

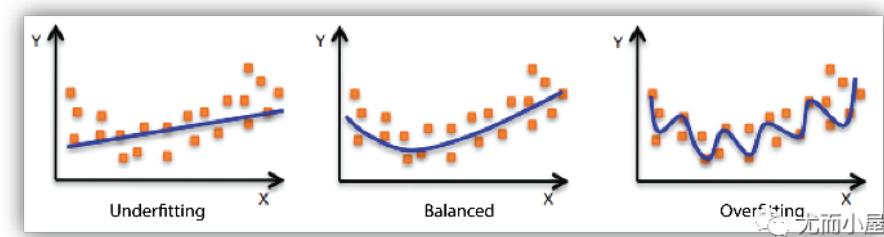
机器学习面试经典题：欠拟合和过拟合

原创 尤而小屋 尤而小屋 2023-04-19 00:47 发表于湖北

收录于合集  
#机器学习 212 #人工智能 84 #数据挖掘 85 #过拟合 1 #正则化 1

公众号：尤而小屋  
作者：Peter  
编辑：Peter

大家好，我是Peter~  
本文给介绍一个机器学习中的经典问题：欠拟合和过拟合



欠拟合和过拟合

在机器学习中，过拟合和欠拟合都是指模型在训练和测试时的表现与期望相差较大的情况。

- 欠拟合：是指模型在训练和测试数据集上表现都不够好，可能是因为模型过于简单，无法拟合数据中的复杂关系，或者是数据量太少，无法充分训练模型。在这种情况下，我们需要增加模型的复杂度，或者增加数据量，以提高模型的表现。
- 过拟合：是指模型在训练数据集上表现很好，但在测试数据集上表现却很差的情况。这可能是因为模型过于复杂，在训练数据集上过度拟合了数据的噪声和细节，导致模型对于新数据的泛化能力较差。在这种情况下，我们需要降低模型的复杂度，或者增加数据量，以减少过拟合的风险。

如何从数学和统计层面理解

从数学和统计学的角度来理解过拟合和欠拟合，我们需要考虑模型的参数数量和拟合数据的程度。

在机器学习中，我们通常使用一个模型来拟合给定的数据集。模型的复杂度可以通过模型中参数的数量来衡量。例如，在线性回归中，模型的参数是斜率和截距，而在多项式回归中，模型的参数是多项式的系数。

欠拟合通常发生在模型过于简单的情况下，即模型的参数数量不足以捕捉数据中的复杂关系。例如，如果我们使用一个线性模型来拟合一个非线性的数据集，那么这个模型就会出现欠拟合的问题。在统计学上，欠拟合通常表现为模型的拟合误差（即模型预测值与真实值之间的差距）较大，这意味着模型不能很好地拟合训练数据集。

过拟合通常发生在模型过于复杂的情况下，即模型的参数数量过多，导致模型过度拟合训练数据集中的噪声和细节。这会导致模型在测试数据集上表现较差，因为模型不能很好地泛化到新的数据。在统计学上，过拟合通常表现为模型的方差较大，即模型在不同数据集上的拟合误差很大。

如何解决欠拟合现象？

1. 增加模型复杂度。欠拟合通常是因为模型过于简单，无法捕捉数据中的复杂关系。因此，我们可以增加模型的复杂度来解决这个问题。例如，在线性回归中，我们可以使用多项式回归来增加模型的复杂度，以便更好地拟合数据。
2. 增加训练数据。欠拟合可能是因为数据量太少，模型无法充分学习数据中的模式。因此，我们可以增加训练数据来解决欠拟合问题。更多的数据可以帮助模型更好地捕捉数据中的模式。
3. 减少正则化参数。如果使用正则化技术来控制模型的复杂度，而导致欠拟合问题，则可以尝试减少正则化参数的值，以允许模型更加灵活。这将使模型能够更好地拟合训练数据，但也会增加过拟合的风险。
4. 使用更复杂的模型。如果以上方法无法解决欠拟合问题，则可以尝试使用更复杂的模型。例如，在神经网络中，我们可以添加更多的隐藏层或更多的神经元来增加模型的复杂度。
5. 特征工程。特征工程是指对原始数据进行预处理或转换，以便更好地反映数据中的模式。例如，在图像分类问题中，我们可以对图像进行预处理，以便更好地提取图像中的特征。

解决欠拟合和过拟合的方法通常是使用正则化技术来平衡模型的复杂度和拟合数据的程度。例如，在线性回归中，我们可以使用L1或L2正则化来惩罚参数的大小，以减少过拟合的风险。

在统计学上，这被称为**偏差-方差权衡**问题，其中偏差是由于模型过于简单而导致的误差，而方差是由于模型过于复杂而导致的误差。通过调整模型的复杂度和正则化参数，我们可以平衡偏差和方差，以获得更好的泛化性能

如何理解偏差-方差权衡问题？

---

偏差-方差权衡（**bias-variance tradeoff**）是机器学习中一个经典的问题。在训练一个机器学习模型时，我们通常会希望模型能够在训练集和测试集上都有好的表现，但是在实践中往往会出现以下两种情况：

1. 模型在训练集上表现很好，但是在测试集上表现很差。这种情况通常被称为高偏差（**high bias**）。
2. 模型在训练集上表现很好，但是在测试集上表现也很差。这种情况通常被称为高方差（**high variance**）。

这两种情况的产生原因是不同的。高偏差通常是由于模型太简单，无法很好地拟合数据，导致在训练集和测试集上都表现很差；高方差通常是由于模型太复杂，过度拟合了训练数据，导致在测试集上表现很差。

解决偏差-方差权衡问题的关键在于找到合适的模型复杂度。如果模型太简单，会导致高偏差；如果模型太复杂，会导致高方差。因此，需要根据具体问题选择合适的模型，并进行模型调参来寻找最佳的模型复杂度。

如何解决过拟合现象？

---

1. 正则化技术。正则化技术可以通过对模型的参数进行惩罚来控制模型的复杂度，从而减少过拟合的风险。常用的正则化技术包括L1正则化和L2正则化。
2. 暂退法Dropout。在神经网络中，还有一种称为Dropout的正则化技术，它随机地丢弃一部分神经元，以减少神经网络的复杂度。
3. 提前停止训练。提前停止训练是一种有效的防止过拟合的方法。在训练神经网络时，我们可以在验证集上监测模型的性能，并在模型性能达到最佳时停止训练，以避免模型过拟合训练数据。
4. 数据增强。数据增强是一种通过对原始数据进行一系列随机变换来生成新的训练数据的方法。这可以增加训练数据的多样性，从而减少过拟合的风险。
5. 集成学习。集成学习是一种通过组合多个模型来提高模型性能的方法。其中，Bagging和Boosting是两种常用的集成学习方法。这些方法可以减少模型的方差，从而减少过拟合的风险。
6. 减少特征数量。过多的特征可能会导致过拟合的问题。因此，我们可以尝试减少特征数量，以减少模型的复杂度，从而避免过拟合。

常用的正则化技术有哪些？

---

- 1. L1正则化：也称为Lasso正则化，它通过对模型参数的绝对值进行惩罚来控制模型的复杂度。L1正则化可以将某些参数设置为0，从而实现特征选择的功能。
- 2. L2正则化：也称为Ridge正则化，它通过对模型参数的平方进行惩罚来控制模型的复杂度。L2正则化可以防止模型过拟合，因为它强制模型参数变得平滑。
- 3. Dropout：Dropout是一种神经网络正则化技术，它随机地丢弃一部分神经元，以减少神经网络的复杂度。Dropout可以防止神经网络过拟合训练数据。

深入理解L1正则化技术

L1正则化也被称为Lasso正则化，是一种常用的正则化技术，它可以通过对模型参数的绝对值进行惩罚来控制模型的复杂度。L1正则化可以将某些参数设置为0，从而实现特征选择的功能。下面详细介绍一下L1正则化的原理和实现方式。

L1正则化的原理

在机器学习中，通常使用最小化损失函数来训练模型，而L1正则化可以通过在损失函数中添加正则项来控制模型的复杂度，其目标是 minimized 损失函数和正则项之和：

$$\operatorname{argmin}_w \left\{ \frac{1}{N} \sum_{i=1}^N \mathcal{L}(y_i, f(x_i, w)) + \lambda \|w\|_1 \right\}$$

其中， $w$ 是模型参数， $N$ 是样本数量， $\mathcal{L}$ 是损失函数， $y_i$ 和 $x_i$ 分别是第*i*个样本的标签和特征， $f(x_i, w)$ 是模型预测值， $\|w\|_1$ 是模型参数的L1范数， $\lambda$ 是正则化强度的超参数。

L1正则化的目标是 minimized 损失函数和L1范数之和，因为L1范数是模型参数的绝对值之和，所以它会惩罚模型参数中的非零值，从而促使一些参数变为0。这种特性使得L1正则化可以实现特征选择的功能，即通过使一些特征的权重为0来选择重要的特征。

L1正则化实现方式

L1正则化的实现方式比较简单，可以通过在优化算法中添加一个L1范数惩罚项来实现。在梯度下降法中，参数的更新规则可以表示为：

$$w_{t+1} = w_t - \eta \frac{\partial \mathcal{L}(y_t, f(x_t, w_t))}{\partial w_t} - \lambda \operatorname{sgn}(w_t)$$

其中， $\eta$ 是学习率， $\mathcal{L}$ 是损失函数， $y_t$ 和 $x_t$ 分别是第*t*个样本的标签和特征， $f(x_t, w_t)$ 是模型预测值， $\lambda$ 是正则化强度的超参数， $\operatorname{sgn}(w_t)$ 是 $w_t$ 的符号函数，即：

$$\operatorname{sgn}(w_t) = \begin{cases} -1, & \text{if } w_t < 0 \\ 0, & \text{if } w_t = 0 \\ 1, & \text{if } w_t > 0 \end{cases}$$

这个更新规则的第三项即为L1正则化的惩罚项，它使得模型参数 $w_t$ 中非零值受到惩罚，从而促使一些参数变为0。通过这种方式，L1正则化可以实现特征选择的功能，即通过使一些特征的权重为0来选择重要的特征。

除了梯度下降法外，L1正则化还可以在其他优化算法中使用，例如坐标下降法、拟牛顿法等。不同的优化算法在实现L1正则化时可能有所不同，但核心思想都是在损失函数中添加L1范数惩罚项。

总结

L1正则化是一种常用的正则化技术，可以通过对模型参数的绝对值进行惩罚来控制模型的复杂度。L1正则化可以将某些参数设置为0，从而实现特征选择的功能。在实现L1正则化时，可以在损失函数中添加L1范数惩罚项，或者在优化算法中对参数进行更新时添加L1范数惩罚项。

python实现L1正则化

在Python中，可以使用Scikit-learn库来实现L1正则化。Scikit-learn中提供了多种回归模型，包括线性回归、岭回归、Lasso回归等。其中，Lasso回归就是使用L1正则化的线性回归模型。

以下是使用Scikit-learn库实现Lasso回归的示例代码：

```
from sklearn.linear_model import Lasso

# 构建模型对象
lasso = Lasso(alpha=0.1)

# 训练模型
lasso.fit(X_train, y_train)

# 预测
y_pred = lasso.predict(X_test)

# 计算模型得分
score = lasso.score(X_test, y_test)
```

在这个例子中，我们首先导入Lasso类，然后创建一个模型对象lasso。通过指定alpha参数的值，可以控制L1正则化的强度。当alpha越大，惩罚项的作用就越强，模型参数越趋向于稀疏。在训练模型之后，我们可以使用predict()函数对测试数据进行预测，使用score()函数计算模型的得分。

需要注意的是，在使用L1正则化时，通常需要对数据进行标准化处理，以避免不同特征的权重差异过大导致模型参数难以优化的问题。可以使用Scikit-learn中的StandardScaler类来进行标准化处理。

深入理解L2正则化技术

L2正则化基本原理

L2正则化是一种常用的正则化技术，也被称为权重衰减（weight decay）。与L1正则化类似，L2正则化也是通过对模型参数进行惩罚来控制模型的复杂度。不同之处在于，L2正则化使用的是L2范数作为惩罚项。

L2正则化的惩罚项为：

$$\frac{\lambda}{2} \sum_{i=1}^n w_i^2$$

其中，λ为正则化参数，n为模型参数的数量，wi为第i个模型参数。

在训练模型时，我们的目标是最小化损失函数，同时还要使模型参数尽量小。为了实现这个目标，我们可以在损失函数中添加L2范数惩罚项。这样，当模型参数过大时，惩罚项就会增大，从而使损失函数增大。反之，当模型参数较小时，惩罚项就会减小，从而使损失函数减小。通过这种方式，L2正则化可以帮助我们控制模型的复杂度，并降低过拟合的风险。

L2正则化的导数为：

$$\frac{\partial}{\partial w_t} \left( J + \frac{\lambda}{2} \sum_{i=1}^n w_i^2 \right) = \frac{\partial J}{\partial w_t} + \lambda w_t$$

其中，J为损失函数。

在使用L2正则化时，我们通常将惩罚项的系数λ设置为一个较小的正值，通常在0.01到0.1之间。较小的惩罚系数可以使模型更加灵活，但过大的惩罚系数可能会导致模型欠拟合。

在实际应用中，L2正则化通常与其他正则化技术一起使用，例如Dropout和数据增强等。

- L2正则化的优点是可以减少过拟合的风险，并且在某些数据集上比L1正则化更有效
- 缺点是无法将一些参数设置为0，不能实现特征选择的功能。

Python实现L2正则化

Python实现L2正则化也非常简单。以PyTorch为例，可以使用torch.optim模块中的weight\_decay参数来实现L2正则化。例如：

```
import torch
import torch.nn as nn
import torch.optim as optim

# 定义模型
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.fc1 = nn.Linear(100, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = self.fc1(x)
        x = nn.ReLU()(x)
        x = self.fc2(x)
        return x

# 定义损失函数和优化器
model = MyModel()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01, weight_decay=0.01)

# 训练模型
for epoch in range(num_epochs):
    for i, (inputs, labels) in enumerate(train_loader):
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

优化器的weight\_decay参数设置为0.01，这就是L2正则化的惩罚系数。在训练过程中，模型的参数会根据损失函数的梯度进行更新，同时也会受到L2正则化的惩罚，从而控制模型的复杂度。

#### L1和L2正则化使用场景

---

L1正则化和L2正则化在应用中有着不同的使用场景。

L1正则化适用于特征选择或者稀疏性问题。由于L1正则化会将一部分参数压缩为0，因此可以通过L1正则化来选出最重要的特征，达到特征选择的目的。此外，L1正则化也可以应用于稀疏性问题，例如图像处理中的压缩感知等。

L2正则化适用于防止过拟合问题。由于L2正则化会对参数进行平滑处理，使得模型更加平稳，因此可以通过L2正则化来防止模型过拟合训练数据，提高模型的泛化能力。

一般来说，如果模型的目标是特征选择或者需要达到稀疏性的目的，可以选择L1正则化；如果模型存在过拟合的问题，需要提高模型的泛化能力，可以选择L2正则化。当然，也可以同时使用L1和L2正则化，这样既可以达到特征选择的目的，又可以防止模型过拟合。

#### 什么是Dropout正则化

---

##### 基本原理

Dropout正则化是一种常用的神经网络正则化方法，可以有效地防止神经网络的过拟合问题。

Dropout正则化的基本思想是在神经网络的训练过程中，以一定的概率将一些神经元随机地舍弃掉，从而减少神经网络中神经元的相互依赖性，增强模型的泛化能力。具体来说，假设神经网络中某一层有 $n$ 个神经元，那么在Dropout正则化中，每个神经元都有一定的概率 $p$ 被保留下来，有 $1-p$ 的概率被丢弃。这样可以让模型在训练过程中不依赖于任何一个神经元，从而避免神经元之间的过度适应训练数据的情况，提高模型的泛化能力。

在预测时，所有的神经元都被保留下来，但是每个神经元的权重需要乘以 $p$ ，以保持训练时的期望输出。这样可以使得Dropout正则化不影响模型在预测时的表现。

Dropout正则化可以通过在神经网络中添加Dropout层来实现，这个层会以一定的概率随机地舍弃一些神经元。另外，Dropout正则化还可以与其他正则化方法如L1和L2正则化一起使用，从而进一步提高模型的泛化能力。

在实际应用中，Dropout正则化已经被证明是一种有效的正则化方法，可以有效地防止过拟合问题，提高模型的泛化能力。

### 从数学层面理解Dropout正则化

从数学公式层面理解Dropout正则化可以更加深入地理解其原理。在神经网络中使用Dropout正则化时，其数学公式可以表示为：

$$\mathbf{y} = f(\mathbf{x} \cdot \mathbf{W} + \mathbf{b}) \cdot \mathbf{d}$$

其中， $\mathbf{x}$  表示输入向量， $\mathbf{y}$  表示输出向量， $f(\cdot)$  表示激活函数， $\mathbf{W}$  表示权重矩阵， $\mathbf{b}$  表示偏置向量， $\mathbf{d}$  表示dropout向量。

其中， $\mathbf{d}$ 表示一个由0和1构成的向量，0表示相应的神经元被丢弃，1表示相应的神经元被保留。在训练过程中， $\mathbf{d}$ 是一个随机向量，由Bernoulli分布生成，每个元素都有一定的概率 $p$ 被保留下来，有 $1-p$ 的概率被丢弃。在预测时，所有的神经元都被保留下来，但是每个神经元的权重需要乘以 $p$ ，以保持训练时的期望输出。

在使用Dropout正则化时，由于一些神经元被随机丢弃掉，因此在每次训练时都相当于使用不同的神经网络子集来训练模型，这些子集之间是互相独立的。因此，Dropout正则化可以看作是对所有可能的神经网络的平均，相当于集成了多个模型的预测结果，从而减少了模型的方差，提高了模型的泛化能力。

从数学公式层面来看，Dropout正则化可以通过在神经网络中添加Dropout层来实现，这个层会以一定的概率随机地舍弃一些神经元。在实现时，通常可以通过设置Dropout层的参数 $p$ 来控制神经元的保留率，从而控制Dropout正则化的效果。

### python实现Dropout正则化技术

在Python中，可以使用TensorFlow或PyTorch等深度学习框架来实现Dropout正则化技术。下面分别给出TensorFlow和PyTorch的实现示例。

使用TensorFlow实现Dropout正则化：

```
import tensorflow as tf

# 定义神经网络模型
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5), # 添加Dropout层，保留率为0.5
    tf.keras.layers.Dense(10)
])

# 编译模型
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
# 训练模型
model.fit(train_dataset, epochs=10)
```

使用PyTorch实现Dropout正则化:

```
import torch
import torch.nn as nn

# 定义神经网络模型
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 64)
        self.dropout = nn.Dropout(0.5) # 添加Dropout层, 保留率为0.5
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = nn.functional.relu(x)
        x = self.dropout(x)
        x = self.fc2(x)
        return x

model = Net()

# 定义损失函数和优化器
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters())

# 训练模型
for epoch in range(10):
    for data, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(data)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

除了正则化，还有哪些方法解决过拟合？

除了正则化技术，还有其他方法可以用来解决过拟合问题，下面列举几个常用的方法：

1. 增加数据量：增加更多的数据可以让模型更好地学习到数据的分布情况，从而减少过拟合的风险。
2. 数据增强：通过对数据进行旋转、平移、缩放等变换，可以生成更多的数据，从而提高模型的泛化能力。
3. 早停法（Early stopping）：在训练模型时，可以监控模型在验证集上的性能，并在性能不再提高时停止训练，以避免过拟合。
4. 增加噪声：在输入数据中加入噪声可以使得模型对输入数据中的随机波动更加鲁棒，从而提高模型的泛化能力。
5. 模型集成（Model ensemble）：将多个不同的模型进行集成，可以减少过拟合的风险，提高模型的泛化能力等。



尤而小屋

4



收录于合集 #机器学习 212

上一篇 · 推特算法开源，Star秒破万！

喜欢此内容的人还喜欢

12种机器学习论文  
泛函的范



机器学习系列|时间序列相似度的度量方法  
科研生信充电宝



机器学习：XGBoost算法介绍  
冷冻工厂





