计算机科学188：人工智能导论
2024年春季 笔记20

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：2023年8月26日

感知机

线性分类器

朴素贝叶斯背后的核心思想是提取训练数据的某些属性，即特征，然后估计给定特征下标签的概率：$P\left(y \mid f_1, f_2, \ldots f_n\right)$。因此，给定一个新的数据点，我们可以提取相应的特征，并根据给定特征下概率最高的标签对新数据点进行分类。然而，这一切都需要我们估计分布，我们使用最大似然估计法进行了估计。如果我们决定不估计概率分布会怎样呢？让我们从一个简单的线性分类器开始，它可用于二分类，即标签有两种可能性，正或负。

线性分类器的基本思想是使用特征的线性组合进行分类——我们将这个值称为激活值。具体来说，激活函数接收一个数据点，将我们数据点的每个特征（$f_i\left(\mathbf{x}\right)$）与相应的权重（$w_i$）相乘，并输出所有结果值的总和。以向量形式表示，我们也可以将其写为权重向量（$\mathbf{w}$）与特征化数据点向量（$\mathbf{f}\left(\mathbf{x}\right)$）的点积：

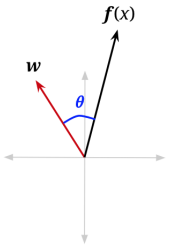
$$\text{activation}_w(\mathbf{x}) = h_{\mathbf{w}}(\mathbf{x}) = \sum_i w_i f_i(\mathbf{x}) = \mathbf{w}^T \mathbf{f}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{f}(\mathbf{x})$$

如何使用激活值进行分类？对于二分类，当一个数据点的激活值为正时，我们将该数据点分类为正标签，如果为负，则将其分类为

$$\text{classify}(\mathbf{x}) = \begin{cases} + & \text{if } h_{\mathbf{w}}(\mathbf{x}) > 0 \\ - & \text{if } h_{\mathbf{w}}(\mathbf{x}) < 0 \end{cases}$$

为了从几何角度理解这一点，让我们重新审视矢量化激活函数。我们可以将点积改写如下，其中 $\|\cdot\|$ 是模运算符，θ 是 $\mathbf{w}$ 与 $\mathbf{f}\left(\mathbf{x}\right)$ 之间的夹角：

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{f}(\mathbf{x}) = \|\mathbf{w}\|\|\mathbf{f}(\mathbf{x})\| \cos(\theta)$$



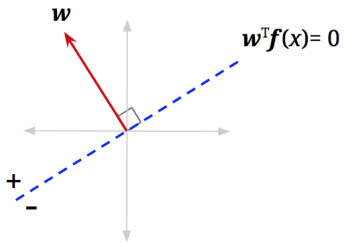
由于幅值始终是非负的，并且我们的分类规则关注激活的符号，所以对于确定类别而言唯一重要的项是 $\cos(\theta)$ 。

$$\text{classify}(\mathbf{x}) = \begin{cases} + & \text{if } cos(\theta) > 0 \\ - & \text{if } cos(\theta) < 0 \end{cases}$$

因此，我们关注 $\cos(\theta)$ 何时为负或为正。很容易看出，对于 $\theta < \frac{\pi}{2}, \cos(\theta)$ 将位于区间 $(0, 1]$ 中的某个位置，该区间是正的。对于 $\theta > \frac{\pi}{2}, \cos(\theta)$ 将位于区间 $[-1, 0)$ 中的某个位置，该区间是负的。你可以通过查看单位圆来确认这一点。本质上，我们的简单线性分类器正在检查新数据点的特征向量是否大致与预定义的权重向量指向相同方向，如果是则应用正标签。

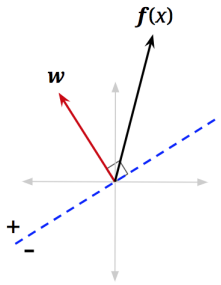
$$\text{classify}(\mathbf{x}) = \begin{cases} + & \text{if } \theta < \frac{\pi}{2} \quad \text{(i.e. when } \theta \text{ is less than } 90°\text{, or acute)} \\ - & \text{if } \theta > \frac{\pi}{2} \quad \text{(i.e. when } \theta \text{ is greater than } 90°\text{, or obtuse)} \end{cases}$$

到目前为止，我们还没有考虑激活 $_w(\mathbf{x}) = \mathbf{w}^T\mathbf{f}(\mathbf{x}) = 0$ 的点。按照相同的逻辑，我们会发现这些点的 $\cos(\theta) = 0$ 。此外，这些点的 $\theta = \frac{\pi}{2}$ （即 θ 是 $90°$ ）。换句话说，这些是特征向量与 $\mathbf{w}$ 正交的数据点。我们可以添加一条与 $\mathbf{w}$ 正交的蓝色虚线，位于这条线上的任何特征向量的激活值都将等于0。

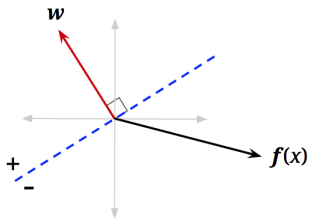


决策边界

我们将这条蓝色线称为决策边界，因为它是将我们将数据点分类为正类的区域与负类区域分隔开的边界。在更高维度中，线性决策边界通常称为超平面。超平面是一个线性曲面，其维度比潜在空间低一维，从而将该曲面一分为二。对于一般的分类器（非线性分类器），决策边界可能不是线性的，而是简单地定义为特征向量空间中分隔类别的曲面。为了对落在决策边界上的点进行分类，我们可以应用任意一个标签，因为两个类别都是同样有效的（在下面的算法中，我们将把线上的点分类为正类）。



x 分类为正类



x 分类为负类

二元感知机

很好，现在你知道线性分类器是如何工作的了，但我们如何构建一个好的线性分类器呢？构建分类器时，你从带有正确类别的数据开始，我们将其称为训练集。你通过在训练数据上评估分类器，将其与训练标签进行比较，并调整分类器的参数，直到达到目标来构建分类器。

让我们探讨一种简单线性分类器的具体实现：二元感知机。感知机是一种二元分类器——尽管它可以扩展到处理两个以上的类别。二元感知机的目标是找到一个能完美分离训练数据的决策边界。换句话说，我们在寻找最佳权重——最佳的w——使得任何与权重相乘的特征训练点都能被完美分类。

算法
感知机算法的工作原理如下：

1. 将所有权重初始化为 0：$\mathbf{w} = \mathbf{0}$
2. 对于每个训练样本，其特征为 $\mathbf{f}(\mathbf{x})$ 且真实类别标签为 $y^* \in \{-1, +1\}$，执行以下操作：

   (a) 使用当前权重对样本进行分类，设 y 为当前 $\mathbf{w}$ 预测的类别：

$$y = \text{classify}(x) = \begin{cases} +1 & \text{if } h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{f}(\mathbf{x}) > 0 \\ -1 & \text{if } h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{f}(\mathbf{x}) < 0 \end{cases}$$

   (b) 将预测标签 y 与真实标签 y^* 进行比较：
   - 如果 $y = y^*$，则不进行任何操作
   - 否则，如果 $y \neq y^*$，那么更新你的权重：$\mathbf{w} \leftarrow \mathbf{w} + y^*\mathbf{f}(\mathbf{x})$

3. 如果遍历了所有训练样本而无需更新权重（所有样本都被正确预测），则终止。否则，重复步骤2

更新权重
让我们研究并说明更新权重的过程。回想一下，在上面的步骤2b中，当我们的分类器正确时，什么都不会改变。但是当我们的分类器错误时，权重向量会按如下方式更新：

$$\mathbf{w} \leftarrow \mathbf{w} + y^*\mathbf{f}(\mathbf{x})$$

其中 y^* 是真实标签，取值为1或 -1， $\mathbf{x}$ 是我们误分类的训练样本。你可以将此更新规则解释为：

情况1：将正样本误分类为负样本 $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(\mathbf{x})$

案例2：将误分类的阴性判定为阳性 $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{f}(\mathbf{x})$

为什么这会起作用呢？一种看待它的方式是将其视为一种平衡行为。错误分类会在以下两种情况下发生：一种是训练样本的激活值远小于其应有的值（导致情况1错误分类），另一种是远大于其应有的值（导致情况2错误分类）。
考虑情况1，即激活值本应为正时却为负。换句话说，激活值太小。我们对 $\mathbf{w}$ 的调整应致力于解决此问题，并使该训练样本的激活值增大。为了让自己相信我们的更新规则 $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}\left(\mathbf{x}\right)$ 能做到这一点，让我们更新 $\mathbf{w}$ 并看看激活值是如何变化的。

$$h_{\mathbf{w}+\mathbf{f}(\mathbf{x})}(\mathbf{x}) = (\mathbf{w}+\mathbf{f}(x))^T\mathbf{f}(\mathbf{x}) = \mathbf{w}^T\mathbf{f}(\mathbf{x}) + \mathbf{f}(\mathbf{x})^T\mathbf{f}(\mathbf{x}) = h_{\mathbf{w}}(\mathbf{x}) + \mathbf{f}(\mathbf{x})^T\mathbf{f}(\mathbf{x})$$

使用我们的更新规则，我们发现新的激活值增加了 $\mathbf{f}\left(\mathbf{x}\right)^T\mathbf{f}\left(\mathbf{x}\right)$ ，这是一个正数，因此表明我们的更新是合理的。激活值在变大——越来越接近变为正数。当分类器误分类是因为激活值太大（激活值在应该为负时为正）时，你可以重复相同的逻辑。你会发现更新会使新的激活值减少 $\mathbf{f}\left(\mathbf{x}\right)^T\mathbf{f}\left(\mathbf{x}\right)$ ，从而变得更小并更接近正确分类。

虽然这清楚地说明了我们为什么要加上和减去某些东西，但我们为什么要加上和减去样本点的特征呢？一种理解方式是，权重并不是决定这个分数的唯一因素。分数是通过将权重与相关样本相乘来确定的。这意味着样本的某些部分比其他部分贡献更大。考虑以下情况，其中 x 是我们得到的带有真实标签 $y^* = -1$ 的训练样本：

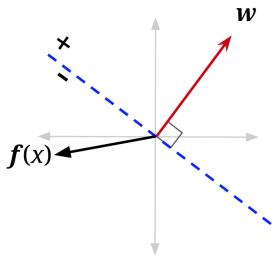
$$\mathbf{w}^T = \begin{bmatrix}2 & 2 & 2\end{bmatrix}, \mathbf{f}(\mathbf{x}) = \begin{bmatrix}4\\0\\1\end{bmatrix} \qquad h_{\mathbf{w}}(\mathbf{x}) = (2*4) + (2*0) + (2*1) = 10$$

我们知道我们的权重需要更小，因为激活需要为负才能正确分类。不过，我们不想对它们进行相同程度的改变。你会注意到，我们样本的第一个元素，即4，对我们10分的得分贡献比第三个元素大得多，而第二个元素根本没有贡献。那么，合适的权重更新应该是大幅改变第一个权重，小幅改变第三个权重，而第二个权重根本不应该改变。毕竟，第二个和第三个权重可能甚至没有那么糟糕，我们不想修复没坏的东西！

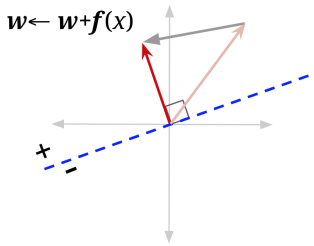
当思考改变我们的权重向量以满足上述需求的好方法时，结果发现仅使用样本本身实际上就能达到我们的目的；它会大幅改变第一个权重，小幅改变第三个权重，而完全不改变第二个权重！

可视化可能也会有所帮助。在下图中，$\mathbf{f}\left(\mathbf{x}\right)$ 是一个具有正类 $\left(y^* = +1\right)$ 的数据点的特征向量，该数据点当前被误分类——它位于由“旧w”定义的决策边界的错误一侧。将其添加到权重向量会产生一个与 $\mathbf{f}\left(\mathbf{x}\right)$ 夹角更小的新权重向量。它还会移动决策边界。在这个例子中，它已经将决策边界移动得足够多，以至于 x 现在将被正确分类（请注意，错误并不总是能被修正——这取决于权重向量的大小以及 $\mathbf{f}\left(\mathbf{x}\right)$ 当前超出边界的程度）。

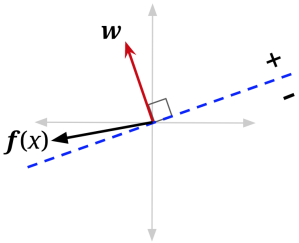
偏差
如果你尝试根据到目前为止所提到的内容来实现一个感知机，你会注意到一个特别不友好的怪癖。你最终绘制的任何决策边界都会穿过原点。基本上，你的感知机只能产生一个可以由函数 $\mathbf{w}^\top\mathbf{f}\left(\mathbf{x}\right) = 0, \mathbf{w}, \mathbf{f}\left(\mathbf{x}\right) \in \mathbb{R}^n$ 表示的决策边界。问题在于，即使在存在线性决策边界的问题中



Mis-classifying x with old w



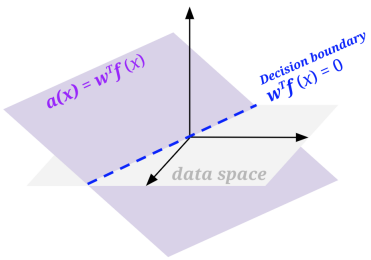
Updating w



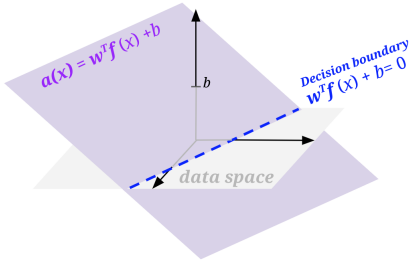
Updated classification of x

在数据中分隔正类和负类的那个边界可能不经过原点，而我们希望能够画出那些线。

为此，我们将修改我们的特征和权重以添加一个偏差项：在你的样本特征向量中添加一个始终为1的特征，并在你的权重向量中为这个特征添加一个额外的权重。这样做本质上使我们能够生成一个由 $\mathbf{w}^\top \mathbf{f}(\mathbf{x}) + b = 0$ 表示的决策边界，其中 b 是加权偏差项（即1乘以权重向量中的最后一个权重）。
从几何角度来看，我们可以通过思考激活函数在 $\mathbf{w}^\top \mathbf{f}(\mathbf{x})$ 时以及存在偏差 $\mathbf{w}^\top \mathbf{f}(\mathbf{x}) + b$ 时的样子来进行可视化。为此，我们需要比特征化数据的空间（如下 figures 中的标记数据空间）高一个维度。在上述所有部分中，我们一直只看数据空间的平面图。



Without bias



With bias

示例
让我们来看一个逐步运行感知机算法的示例。
让我们使用感知机算法对数据进行一次遍历，按顺序处理每个数据点。我们将从权重向量 $[w_0, w_1, w_2] = [-1, 0, 0]$ 开始（其中 w_0 是偏差特征的权重，记住它始终为 1）。

训练集

1	1	1	-
2	3	2	+
3	2	4	+
4	3	4	+
5	2	3	-

单感知机更新步骤

step	Weights	Score	Correct?	Update
1	[-1, 0, 0]	$-1\cdot 1+0\cdot 1+0\cdot 1=-1$	yes	none
2	[-1, 0, 0]	$-1\cdot 1+0\cdot 3+0\cdot 2=-1$	no	+[1, 3, 2]
3	[0, 3, 2]	$0\cdot 1+3\cdot 2+2\cdot 4=14$	yes	none
4	[0, 3, 2]	$0\cdot 1+3\cdot 3+2\cdot 4=17$	yes	none
5	[0, 3, 2]	$0\cdot 1+3\cdot 2+2\cdot 3=12$	no	-[1, 2, 3]
6	[-1, 1, -1]			

我们先讲到这里，但实际上在所有数据点都能在单次遍历中被正确分类之前，这个算法会对数据进行更多次的遍历。

多类感知机

上述感知机是一个二分类器，但我们可以很容易地将其扩展以处理多个类别。主要区别在于我们如何设置权重以及如何更新这些权重。对于二分类情况，我们有一个权重向量，其维度等于特征数量（加上偏置特征）。对于多类情况，我们将为每个类别都有一个权重向量，所以在三分类情况下，我们有3个权重向量。为了对一个样本进行分类，我们通过计算特征向量与每个权重向量的点积来为每个类别计算一个分数。得分最高的类别就是我们选择作为预测结果的类别。

例如，考虑三分类的情况。设我们的样本具有特征 $\mathbf{f}(\mathbf{x}) = \begin{bmatrix} -2 & 3 & 1 \end{bmatrix}$，并且我们为类别0、1和2设置的权重分别为：

$$\mathbf{w}_0 = \begin{bmatrix} -2 & 2 & 1 \end{bmatrix}$$
$$\mathbf{w}_1 = \begin{bmatrix} 0 & 3 & 4 \end{bmatrix}$$
$$\mathbf{w}_2 = \begin{bmatrix} 1 & 4 & -2 \end{bmatrix}$$

对每个类别进行点积运算会得到分数 $s_0 = 11, s_1 = 13, s_2 = 8$。因此，我们会预测 $\mathbf{x}$ 属于类别1。

需要注意的一个重要事项是，在实际实现中，我们不会将权重作为单独的结构来跟踪，而是通常将它们堆叠在一起以创建一个权重矩阵。这样，我们无需像类别数量那样进行多次点积运算，而是可以进行一次矩阵与向量的乘法运算。在实践中，这往往效率更高（因为矩阵与向量的乘法运算通常有高度优化的实现）。

在我们上述的例子中，那将是：

$$\mathbf{W} = \begin{bmatrix} -2 & 2 & 1 \\ 0 & 3 & 4 \\ 1 & 4 & -2 \end{bmatrix}, \mathbf{x} = \begin{bmatrix} -2 \\ 3 \\ 1 \end{bmatrix}$$

并且我们的标签将是：

$$\arg\max(\mathbf{W}\mathbf{x}) = \arg\max(\begin{bmatrix} 11 \\ 13 \\ 8 \end{bmatrix}) = 1$$

当我们转向多类别情况时，随着权重结构的变化，权重更新也会改变。如果我们正确地对数据点进行分类，那么就像在二分类情况下一样不做任何操作。如果我们选错了，比如说我们选择了类别 $y \neq y^*$，那么我们将特征向量加到真实类别的权重向量上，加到 y^*，并从与预测类别 y 对应的权重向量中减去该特征向量。在我们上面的例子中，假设正确类别是类别2，但我们预测的是类别1。现在我们将对应类别1的权重向量从中减去 x，

$$\mathbf{w}_1 = \begin{bmatrix} 0 & 3 & 4 \end{bmatrix} - \begin{bmatrix} -2 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 3 \end{bmatrix}$$

接下来，我们获取对应正确类别的权重向量，在我们的例子中是类别2，并向其中添加 x：

$$\mathbf{w}_2 = \begin{bmatrix} 1 & 4 & -2 \end{bmatrix} + \begin{bmatrix} -2 & 3 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 7 & -1 \end{bmatrix}$$

这相当于“奖励”正确的权重向量，“惩罚”有误导性的、错误的权重向量，而其他权重向量则保持不变。

考虑到权重和权重更新的差异，算法的其余部分基本相同；遍历每个样本点，当出现错误时更新权重，直到不再出错。
为了纳入偏差项，做法与二元感知机相同——给每个特征向量添加一个值为1的额外特征，并给每个类别的权重向量添加一个针对此特征的额外权重（这相当于在矩阵形式中添加一列）。