「Python数据分析系列」11.机器学习

大数据与人工智能 4月27日

以下文章来源于数据与智能 ， 作者Joel Grus

来源 | Data Science from Scratch， Second Edition

作者 | Joel Grus

译者 | cloverErna

校对 | gongyouliu

编辑 | auroral-L

全文共5364字，预计阅读时间45分钟。

第十一章 机器学习

> 耳提面命诚可贵，求知若渴价更高。
>
> ——温斯顿 · 丘吉尔

在很多人眼里，数据科学几乎就是机器学习，而数据科学家每天做的事就是建立、训练和调整机器学习模型（而且，这些人当中有很大一部分并不真正知道机器学习是什么）。事实上，数据科学的主要内容是把商业问题转换为数据问题，然后收集数据、理解数据、清理数据、整理数据格式，而后才轮到机器学习这一后续工作。尽管如此，机器学习也是一种有趣且必要的后续工作，为了做好数据科学工作，你很有必要学习它。

11.1 建模

在讨论机器学习之前，我们需要谈谈**模型**（model）。

什么是模型？它实际上是针对存在于不同变量之间的数学（或概率）联系的一种规范。

比如，如果你想为你的社交网站融资，可以建立一个商业模型（大多数情况下建立在一个工作表里），模型的输入是诸如“用户数”“每位用户的广告收入”“雇员数”之类的变量，输出是接下来几年的年度利润。某本烹调指南涉及的模型是输入“吃饭的人数”和“饥饿的程度”来量化所需要的材料。如果你在电视上看过扑克比赛，就会知道选手们通过一个记牌的模型来实时地估计每位玩家的“获胜概率”，这个模型考虑了已经出的牌和还在牌桌上的牌的分布。

商业模型很可能是建立在简单的数学联系上：利润是收入减去支出，收入是平均价格乘以单位销售量，诸如此类。菜谱模型则可能基于反复试验之上——某人走进厨房，尝试不同的原料组合，直到发现自己喜欢的口味。扑克模型基于概率论、扑克规则和某些关于处理牌的随机过程的合理假设。

11.2 什么是机器学习

关于什么是机器学习，每个人都有自己确切的定义。在这里，我们使用的定义是创建并使用那些由学习数据而得出的模型。在其他语境中，这也可以叫作预测建模或者数据挖掘，但是我们选择使用机器学习这个术语。一般来说，我们的目标是用已存在的数据来开发可用来对新数据预测多种可能结果的模型，比如：

- 预测一封邮件是否是垃圾邮件

- 预测一笔信用卡交易是否是欺诈行为
- 预测哪种广告最有可能被购物者点击
- 预测哪支橄榄球队会赢得超级杯大赛

下面我们会看到有监督的模型（其中的数据标注有正确答案，可供学习）和无监督的模型（没有标注）。还有一些其他类型的模型，我们不会在本书中进行讨论，如半监督的模型（其中有一部分数据带有标注）和在线的模型（模型需要根据新加入的数据做持续调整）和强化（经过一系列预测后，模型得到指示效果的信号）。

现在，甚至在最简单的情况下都有一整套模型来描述我们感兴趣的联系。大多数情况下我们会自己选择参数化的模型族，然后使用数据来学习从某种程度上进行优化了的参数。

例如，我们假设一个人的身高（大致上）是他体重的线性函数，然后用数据来学习这个线性函数。或者，我们也可以假设决策树是一种用来诊断患者疾病的好方法，然后使用数据来学习一颗“最优”的树。本书剩余部分会穿插讲述可供我们学习的不同的模型族。

但在此之前，我们需要更好地理解机器学习的基础。本章剩余部分将探讨一些机器学习的基本概念，随后才会讨论到模型本身。

11.3 过拟合和欠拟合

在机器学习中，一种常见的困境是**过拟合**（overfitting）——指一个在训练数据上表现良好，但对任何新数据的泛化能力却很差的模型。这可能牵扯到对数据中噪声的学习，也可能涉及学习识别特别的输入，而不是对可以得到期望的输出进行准确预测的任何因素。

另一种有害的情况是欠拟合（underfitting），它产生的模型甚至在训练数据上都没有好的表现，通常这暗示你模型不够好而要继续寻找改进的模型。

在图 11-1 中，我对一组简单的数据拟合了 3 种多项式（别担心这是怎么做到的，我们会在后面的章节中介绍）。

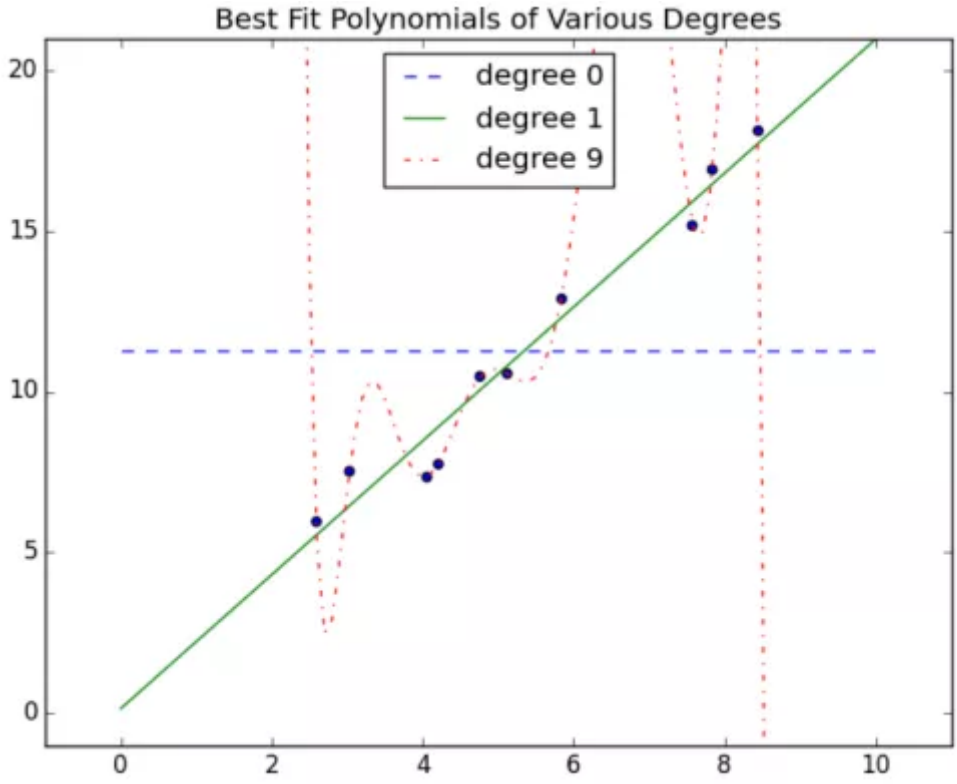

图 11-1：过拟合和欠拟合

水平线显示了最佳拟合阶数为 0（也就是常数）的多项式，它对训练数据来说存在严重的欠拟合。最佳拟合的阶数为 9（也就是有 10 个参数）的多项式精确地穿过训练数据的每个点，但这是严重过拟合的。如果我们能取到更多的一些点，这个多项式很有可能会偏离它们很多。阶数为 1 的线把握了很好的平衡——它和每个点都很接近，并且（如果这些数据是有代表性的）它也会和新的数据点很接近。

很明显，太复杂的模型会导致过拟合，并且在训练数据集之外不能很好地泛化。所以，我们该如何确保我们的模型不会太复杂呢？最基本的方法包括使用不同的数据来训练和测试模型。

最简单的做法是划分数据集，使得（比如说）三分之二的数据用来训练模型，之后用剩余的三分之一来衡量模型的表现：

```
import random
from typing import TypeVar, List, Tuple
X = TypeVar('X')  # generic type to represent a data point

def split_data(data: List[X], prob: float) -> Tuple[List[X], List[X]]:
    """Split data into fractions [prob, 1 - prob]"""
    data = data[:]                      # Make a shallow copy
    random.shuffle(data)                # because shuffle modifies the
list.
    cut = int(len(data) * prob)         # Use prob to find a cutoff
    return data[:cut], data[cut:]       # and split the shuffled list
there.

data = [n for n in range(1000)]
train, test = split_data(data, 0.75)

# The proportions should be correct
assert len(train) == 750
assert len(test) == 250

# And the original data should be preserved (in some order)
assert sorted(train + test) == data
```

通常，我们会有配对的输入变量和输出变量。在这种情况下，我们需要确保在训练数据或测试数据中组合相应的值：

```
Y = TypeVar('Y')  # generic type to represent output variables

def train_test_split(xs: List[X],
                     ys: List[Y],
                     test_pct: float) -> Tuple[List[X], List[X],
List[Y],

List[Y]]:
    # Generate the indices and split them
    idxs = [i for i in range(len(xs))]
    train_idxs, test_idxs = split_data(idxs, 1 - test_pct)

    return ([xs[i] for i in train_idxs],  # x_train
            [xs[i] for i in test_idxs],   # x_test
            [ys[i] for i in train_idxs],  # y_train
            [ys[i] for i in test_idxs])   # y_test
```

和往常一样，我们希望确保我们的代码能正常工作：

```
xs = [x for x in range(1000)]  # xs are 1 ... 1000
ys = [2 * x for x in xs]       # each y_i is twice x_i
x_train, x_test, y_train, y_test = train_test_split(xs, ys, 0.25)

# Check that the proportions are correct
assert len(x_train) == len(y_train) == 750
assert len(x_test) == len(y_test) == 250

# Check that the corresponding data points are paired correctly
assert all(y == 2 * x for x, y in zip(x_train, y_train))
assert all(y == 2 * x for x, y in zip(x_test, y_test))
```

这样你就可以做一些类似下面这样的处理：

```
model = SomeKindOfModel()
x_train, x_test, y_train, y_test = train_test_split(xs, ys, 0.33)
model.train(x_train, y_train)
performance = model.test(x_test, y_test)
```

如果模型对训练数据是过拟合的，那么它在（完全划分开的）测试数据集上会有可能真的表现得很不好，换句话说，如果它在测试数据上表现良好，那么你可以肯定地说它拟合良好而非过拟合。

然而在有些情况下这也可能会出错。

第一种情况是训练和测试数据集中的共有模式不能泛化到大型数据集上。

比如，假设数据集包括用户活跃度，每位用户每周一列。在此情形下，大多数用户会出现在训练数据和测试数据中，而且有些模型可能会学习识别用户而不是去发现涉及属性的联系。这不是个太大的问题，尽管我曾经遇到过一次。

一个更大的问题是，如果你划分训练集和测试集的目的不仅仅是为了判断模型，也是为了在许多模型中进行选择。此时，尽管不是所有的模型都是过拟合的，但“选择在测试集上表现最好的模型”是一种元训练（meta-training），相当于把测试集当作另一个训练集了。（当然，在测试集上有最好表现的模型会在测试集上有持续的好表现。）

在这种情况下，你应该把数据划分为三部分：一个用来建立模型的训练集，一个为在训练好的模型上进行选择的验证集，一个用来判断最终的模型的测试集。

11.4 正确性（correctness）

当我不做数据科学的时候，我会涉猎医疗研究。在业余时间，我做了一个低成本、无害的新生儿测试——准确性高达 98%——测试新生儿是否会得白血病。我的律师确信我是有专利权的。所以我在这里把细节分享一下：仅仅当婴儿起名为 Luke 时预测会得白血病（因为这个名字听起来像白血病的英文 leukemia）。

如我们下面所见，这种测试确实有超过 98% 的准确性。然而，这是一个极为愚蠢的测试，也很好地解释了我们为什么不能仅用“准确率”这个概念来测量一个模型的好坏。

假设建立一个模型来做二元的判断，比如：一封邮件是否是垃圾邮件？我们是否该聘用这位应聘者？这个乘客是潜藏的恐怖分子吗？

给定一个标签数据集和一个预测模型，每个数据集都会落在下面其中一个属性中。

- 真阳性：“这封邮件是垃圾邮件，我们做了正确的预测。”
- 假阳性（又称第 1 类错误）：“这封邮件不是垃圾邮件，但是我们预测它是垃圾邮件。”
- 假阴性（又称第 2 类错误）：“这封邮件是垃圾邮件，但是我们预测它不是垃圾邮件。”
- 真阴性：“这封邮件不是垃圾邮件，而且我们正确地预测了它不是垃圾邮件。”

我们常用**混淆矩阵**（confusion matrix）中的计数来表示上面的四种情况：

	Spam	Not spam
Predict “spam”	True positive	False positive
Predict “not spam”	False negative	True negative

让我们来看看我的白血病测试是如何符合这种框架的。现如今，大约每 1000 名婴儿中有5人会起名叫 Luke 。每人一生中罹患白血病的概率大约是 1.4%，或者说每 1000 人中会有 14 人患病。

如果我们相信这两个因素是独立的，然后对 1 百万人运用我的“Luke 是白血病患者”测试，预计能看到这样的混淆矩阵：

	Leukemia	No leukemia	Total
“Luke”	70	4,930	5,000
Not “Luke”	13,930	981,070	995,000
Total	14,000	986,000	1,000,000

然后我们由此计算关于模型表现的多个统计量。例如，**准确率**（accuracy）定义为正确预测的比例：

```
def accuracy(tp: int, fp: int, fn: int, tn: int) -> float:
    correct = tp + tn
    total = tp + fp + fn + tn
    return correct / total

assert accuracy(70, 4930, 13930, 981070) == 0.98114
```

看起来这个数字令人印象十分深刻，但很明显这并不是一个好的测试，这意味着我们不能对原始的准确率有过多的信心。

更常见的做法是把**查准率**（precision）和**查全率**（recall）结合起来看待。查准率度量我的模型所做的关于“阳性”的预测有多准确：

```
def precision(tp: int, fp: int, fn: int, tn: int) -> float:
    return tp / (tp + fp)


assert precision(70, 4930, 13930, 981070) == 0.014
```

查全率度量我的模型所识别的“阳性”的比例：

```
def recall(tp: int, fp: int, fn: int, tn: int) -> float:
    return tp / (tp + fn)

assert recall(70, 4930, 13930, 981070) == 0.005
```

这两个结果都低得可怕，反映出这是一个很不好的模型。

有时候可以把查准率和查全率组合成**F1得分**（F1 score），它是这样定义的：

```
def f1_score(tp: int, fp: int, fn: int, tn: int) -> float:
    p = precision(tp, fp, fn, tn)
    r = recall(tp, fp, fn, tn)

    return 2 * p * r / (p + r)
```

它是查准率和查全率的调和平均值，因此必然会落在两者之间。

模型的选择通常是查准率和查全率之间的权衡。一个模型如果在信心不足的情况下预测“是”，那么它的查全率可能会较高，但查准率却较低；而如果一个模型在信心十足的情况下预测“是”，那么它的查全率可能会较低，但查准率却较高。

另一方面，也可以把这当作假阳性和假阴性之间的权衡。预测的“是”太多通常会给出很多的假阳性。预测的“否”太多通常会给出很多的假阴性。

假设对白血病来说有10个风险因素，你的身体具备的因素越多，就越容易患上白血病。这种情况下，你可以假设进行一系列连续性的测试：“至少有一个风险因素预测会得白血病”“至少有两个风险因素预测会得白血病”诸如此类。随着临界值的不断提高，测试的查准率也提高了（因为具有更多风险因素的人更容易患上白血病），并且降低了测试的查全率（因为能够达到临界值的最终患病者越来越少）。对于类似这样的情况，选择合适的临界值实际上就是做出正确的权衡。

11.5 偏差-方差（Bias-Variance）权衡

思考过拟合问题的另一种角度是把它作为偏差和方差之间的权衡。

这两种方法都是衡量如果你在不同的训练数据集（来自相同较大的数据集）上多次重新训练模型将会发生什么。

比如，在 11.3 节“过拟合和欠拟合”中提到的 0 阶模型对任何可能的（取自同一总体的）训练集都会造成大量的错误。这表明该模型偏差较高。然而任何两个随机选择的训练集会给出很相似的模型（因为任何两个随机选择的训练集都应该有大致相似的平均值）。所以我们称这个模型有低方差。高偏差和低方差典型地对应着欠拟合。

另一方面，9 阶模型完美地拟合训练集，它具有很低的偏差和很高的方差（因为任何两个训练集都可能给出非常不同的模型形式）。这种情况对应过拟合。

通过这种方式考虑模型问题可以帮助你弄清楚当模型工作得不太好时该如何处理。

如果你的模型有高偏差（这意味着即使在训练数据上也表现不好），可以尝试加入更多的特征。从 0 阶模型到 1 阶模型就是一个很大的改进。

如果你的模型有高方差，那可以类似地移除特征；另一种解决方法是（如果可能的话）获得更多的数据。

在图 11-2 中，我们对不同大小的样本拟合了 9 阶多项式。如我们前面所见，基于10个点拟合的模型是一塌糊涂的。如果在 100 个数据点上训练，就会大大减少过拟合的问题。如果在 1000 个点上训练，看起来就像是 1 阶模型。在模型复杂度不变的前提下，你有越多的数据，就越难过拟合。另一方面，更多的数据对偏差并不会有帮助。如果模型不能使用足够多的特征来捕捉数据中的规律，那么再多的数据也不会有帮助。

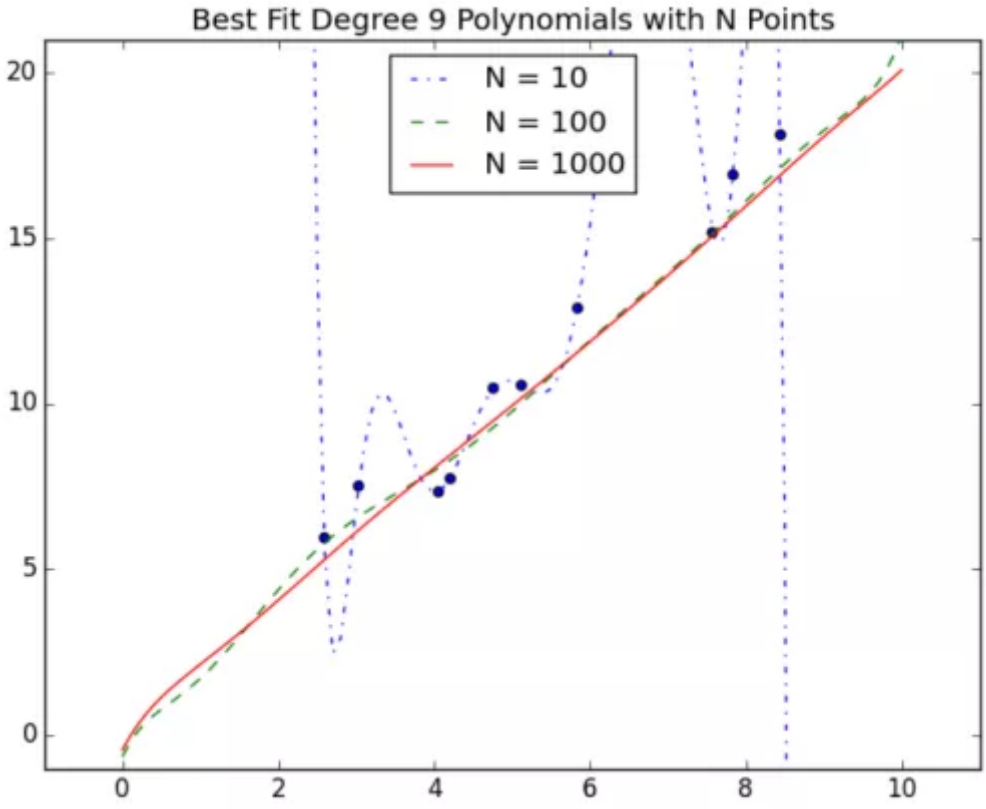

图 11-2：利用更多数据降低方差

11.6 特征提取和选择

我们之前提到，如果数据没有足够的特征，模型很可能就会欠拟合。但如果数据有太多的特征，模型又容易过拟合。那什么是特征呢，它们又从何而来？

特征（feature）是指提供给模型的任何输入。

在最简单的情况下，特征是直接提供给你的。如果你想基于某人的工作年限来预测其薪水，那工作年限就是你所拥有的唯一的特征。（尽管如此，如同我们在 11.3 节"过拟合和欠拟合"中所见的，如果可以帮助你建立更好的模型，可以加入工作年限的平方项和立方项。）

当数据变得更复杂时，事情变得有趣起来。设想我们尝试建立一个垃圾邮件过滤器来预测一封邮件是否是垃圾邮件。大多数模型不知道如何处理原始邮件，邮件就是一组文本。你需要提取特征，比如：

- 邮件中是否包含单词"Viagra"；
- 字母 d 出现了多少次；
- 寄件人的域名是什么。

第一个问题的特征就是简单的是或否，可以被典型地编码为 1 或 0。第二个问题的特征是个数字。第三个问题的特征是从一个离散的选项集中做出的选择。

多数情况下，我们会从符合这三种特征的数据中提取特征。此外，特征的类型限制了我们所用模型的类型。

•第 13 章中使用的朴素贝叶斯分类器适合“是或否”这样的二元特征，就像上面列出的第一种情况一样。
•第 14 章和第 16 章将要提到的回归模型要求有数值型的特征（它可能会包括 0 或 1 这样的哑变量）。
•第 17 章讲到的决策树，会涉及数值或属性数据。

尽管在垃圾邮件过滤器的例子中我们探索了创建特征的方法，但有时我们还需要设法移除特征。

比如，输入可能是包含几百个数的向量。根据具体情况，最好是剔除一些维度，缩减到只剩少量重要的维度（见 10.5 节“降维”），只使用这些少数的特征，或者最好采用一些技术（比如正则化技术，见 15.8 节“正则化”）对应用过多特征的模型进行惩罚。

我们该如何选择特征呢？这需要经验和专业知识的结合。如果你收到了大量的邮件，可能会对某个特定的词比较敏感，这个词会成为垃圾邮件的好指标。同时，你也可能会觉得，字母 d 的个数不像是判断垃圾邮件的好指标。但通常来说，你需要尝试不同的特征，这也不失为一种乐趣。

11.7 延伸学习

• 继续读下去，后面几章讨论了不同种类的机器学习模型。
•Coursera 的机器学习课程（https://www.coursera.org/learn/machine-learning）是原创的 MOOC，是深入理解机器学习基础知识的好途径。
• The Elements of Statistical Learning是由杰罗姆·H·弗里德曼，罗伯特·蒂布希拉尼和特雷弗·哈斯蒂（施普林格）写的一本相当权威的教材，可以从网络上免费下载（https://web.stanford.edu/~hastie/ElemStatLearn//）。但是要注意，它里面有非常多的数学公式。