计算机科学188：人工智能导论
2024年春季 笔记21

作者（其他所有笔记）：尼基尔·夏尔马
作者（贝叶斯网络笔记）：乔希·胡格和杰基·梁，由王蕾吉娜编辑
作者（逻辑笔记）：亨利·朱，由考佩林编辑
致谢（机器学习与逻辑笔记）：部分章节改编自教材《人工智能：一种现代方法》。

最后更新时间：2024年4月2日

线性回归

现在我们将从之前对朴素贝叶斯的讨论转向线性回归。这种方法，也称为最小二乘法，可追溯到卡尔·弗里德里希·高斯，是机器学习和计量经济学中研究最多的工具之一。

回归问题是机器学习问题的一种形式，其中输出是一个连续变量（用 y 表示）。特征可以是连续的或分类的。我们将用 $\mathbf{x} \in \mathbb{R}^n$ 表示一组特征，其中有 n 个特征，即 $\mathbf{x} = \left(x^1, \ldots, x^n\right)$ 。

我们使用以下线性模型来预测输出：

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x^1 + \cdots w_n x^n$$

线性模型的权重 w_i 是我们想要估计的量。权重 w_0 对应于模型的截距。有时在文献中，我们在特征向量 $\mathbf{x}$ 上添加一个1，这样我们可以将线性模型写成 $\mathbf{w}^T\mathbf{x}$ ，此时 $\mathbf{x} \in \mathbb{R}^{n+1}$ 。为了训练模型，我们需要一个衡量模型预测输出效果的指标。为此，我们将使用 $L2$ 损失函数，该函数使用 $L2$ 范数来惩罚预测值与实际输出之间的差异。如果我们的训练数据集有 N 个数据点，那么损失函数定义如下：

$$Loss(h_{\mathbf{w}}) = \frac{1}{2}\sum_{j=1}^{N} L2(y^j, h_{\mathbf{w}}(\mathbf{x}^j)) = \frac{1}{2}\sum_{j=1}^{N} (y^j - h_{\mathbf{w}}(\mathbf{x}^j))^2 = \frac{1}{2}\left\|\mathbf{y} - \mathbf{X}\mathbf{w}\right\|_2^2$$

请注意， $\mathbf{x}^j$ 对应于第 j 个数据点 $\mathbf{x}^j \in \mathbb{R}^n$ 。术语 $\frac{1}{2}$ 只是为了简化闭式解的表达式而添加的。最后一个表达式是损失函数的等效形式，这使得最小二乘推导更容易。量 $\mathbf{y}, \mathbf{X}$ 和 $\mathbf{w}$ 定义如下：

$$\mathbf{y} = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_1^1 & \cdots & x_1^n \\ 1 & x_2^1 & \cdots & x_2^n \\ \vdots & \vdots & \cdots & \vdots \\ 1 & x_N^1 & \cdots & x_N^n \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix},$$

其中 $\mathbf{y}$ 是堆叠输出的向量， $\mathbf{X}$ 是特征向量的矩阵，其中 x_j^i 表示第 j 个数据点的第 i 个分量。用 $\widehat{\mathbf{w}}$ 表示的最小二乘解现在可以使用基本线性代数规则 [1] 推导出来。更具体地说，我们将通过对损失函数求导并将导数设为零来找到使损失函数最小化的 $\widehat{\mathbf{w}}$ 。

$$\begin{aligned}\nabla_{\mathbf{w}} \frac{1}{2}\|\mathbf{y}-\mathbf{X}\mathbf{w}\|_2^2 &= \nabla_{\mathbf{w}} \frac{1}{2}(\mathbf{y}-\mathbf{X}\mathbf{w})^T(\mathbf{y}-\mathbf{X}\mathbf{w}) = \nabla_{\mathbf{w}} \frac{1}{2}\left(\mathbf{y}^T\mathbf{y}-\mathbf{y}^T\mathbf{X}\mathbf{w}-\mathbf{w}^T\mathbf{X}^T\mathbf{y}+\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}\right) \\ &= \nabla_{\mathbf{w}} \frac{1}{2}\left(\mathbf{y}^T\mathbf{y}-2\mathbf{w}^T\mathbf{X}^T\mathbf{y}+\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}\right) = -\mathbf{X}^T\mathbf{y}+\mathbf{X}^T\mathbf{X}\mathbf{w}.\end{aligned}$$

将梯度设为零，我们得到：

$$-\mathbf{X}^T\mathbf{y}+\mathbf{X}^T\mathbf{X}\mathbf{w}=0 \Rightarrow \hat{\mathbf{w}}=(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}.$$

在获得权重的估计向量后，我们现在可以对新的未见过的测试数据点进行如下预测：

$$h_{\hat{\mathbf{w}}}(\mathbf{x})=\hat{\mathbf{w}}^T\mathbf{x}.$$

在本笔记中，我们将介绍如何使用梯度下降算法优化函数。我们还将了解一种称为逻辑回归的分类方法，以及它如何扩展到多类分类。这将激发我们对神经网络和反向传播的进一步讨论。

优化

我们在上一篇关于线性回归方法的笔记中看到，通过对损失函数求导并将梯度设为零，我们可以推导出最优权重的闭式解。然而，一般来说，对于给定的目标函数可能不存在闭式解。在这种情况下，我们必须使用基于梯度的方法来找到最优权重。其背后的思想是，梯度指向目标函数增长最快的方向。我们通过朝着最陡上升方向移动来最大化一个函数，通过朝着最陡下降方向移动来最小化一个函数。
如果目标是一个我们试图最大化的函数，则使用梯度上升。

随机初始化 $\mathbf{w}$
当 w 未收敛时执行
    $\mathbf{w} \leftarrow \mathbf{w}+\alpha\nabla_{\mathbf{w}} f(\mathbf{w})$
结束

算法 1：梯度上升

如果目标是一个我们试图最小化的损失函数，就使用梯度下降法。请注意，这与梯度上升法的唯一区别在于我们沿着梯度的相反方向进行。

随机初始化 $\mathbf{w}$
当w未收敛时执行
    $\mathbf{w} \leftarrow \mathbf{w}-\alpha\nabla_{\mathbf{w}} f(\mathbf{w})$
结束

以及算法2：梯度下降法

[1] 对于矩阵代数规则，你可以参考《矩阵手册》。

一开始，我们随机初始化权重。我们用 α 表示学习率，它决定了我们朝着梯度方向所走步长的大小。对于机器学习领域的大多数函数而言，很难找到学习率的最优值。实际上，我们希望学习率足够大，以便能快速朝着正确方向前进，但同时又足够小，以免方法发散。机器学习文献中的一种典型方法是，以相对较大的学习率开始梯度下降，并随着迭代次数的增加而降低学习率（学习率衰减）。

如果我们的数据集有大量的 n 数据点，那么在梯度下降算法的每次迭代中按上述方法计算梯度可能计算量过大。因此，人们提出了诸如随机梯度下降和批量梯度下降等方法。在随机梯度下降中，在算法的每次迭代中，我们仅使用一个数据点来计算梯度。该数据点每次都是从数据集中随机采样得到的。鉴于我们仅使用一个数据点来估计梯度，随机梯度下降可能会导致梯度有噪声，从而使收敛变得有点困难。小批量梯度下降是随机梯度下降和普通梯度下降算法之间的一种折衷，因为它每次使用一批大小为 m 的数据点来计算梯度。批量大小 m 是一个用户指定的参数。
让我们来看一个在我们之前见过的模型——线性回归上进行梯度下降的例子。回想一下，在 线性回归中，我们将损失函数定义为

$$Loss(h_{\mathbf{w}}) = \frac{1}{2}\|\mathbf{y} - \mathrm{X}\mathbf{w}\|_2^2$$

线性回归有一个著名的闭式解 $\hat{\mathbf{w}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$ ，我们在上一篇笔记中已经看到。然而，我们也可以选择通过运行梯度下降来求解最优权重。我们将损失函数的梯度计算为

$$\nabla_{\mathbf{w}} Loss(h_{\mathbf{w}}) = -\mathrm{X}^T\mathbf{y} + \mathrm{X}^T\mathrm{X}\mathbf{w}$$

然后，我们使用这个梯度来写出线性回归的梯度下降算法：

随机初始化 $\mathbf{w}$
当w未收敛时执行
  $\mathbf{w} \leftarrow \mathbf{w} - \alpha(-\mathrm{X}^T\mathbf{y} + \mathrm{X}^T\mathrm{X}\mathbf{w})$
结束

算法3：最小二乘梯度下降

创建一个线性回归问题并确认闭式解与通过梯度下降获得的收敛解相同，这是一个很好的练习。

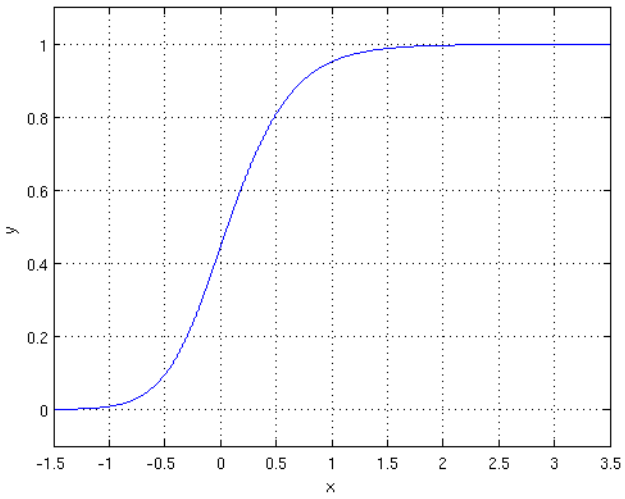
逻辑回归

让我们再次思考我们之前的线性回归方法。在该方法中，我们假设输出是一个数值实值数。
如果我们想要预测一个分类变量会怎样呢？逻辑回归允许我们使用逻辑函数将输入特征的线性组合转换为概率：

$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}}.$$

需要注意的是，尽管逻辑回归被称为回归，但这是一个误称。逻辑回归用于解决分类问题，而非回归问题。

逻辑函数 $g(z) = \frac{1}{1+e^{-z}}$ 经常用于对二元输出进行建模。请注意，该函数的输出始终介于0和1之间，如下图所示：



直观地说，逻辑函数对数据点属于标签为1的类的概率进行建模。原因在于逻辑函数的输出被限制在0和1之间，并且我们希望我们的模型能够捕捉特征具有特定标签的概率。例如，在训练逻辑回归之后，我们得到新数据点的逻辑函数输出。如果输出值大于0.5，我们将其分类为标签1，否则分类为标签0。更具体地说，我们按如下方式对概率进行建模：

$$P(y=+1|\mathbf{f}(\mathbf{x});\mathbf{w}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{f}(\mathbf{x})}}$$

$$P(y=-1|\mathbf{f}(\mathbf{x});\mathbf{w}) = 1 - \frac{1}{1+e^{-\mathbf{w}^T\mathbf{f}(\mathbf{x})}},$$

其中我们使用 $\mathbf{f}(\mathbf{x})$ 表示特征向量 $\mathbf{x}$ 的函数（通常是恒等函数），分号 ; 表示概率是参数权重 $\mathbf{w}$ 的函数。

需要注意的一个有用性质是，逻辑函数的导数是 $g'(z) = g(z)(1-g(z))$ 。

我们如何训练逻辑回归模型？逻辑回归的损失函数是 $L2$ 损失 $\text{Loss}(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - h_{\mathbf{w}}(\mathbf{x}))^2$ 。由于逻辑回归没有闭式解，我们通过梯度下降来估计未知权重 $\mathbf{w}$ 。为此，我们需要使用微分的链式法则来计算损失函数的梯度。损失函数关于坐标 i 权重的梯度由下式给出：

$$\frac{\partial}{\partial w_i}\frac{1}{2}(y-h_{\mathbf{w}}(\mathbf{x}))^2 = (y-h_{\mathbf{w}}(\mathbf{x}))\frac{\partial}{\partial w_i}(y-h_{\mathbf{w}}(\mathbf{x})) = -(y-h_{\mathbf{w}}(\mathbf{x}))h_{\mathbf{w}}(\mathbf{x})(1-h_{\mathbf{w}}(\mathbf{x}))x_i,$$

我们利用了逻辑函数 $g(z) = \frac{1}{1+e^{-z}}$ 的梯度满足 $g'(z) = g(z)(1-g(z))$ 这一事实。然后我们可以使用梯度下降来估计权重，进而如上文详述的那样进行预测。

多分类逻辑回归

在多分类逻辑回归中，我们希望将数据点分类到 K 个不同的类别中，而不仅仅是两个。因此，我们想要构建一个模型，该模型输出新数据点属于 K 个可能类别中每一个类别的概率估计值。出于这个原因，我们使用softmax函数代替逻辑函数，softmax函数对具有特征 $\mathbf{x}$ 的新数据点具有标签 i 的概率进行建模，如下所示：

$$P(y=i|\mathbf{f}(\mathbf{x});\mathbf{w})=\frac{e^{\mathbf{w}_i^T\mathbf{f}(\mathbf{x})}}{\sum_{k=1}^K e^{\mathbf{w}_k^T\mathbf{f}(\mathbf{x})}}.$$

请注意，这些概率估计值的总和为1，因此它们构成一个有效的概率分布。我们估计参数 $\mathbf{w}$ 以最大化我们观察到数据的可能性。假设我们已经观察到 n 个标记数据点 $(\mathbf{x}_i, y_i)$ 。似然性定义为我们样本的联合概率分布，用 $\ell\left(\mathbf{w}_1,\ldots,\mathbf{w}_K\right)$ 表示，由下式给出：

$$\ell(\mathbf{w}_1,\ldots,\mathbf{w}_K)=\prod_{i=1}^n P(y_i|\mathbf{f}(\mathbf{x}_i);\mathbf{w}).$$

为了计算使似然性最大化的参数 $\mathbf{w}_i$ 的值，我们计算似然函数关于每个参数的梯度，将其设为零，然后求解未知参数。如果无法得到闭式解，我们计算似然函数的梯度，并使用梯度上升法来获得最优值。

我们用来简化这些计算的一个常见技巧是先对似然函数取对数，这会将乘积转换为求和，并简化梯度计算。我们可以这样做是因为对数是一个严格递增函数，并且这种变换不会影响函数的最大化点。

对于似然函数，我们需要一种方法来表示概率 $P\left(y_i \mid \mathbf{f}\left(\mathbf{x}_i\right); \mathbf{w}\right)$ ，其中 $y \in \{1,\ldots,K\}$ 。所以对于每个数据点 i ，我们定义 K 个参数 $t_{i,k}, k=1,\ldots,K$ ，使得当 $y_i=k$ 时 $t_{i,k}=1$ ，否则为 0。因此，我们现在可以将似然表示如下：

$$\ell(\mathbf{w}_1,\ldots,\mathbf{w}_K)=\prod_{i=1}^n\prod_{k=1}^K\left(\frac{e^{\mathbf{w}_k^T\mathbf{f}(\mathbf{x}_i)}}{\sum_{\ell=1}^K e^{\mathbf{w}_\ell^T\mathbf{f}(\mathbf{x}_i)}}\right)^{t_{i,k}}$$

并且我们还得到对数似然为：

$$\log\ell(\mathbf{w}_1,\ldots,\mathbf{w}_K)=\sum_{i=1}^n\sum_{k=1}^K t_{i,k}\log\left(\frac{e^{\mathbf{w}_k^T\mathbf{f}(\mathbf{x}_i)}}{\sum_{\ell=1}^K e^{\mathbf{w}_\ell^T\mathbf{f}(\mathbf{x}_i)}}\right)$$

既然我们有了目标函数的表达式，我们就必须估计 $\mathbf{w}_i$ ，使得它们最大化该目标函数。

在多类逻辑回归的例子中，关于 $\mathbf{w}_j$ 的梯度由下式给出：

$$\nabla_{\mathbf{w}_j}\log\ell(\mathbf{w})=\sum_{i=1}^n\nabla_{\mathbf{w}_j}\sum_{k=1}^K t_{i,k}\log\left(\frac{e^{\mathbf{w}_k^T\mathbf{f}(\mathbf{x}_i)}}{\sum_{\ell=1}^K e^{\mathbf{w}_\ell^T\mathbf{f}(\mathbf{x}_i)}}\right)=\sum_{i=1}^n\left(t_{i,j}-\frac{e^{\mathbf{w}_j^T\mathbf{f}(\mathbf{x}_i)}}{\sum_{\ell=1}^K e^{\mathbf{w}_\ell^T\mathbf{f}(\mathbf{x}_i)}}\right)\mathbf{f}(\mathbf{x}_i),$$

这里我们利用了 $\sum_k t_{i,k}=1$ 这一事实。