by scaling each feature to a given range.
preprocessing.normalize(X[, norm, axis, ...])	Scale input vectors individually to unit norm (vector length).
preprocessing.quantile_transform(X[, axis, ...])	Transform features using quantiles information.
preprocessing.robust_scale(X[, axis, ...])	Standardize a data set along any axis.
preprocessing.scale(X[, axis, with_mean, ...])	Standardize a data set along any axis.
preprocessing.power_transform(X[, method, ...])	Power transforms are a family of parametric, monotonic transformations that are applied to make data more Gaussian-like.

这只是标准集。我还发现，具有菊花链拓扑的预处理器通常可以额外改进数据，同时改进最终 ML 模型的性能和调优功能的结果。

提示：建议您花几周时间实验已知数据集与这些预处理器，以了解它们如何转换和修改您的 ML 结果。在现实世界中，我发现使用适当的预处理器可以节省数小时的复杂模型调参时间。

因此，让我们看一个真实的数据集，以演示一些你可以实现的有趣见解。

打开示例代码：Chapter-008-007-Preprocessing-Data-02.ipynb

运行代码并跟踪这一过程。基本概念是从给定的数据集中直观地了解两个选定特征之间的关系和分布。我在许多数据集上运行此过程，以获得有关数据分布和特征的见解。

我们开始吧. . . 让我们来调查一下数据。参见图 8-2。

make_plot(1)

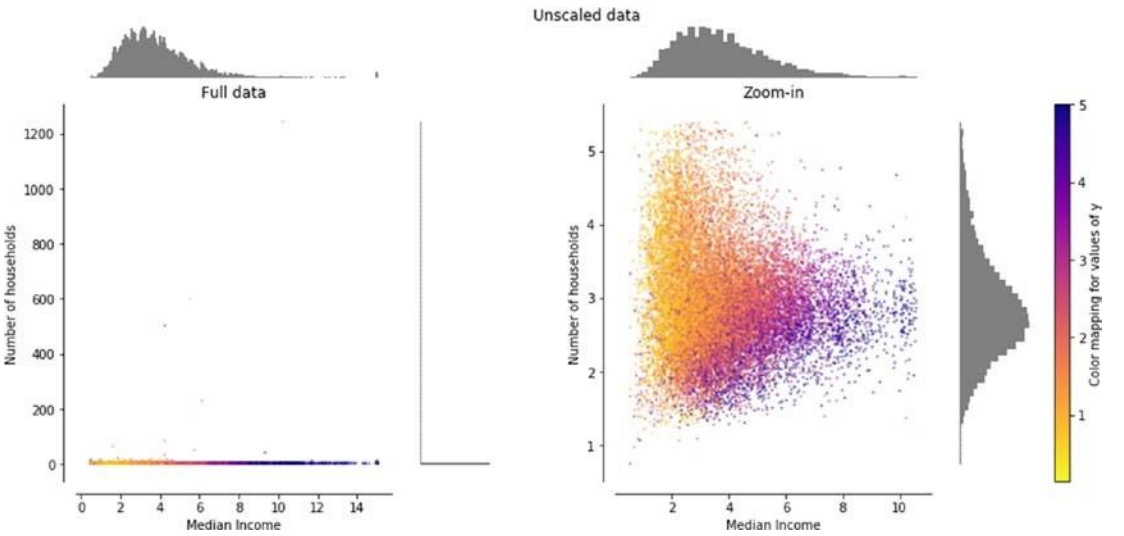

图8-2。 预处理第 0 次结果

请注意，家庭数量都分布在价值轴的下部 。

如果应用标准缩放（standard scaler），您将观察到更好的分布。参见图 8-3。

make_plot(1)

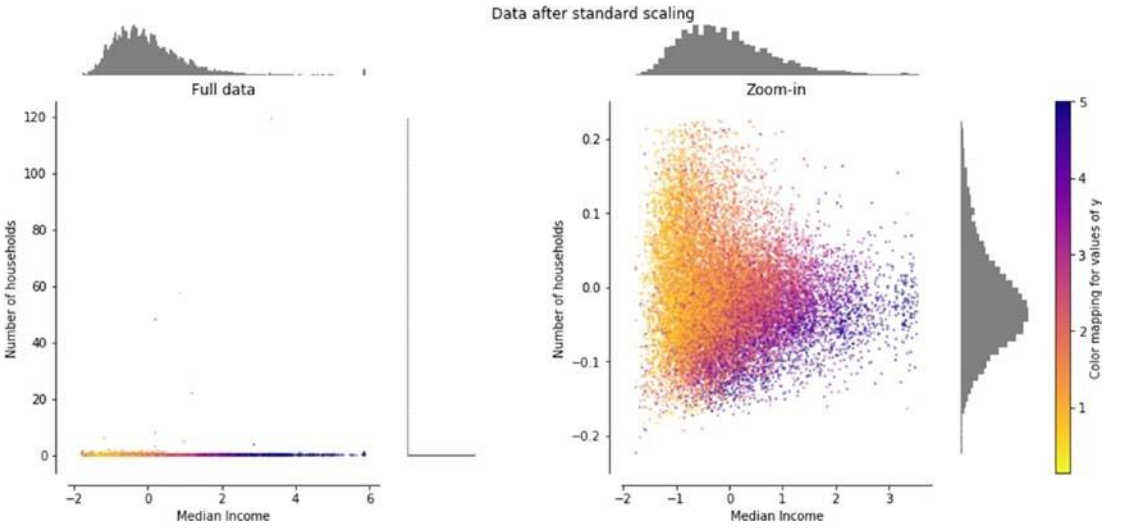

图 8-3. 预处理第一次结果

如果应用最小-最大缩放（Min-Max Scaler,），则发现数据主要分布在均值周围。参见图 8-4。

make_plot(2)

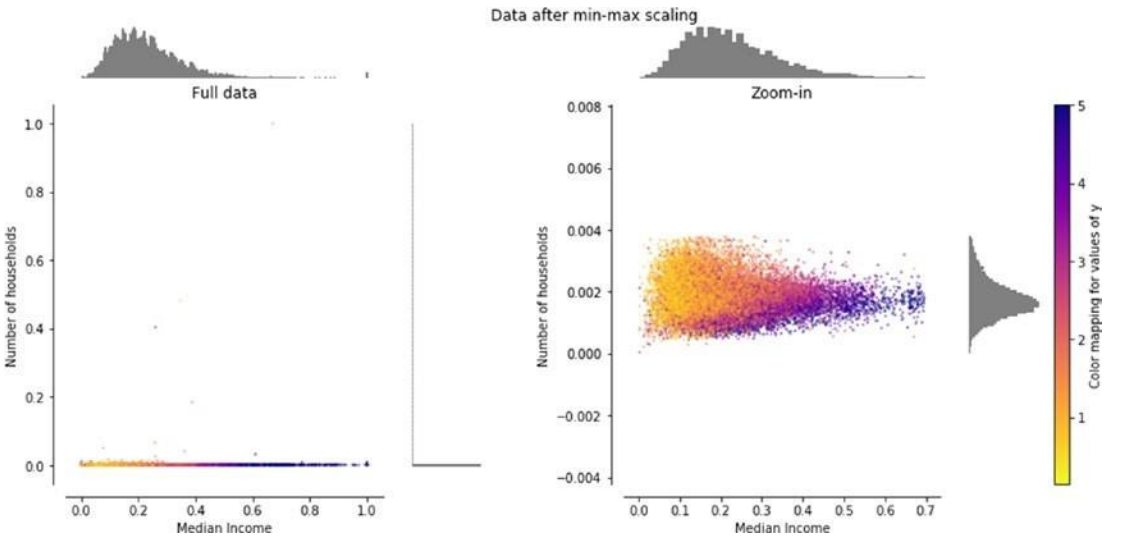

图 8-4. 预处理第二次结果

Min-Abs 处理器提供如下分布。参见图8-5。

make_plot(3)

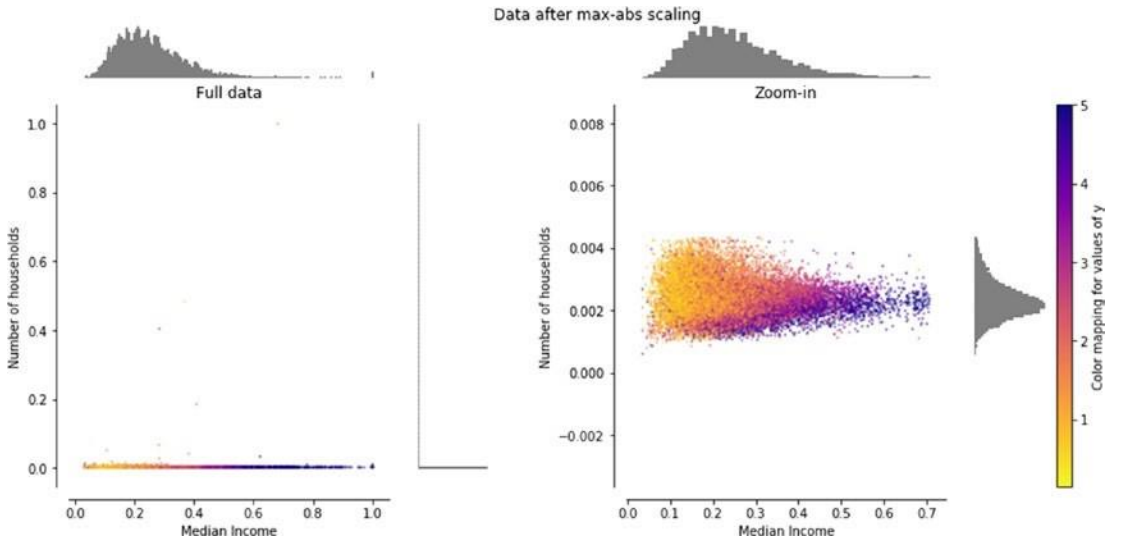

图8-5。 预处理第3次运行 结果

鲁棒性缩放器（robust Scaler）提供此分布。参见图 8-6。

make_plot(4)

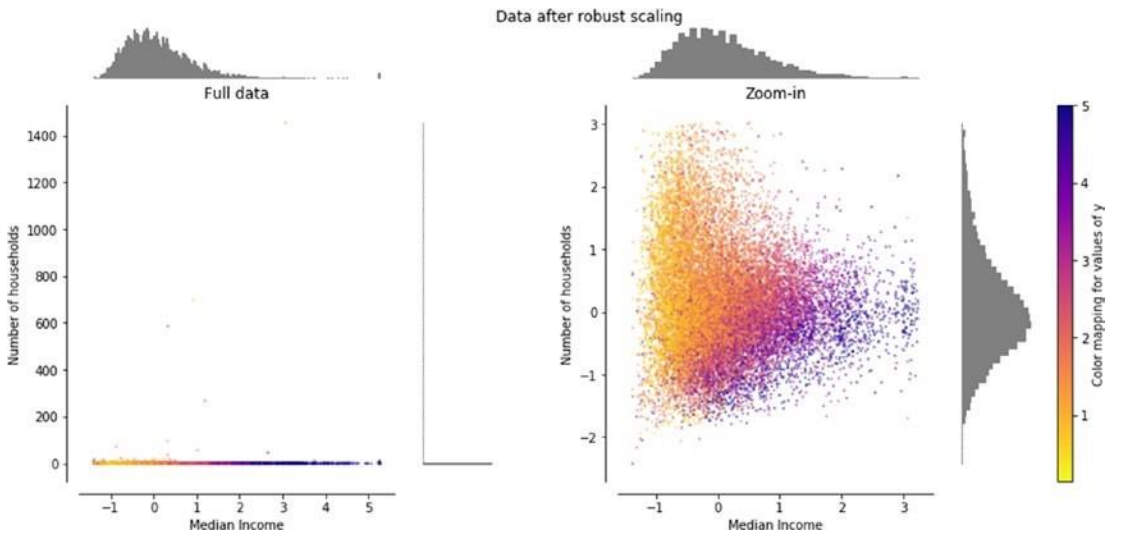

图 8-6. 预处理第 4 次运行结果

有关更多结果，请参阅图 8-7。

make_plot(5)

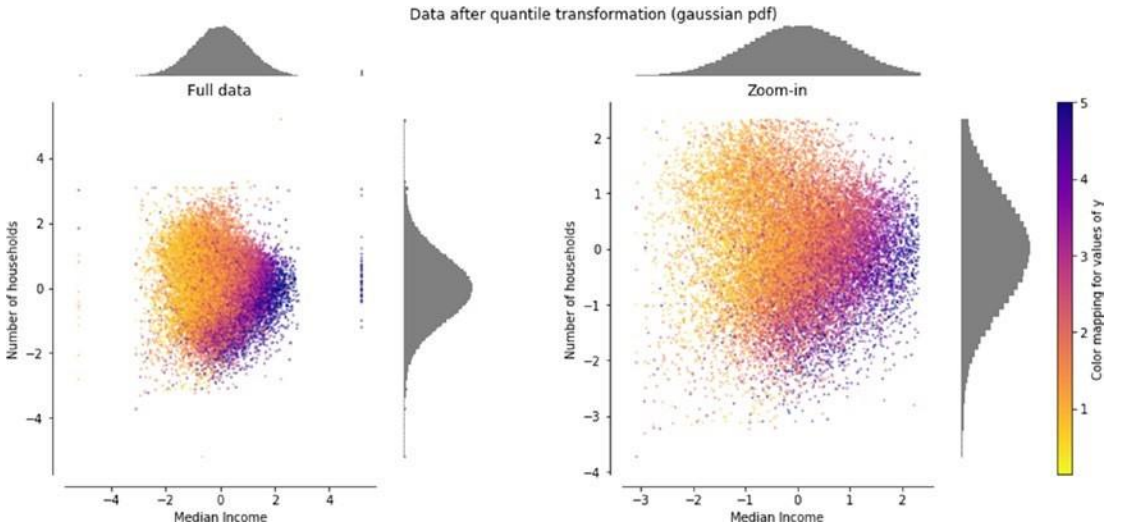

图8-7。 预处理第 5 次运行结果

有关更多结果，请参阅图 8-8。

make_plot(6)

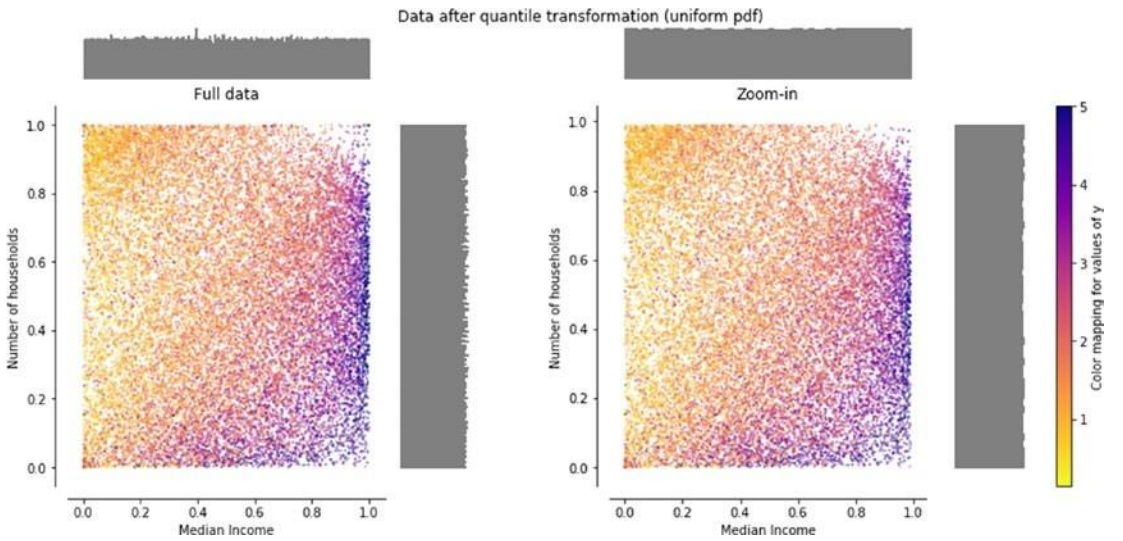

图8-8。 预处理第 6 次运行结果

有关更多结果，请参阅图 8-9。

make_plot(7)

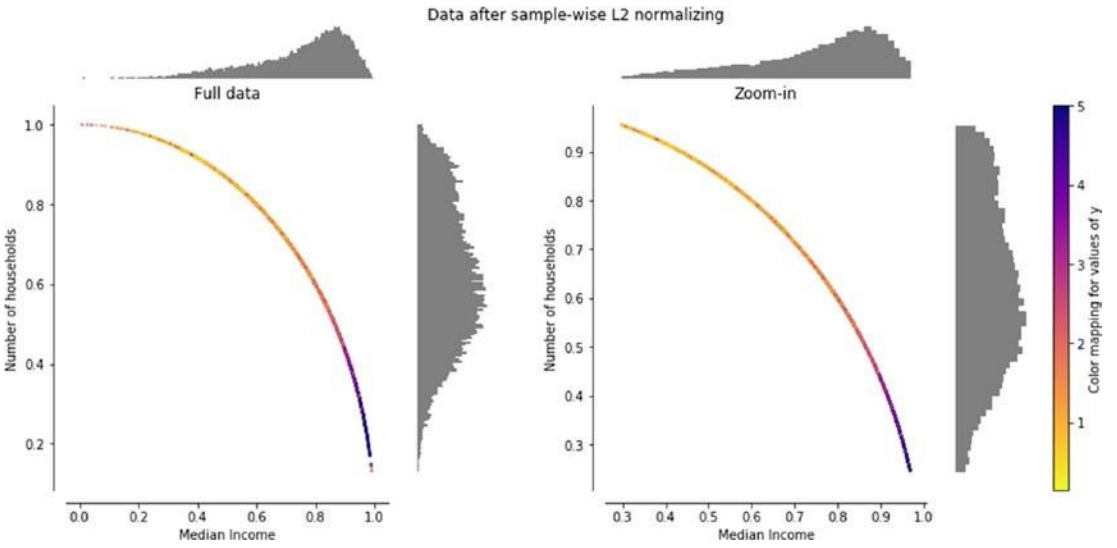

图8-9。 预处理第 7 次运行结果

特征

特征是机器学习过程的支柱；如果您无法识别数据中的模式，也就是无法判断特征，则您将无法很好地执行机器学习这一过程。我建议你学习一下代码，以便深入了解特征的作用以及可以执行的技术，以增强特征这些能力。打开以下代码：

- Chapter-008-008-Features-01.ipynb
- Chapter-008-009-Model-Bagging.ipynb
- Chapter-008-010-Model-Boosting.ipynb
- Chapter-008-011-Model-Stacking.ipynb

应用

我将指导您完成一个无监督学习应用程序，并且演示如何使用您获得的知识来解决现实生活中的问题。

股市

该应用程序是一个股票市场交易公司，交易特定的蓝筹股客户投资组合。你将挖掘特定的蓝筹股，并跟踪其价值。

打开代码：Chapter-008-012-Stock-Market.ipynb。见图 8-10.

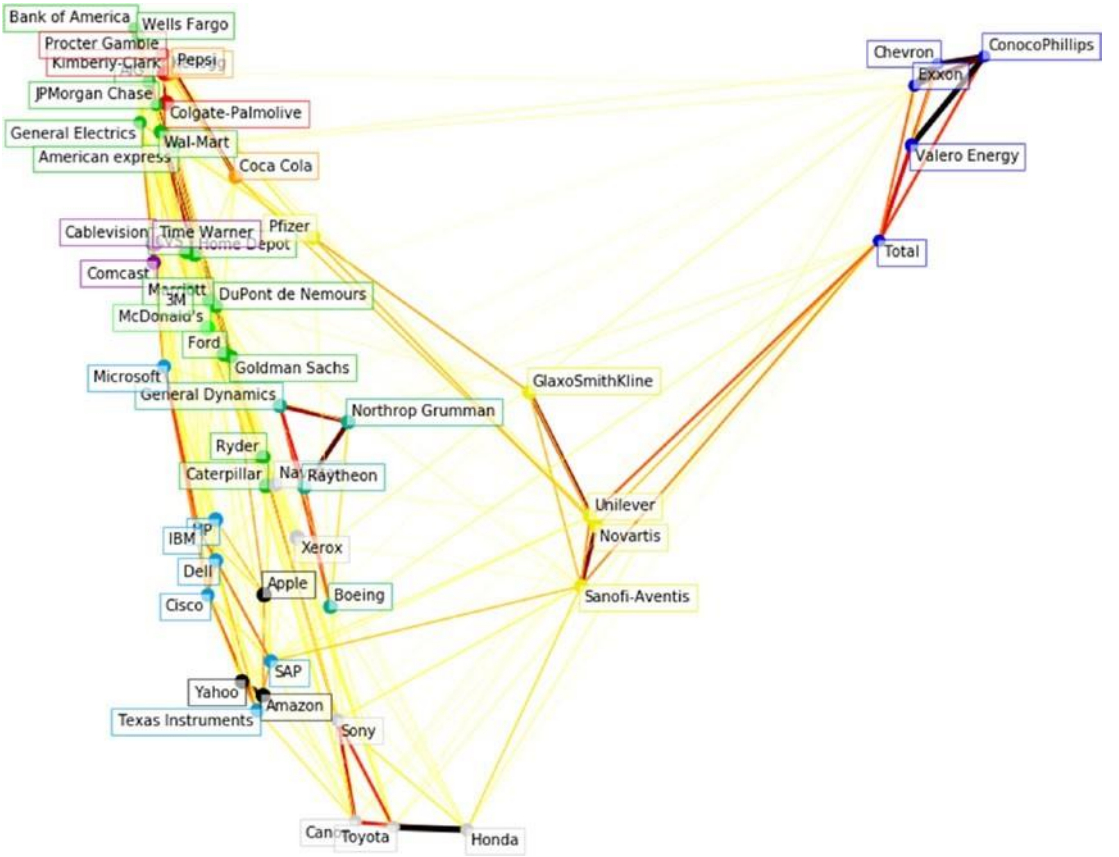

图8-10。 股票市场结果

进展良好！您已完成股票市场解决方案。

警告：ML 建议需要向读者提供大量咖啡和茶。

现在，我将分享一些非机器学习工具，您可以使用这些工具实现机器学习的工业化。请记住，机器学习是决策科学系统 的一部分，外部研究领域的任何解决方案都适用于您的数据科学。

提示：建议您研究适用于决策科学、运营研究和业务管理的解决方案，以便找到适合您的 ML 解决方案的正确解决方案。

接下来是什么？

现在，您已成功完成无监督的机器学习。现在，您已准备好轻松处理任何数据。由于您现在无需任何事先标记就可以处理数据，因此无监督的方法将支持任何数据要求。

您现在已经学习了最常见的机器学习类型，您做得很好。下一章将介绍强化学习。