目录

[Linear classification 2](#_Toc59134308)

[Loss function 6](#_Toc59134309)

[Practical Considerations 10](#_Toc59134310)

[Softmax classifier 11](#_Toc59134311)

[SVM vs. Softmax 13](#_Toc59134312)

[Summary 15](#_Toc59134313)

# Linear classification

针对KNN classifier的缺点，提出了一个更强大的方法对于图像分类问题，并将其扩展到NN和CNN.

包含两个部分:

score function(map the raw data to class scores) 将原始数据映射为类的分数

loss function(quantify the agreement between the predicted scores and the ground truth labels) 量化预测的分数和实际标签之间的一致性。

最后将其视为一个优化 (Optimization) 问题，通过调整 score function 的参数来使loss function 达到最小值。

**从图像到标签分值的参数化映射Parameterized mapping from images to label scores:**

假设有一个包含很多图像的训练集 ，每个图像都有一个对应的分类标签  。这里 i=1,2,…,N 并且  。这就是说，有 N 个图像样本，每个图像的维度是 D，共有 K种不同的分类。举例来说，在 CIFAR-10 中，我们有一个 **N**=50000 的训练集，每个图像有 **D**= 32x32x3=3072 个像素，而 **K**=10 ，这是因为图片被分为 10 个不同的类别（狗，猫，汽车等）。现在定义映射函数为：，该函数是原始图像像素到分类分值的映射。

**线性分类器Linear classifier：**

其中： 为图像像素（展开为列向量），[D\*1]

W—权重矩阵weight，[K\*D]

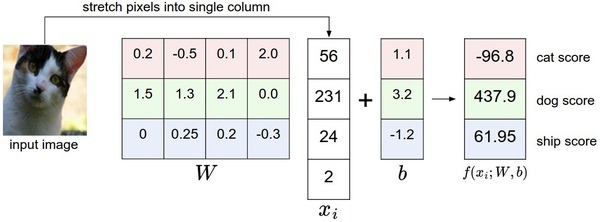
b—偏置向量bias vector, [K\*1] 影响输出数值，但是并不和原始数据产生关联

值得注意的地方：

* 矩阵乘法高效地并行评估10个不同的分类, 其中每个类的参数就是W的一个行向量。(W的每一行都是一个分类器)
* 输入数据 是给定且不可改变的，但参数 W 和 b 是可控制改变的。目标就是通过设置这些参数，使得计算出来的分类分值和训练集中图像数据的真实类别相符。
* 该方法的一个优势是训练数据是用来学习到参数 W 和 b 的，一旦训练完成，训练数据就可以丢弃，留下学习到的参数即可。这是因为一个测试图像可以简单地输入函数，并基于计算出的分类分值来进行分类。
* 得到参数后，只需要做一个矩阵乘法和一个矩阵加法就能对一个测试数据分类，这比 KNN 这种非参方法，每次预测都要读取所有数据集快得多。

**理解线性分类器Interpreting a linear classifier：**

线性分类器计算图像中 3 个颜色通道中所有像素的值与 W 相乘，从而得到预测值。根据训练得到的权重，对于图像中的某些位置的某些颜色，函数表现出喜好或者厌恶（根据每个权重的符号而定）。举个例子，可以想象“船”分类就是被大量的蓝色所包围（对应的就是水）。那么“船”分类器在蓝色通道上的权重就有很多的正权重（它们的出现提高了“船”分类的分值），而在绿色和红色通道上的权重为负的就比较多（它们的出现降低了“船”分类的分值）。

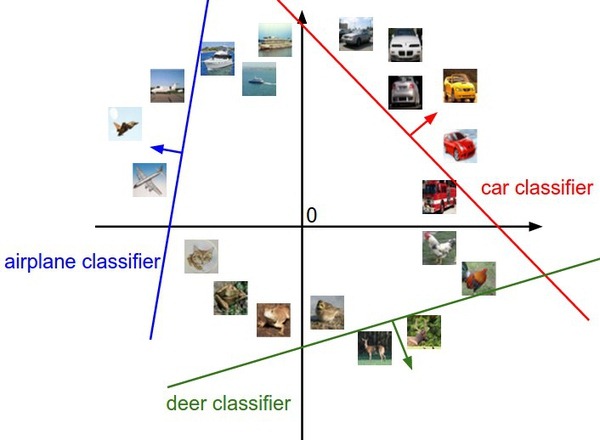


这里举个例子，假设单通道图像只有 4 个像素，有 3 个分类，分别为猫，狗，船。首先将图像像素拉伸为一个列向量，与W进行矩阵乘，然后得到各个分类的分值。需要注意的是，这个 W 一点也不好：猫分类的分值非常低。从上图来看，分类结果为狗。

**将图像看做高维度的点Analogy of images as high-dimensional points：**

既然图像被伸展成为了一个高维度的列向量，那么我们可以把图像看做这个高维度空间中的一个点（即每张图像是 3072 维空间中的一个点）。整个数据集就是一个点的集合，每个点都带有 1 个分类标签。

既然定义每个分类类别的分值是权重和图像的矩阵乘，那么每个分类类别的分数就是这个空间中的一个线性函数的函数值。我们没办法可视化 3072 维空间中的线性函数，但假设把这些维度挤压到二维，那么就可以看看这些分类器在做什么了：



图像空间的示意图。其中每个图像是一个点，有 3 个分类器。以红色的汽车分类器为例，红线表示空间中汽车分类分数为0的点的集合，红色的箭头表示分值上升的方向。所有红线右边的点的分数值均为正，且线性升高。红线左边的点分值为负，且线性降低。

从上面可以看到，**W**的每一行都是一个分类类别的分类器。对于这些数字的几何解释是：如果改变其中一行的数字，会看见分类器在空间中对应的直线开始向着不同方向旋转。而偏差 **b**，则允许分类器对应的直线平移。需要注意的是，如果没有偏差，无论权重如何，在 时分类分值始终为 0 。这样所有分类器的线都不得不穿过原点。

**将线性分类器看做模板匹配Interpretation of linear classifiers as template matching：**

关于权重 W  的另一个解释是它的每一行对应着一个分类的模板template。一张图像对应不同分类的得分，是通过使用内积来比较图像和模板，然后找到和哪个模板最相似。从这个角度来看，线性分类器就是在利用学习到的模板，针对图像做模板匹配。从另一个角度来看，可以认为还是在高效地使用 KNN ，不同的是我们没有使用所有的训练集的图像来比较，而是每个类别只用了一张图片（这张图片是我们学习到的，而不是训练集中的某一张），而且我们会使用（负）内积来计算向量间的距离，而不是使用 L1 或者 L2 距离。



这里展示的是以CIFAR-10为训练集，学习结束后的权重的例子。注意，船的模板如期望的那样有很多蓝色像素。如果图像是一艘船行驶在大海上，那么这个模板利用内积计算图像将给出很高的分数。

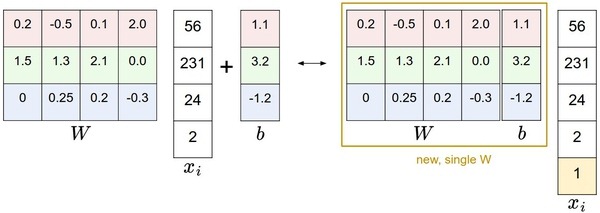
可以看到马的模板看起来似乎是两个头的马，这是因为训练集中的马的图像中马头朝向各有左右造成的。线性分类器将这两种情况融合到一起了。类似的，汽车的模板看起来也是将几个不同的模型融合到了一个模板中，并以此来分辨不同方向不同颜色的汽车。这个模板上的车是红色的，这是因为 CIFAR-10 中训练集的车大多是红色的。线性分类器对于不同颜色的车的分类能力是很弱的，但是后面可以看到神经网络是可以完成这一任务的。神经网络可以在它的隐藏层中实现中间神经元来探测不同种类的车（比如绿色车头向左，蓝色车头向前等）。而下一层的神经元通过计算不同的汽车探测器的权重和，将这些合并为一个更精确的汽车分类得分。

**偏差（和权重的）技巧Bias trick：**

在进一步学习前，要提一下这个经常使用的技巧。它能够将我们常用的参数 W 和 b 合二为一。回忆一下，分类评分函数定义为：

分开处理这两个参数（权重参数 W 和偏差参数 b）有点笨拙，一般常用的方法是把两个参数放到同一个矩阵中，同时向量就要增加一个维度，这个维度的数值是常量 1 ，这就是默认的偏差维度。这样新的公式就简化成下面这样：

还是以 CIFAR-10 为例，那么的大小就变成 [3073x1] ，而不是 [3072x1] 了，多出了包含常量 1 的 1 个维度）。W 大小就是 [10x3073] 了。W 中多出来的这一列对应的就是偏差值 b，具体见下图：



偏差技巧的示意图。左边是先做矩阵乘法然后做加法，右边是将所有输入向量的维度增加 1 个含常量 1 的维度，并且在权重矩阵中增加一个偏差列，最后做一个矩阵乘法即可。左右是等价的。通过右边这样做，我们就只需要学习一个权重矩阵，而不用去学习两个分别装着权重和偏差矩阵了。

**图像数据预处理Image data preprocessing**：

在上面的例子中，所有图像都是使用的原始像素值（从 0 到 255 ）。在机器学习中，对于输入的特征做归一化（normalization）处理是很常见的。而在图像分类的例子中，图像上的每个像素可以看作一个特征。在实践中，对每个特征减去平均值来**中心化**数据是非常重要的。在这些图片的例子中，该步骤意味着根据训练集中所有的图像计算出一个平均图像值，然后每个图像都减去这个平均值，这样图像的像素值就大约分布在 [-127, 127] 之间了。下一个常见步骤是，让所有数值分布的区间变为 [-1, 1] 。**零均值的中心化**是很重要的，这样损失函数在梯度下降时得到的是一个很规则的形状。

# Loss function

在假设函数中，训练数据 是给定的，不能修改。但是可以调整权重矩阵的参数，使得假设函数的结果与训练数据集中图像的真实类别一致，即假设函数在对应的正确分类应当有最高的得分。

回到之前那张猫的图像分类例子，有针对猫，狗，船三个类别的分数。例子中权重值非常差，因为猫分类的得分非常低（-96.8），而狗（437.9）和船（61.95）比较高。所以这里引入**损失函数（Loss Function，Cost Function）**来衡量对结果的不满意程度。直观地讲，当假设函数输出结果与真实结果之间差异越大，损失函数输出越大，反之越小。

**Multiclass Support Vector Machine loss多分类支持向量机损失函数**

损失函数的形式多种多样，比如说使用 Hinge Loss 得到的分类器就是 SVM 。SVM 的损失函数想要 SVM 在正确分类上的得分始终比不正确分类上的得分高出一个边界值。

对于训练样本  和标签  ，假设函数通过公式   来计算不同类别的得分。这里将得分简写为向量s 。比如，针对第 j 个类别的得分就是第 j 个元素：   。针对第i个数据的多类 SVM 的损失函数定义如下：

**举例**：假设有 3 个分类，并且得到了分值 s=[13,−7,11]。其中第一个类别是正确类别，即  。同时假设 Δ 是 10 。上面的公式是将所有不正确分类 加起来，所以例子中有两个分量：

可以看到第一个部分结果是 0 ，这是因为 -7-13+10 得到的是负数，经过 max(0,-) 函数处理后得到 0 。这一对类别分数和标签的损失值是 0，这是因为正确分类的得分 13 与错误分类的得分 –7 的差为 20 ，高于边界值 10 。而 SVM 只关心差距至少要大于 10 ，更大的差值还是算作损失值为 0 。第二个部分计算 11-13+10 得到 8 。虽然正确分类的得分比不正确分类的得分要高（13>11），但是比 10 的边界值还是小了，分差只有 2 ，这就是为什么损失值等于 8 。简而言之：**Hinge Loss 想要正确分类类别  的分数比不正确类别分数高，而且至少要高**Δ。如果不满足这点，损失就会纳入整体的损失中。

对于的是线性假设函数, 可以将损失函数的公式稍微改写一下：

其中 权重 **W** 的第 j 行，被变形为列向量。然而，一旦开始考虑更复杂的假设函数f公式，这不必这样做了。

还有一种 Square Hinge Loss SVM（即 L2-SVM ），它使用的是，将更强烈（平方地而不是线性地）地惩罚过界的值。，在某些数据集中，Square Hinge Loss 会工作得更好。可以通过交叉验证来决定到底使用哪个。

对于预测训练集数据分类标签的情况总有一些不满意的，而损失函数就能将这些不满意的程度量化。



SVM 这是一种最大间隔分类的思想，“想要”正确类别的分类得分比其他不正确分类类别的分数要高，而且至少高出 delta 那么大。 如果其他分类分数进入了红色的区域，甚至更高，那么就开始计算损失。如果没有这些情况，损失值为0。我们的目标是找到一些权重，它们既能够让训练集中的数据样例满足这些限制，也能让总的损失值尽可能地低。

**Regularization正则化**

假设有一个数据集和一个权重集 W能够正确地分类每个数据（即所有的边界都满足，对于所有的 i 都有 Li=0）。问题在于这个 W 并不唯一：可能有很多相似的 W都能正确地分类所有的数据。一个简单的例子：如果 W能够正确分类所有数据，即对于每个数据，损失值都是 0 。那么当 λ>1 时，任何数乘 λW 都能使得损失值为 0 ，因为这个变化将所有分值的大小都均等地扩大了，所以它们之间的绝对差值也扩大了。举个例子，如果一个正确分类的分值和距离它最近的错误分类的分值的差距是 15 ，对W扩大2 倍使得差距变成 30，差值肯定是比Δ 越来越大。

换句话说，我们希望能向某些特定的权重 W添加一些偏好，对其他权重则不添加，以此来消除模糊性。这一点是能够实现的，方法是向损失函数增加一个**正则化惩罚（regularization penalty）**R(W)部分。最常用的正则化惩罚是 L2 范式，L2 范式通过对所有参数进行逐元素的平方惩罚来抑制大数值的权重：

引入正则化惩罚后，就可以给出完整的多类 SVM 损失了，它由两个部分组成：**数据损失（data loss）**，即所有样例的的平均损失  ，以及**正则化损失（regularization loss）**。完整公式如下所示：

将其展开为完整公式为：

其中，N 是训练集的数据量。现在正则化惩罚添加到了损失函数里面，并用超参数 λλ 来计算其权重。该超参数无法简单确定，需要通过交叉验证来获取。

引入了L2惩罚后，SVM 就有了**最大边界（max margin）**这一良好性质。

其中最好的性质就是对大数值权重进行惩罚，可以提升其泛化能力，因为这就意味着没有哪个维度能够独自对于整体分值有过大的影响。举个例子，假设输入向量 x=[1,1,1,1]，两个权重向量=[1,0,0,0] ， =[0.25,0.25,0.25,0.25]。那么 ，两个权重向量都得到同样的内积，但是的 L2 惩罚是 1.0，而的 L2 惩罚是 0.25 。因此，根据 L2 惩罚来看, 更好，因为它的正则化损失更小。这是因为的权重值更小且更分散。既然 L2 惩罚倾向于更小更分散的权重向量，这就使得分类器最终将所有维度上的特征都用起来，而不是强烈依赖其中少数几个维度。在后面的课程中可以看到，这一效果将会提升分类器的泛化能力，并避免**过拟合**。

需要注意的是，和权重不同，偏差没有这样的效果，因为它们并不控制输入维度上的影响强度。因此通常只对权重 W 正则化，而不正则化偏差 b 。在实际操作中，可发现这一操作的影响可忽略不计。而且由于正则化惩罚的存在，不可能在所有的例子中得到0的损失值，这是因为只有当 W=0W=0 的特殊情况下，才能得到损失值为 0 。

**代码**：下面是一个无正则化部分的损失函数的 Python 实现，有非向量化和半向量化两个形式：

**def** **L\_i**(x, y, W):

"""

unvectorized version. Compute the multiclass svm loss for a single example (x,y)

- x is a column vector representing an image (e.g. 3073 x 1 in CIFAR-10)

with an appended bias dimension in the 3073-rd position (i.e. bias trick)

- y is an integer giving index of correct class (e.g. between 0 and 9 in CIFAR-10)

- W is the weight matrix (e.g. 10 x 3073 in CIFAR-10)

"""

delta **=** 1.0 *# see notes about delta later in this section*

scores **=** W.dot(x) *# scores becomes of size 10 x 1, the scores for each class*

correct\_class\_score **=** scores[y]

D **=** W.shape[0] *# number of classes, e.g. 10*

loss\_i **=** 0.0

**for** j **in** range(D): *# iterate over all wrong classes*

**if** j **==** y:

*# skip for the true class to only loop over incorrect classes*

**continue**

*# accumulate loss for the i-th example*

loss\_i **+=** max(0, scores[j] **-** correct\_class\_score **+** delta)

**return** loss\_i

**def** **L\_i\_vectorized**(x, y, W):

"""

A faster half-vectorized implementation. half-vectorized

refers to the fact that for a single example the implementation contains

no for loops, but there is still one loop over the examples (outside this function)

"""

delta **=** 1.0

scores **=** W.dot(x)

*# compute the margins for all classes in one vector operation*

margins **=** np.maximum(0, scores **-** scores[y] **+** delta)

*# on y-th position scores[y] - scores[y] canceled and gave delta. We want*

*# to ignore the y-th position and only consider margin on max wrong class*

margins[y] **=** 0

loss\_i **=** np.sum(margins)

**return** loss\_i

**def** **L**(X, y, W):

"""

fully-vectorized implementation :

- X holds all the training examples as columns (e.g. 3073 x 50,000 in CIFAR-10)

- y is array of integers specifying correct class (e.g. 50,000-D array)

- W are weights (e.g. 10 x 3073)

"""

*# evaluate loss over all examples in X without using any for loops*

*# left as exercise to reader in the assignment*

在本小节的学习中，SVM 损失采取了一种特殊的方法，使得能够衡量对于训练数据预测分类和实际分类标签的一致性。还有，对训练集中数据做出准确分类预测和让损失值最小化这两件事是等价的。接下来要做的，就是找到能够使损失值最小化的权重了。

# Practical Considerations

**Setting Delta设置delta**

你可能注意到上面的内容对超参数 Δ及其设置是一笔带过，那么它应该被设置成什么值？需要通过交叉验证来求得吗？现在看来，该超参数在绝大多数情况下设为 Δ=1 都是安全的。超参数 Δ 和 λ 是两个不同的超参数，但实际上他们一起控制同一个权衡：即损失函数中的数据损失和正则化损失之间的权衡。理解这一点的关键是要知道，权重 W 的大小对于分类分值有直接影响（当然对他们的差异也有直接影响）：当我们将 W 中的值缩小，分类分值之间的差异也变小，反之亦然。因此，不同分类分值之间的边界的具体值（比如 Δ=1或 Δ=100 ）从某些角度来看是没意义的，因为权重自己就可以控制差异变大和缩小。也就是说，真正的权衡是我们允许权重能够变大到何种程度（通过正则化强度 λ 来控制）。

**Relation to Binary Support Vector Machine**.与二元SVM的关系

二分支持向量机对于第 i 个数据的损失计算公式：

其中，C是一个超参数，并且

可以认为本章节介绍的 SVM 公式包含了上述公式，上述公式是多类支持向量机公式只有两个分类类别的特例。也就是说，如果我们要分类的类别只有两个，那么公式就化为二分 SVM 的公式。 这里 Δ=1 且代表了几何间隔是否大于 1 ，大于 1  则损失为 0 ， C 和多类 SVM 公式中的 λ 都控制着同样的权衡，而且它们之间的关系是 C∝1/λ.

# Softmax classifier

除了 SVM ，还有一个比较常见分类器就是 **Softmax**了，SVM 将输出作为每个分类的评分（因为无定标，所以难以直接解释）。与 SVM 不同，Softmax 的输出（归一化的分类概率）更加直观，并且从概率上可以解释。在 Softmax 分类器中，函数映射保持不变，但将这些评分值视为每个分类的未归一化的对数概率，并且将 *hinge loss*替换为**交叉熵损失**（**cross-entropy loss）**。公式如下：

在上式中，使用 来表示分类评分向量 f 中的第 j 个元素。和之前一样，整个数据集的损失值是数据集中所有样本数据的损失值的均值与正则项 R(W) 之和。假设函数的形式为：

该函数也被称作 **softmax 函数**：其输入值是一个向量，向量中元素为任意实数的评分值（z中的），函数对其进行压缩，输出一个向量，其中每个元素值在0到1之间，且所有元素之和为1。所以，包含 softmax 函数的完整交叉熵损失看起唬人，实际上还是比较容易理解的。

**Softmax的信息理论视角Information theory view**

在“真实”分布p和估计分布q之间的*交叉熵*定义如下：

因此，Softmax 分类器所做的就是最小化在估计分类概率（就是上面的 ）和“真实”分布之间的交叉熵，在这个解释中，“真实”分布就是所有概率密度都分布在正确的类别上（比如： p=[0,…,1,…,0] 中在的位置就有一个单独的1 ）。交叉熵还可以写成熵和相对熵的形式：

相对熵（Relative Entropy）也叫做 Kullback-Leibler 差异（Kullback-Leibler Divergence），它衡量的是相同事件空间里的两个概率分布的差异情况。如果 p 的熵是 0 ，那么就能等价的看做是对两个分布之间的相对熵做最小化操作。换句话说，交叉熵损失函数“想要”预测分布的所有概率密度都在正确分类上。

**Softmax的概率解释Probabilistic interpretation**

先看下面的公式：

可以解释为是给定图像数据，以 W 为参数，分配给正确分类标签  的归一化概率。为了理解这点，请回忆一下 Softmax 分类器将输出向量 f 中的评分值解释为没有归一化的对数概率。那么以这些数值做指数函数的幂就得到了没有归一化的概率，而除法操作则对数据进行了归一化处理，使得这些概率的和为1。从概率论的角度来理解，我们就是在最小化正确分类的负对数概率，这可以看做是在进行*最大似然估计*（MLE）。该解释的另一个好处是，损失函数中的正则化部分 R(W) 可以被看做是权重矩阵 W 的高斯先验，这里进行的是最大后验估计（MAP）而不是最大似然估计。

编程实现 softmax 函数计算的时候，中间项和因为存在指数函数，所以数值可能非常大。除以大数值可能导致数值计算的不稳定，所以学会使用归一化技巧非常重要。如果在分式的分子和分母都乘以一个常数C，并把它变换到求和之中，就能得到一个从数学上等价的公式：

C 的值可自由选择，不会影响计算结果，通过使用这个技巧可以提高计算中的数值稳定性。通常将 C 设为。该技巧简单地说，就是应该将向量 f 中的数值进行平移，使得最大值为 0 。代码实现如下：

f **=** np.array([123, 456, 789]) *# example with 3 classes and each having large scores*

p **=** np.exp(f) **/** np.sum(np.exp(f)) *# Bad: Numeric problem, potential blowup*

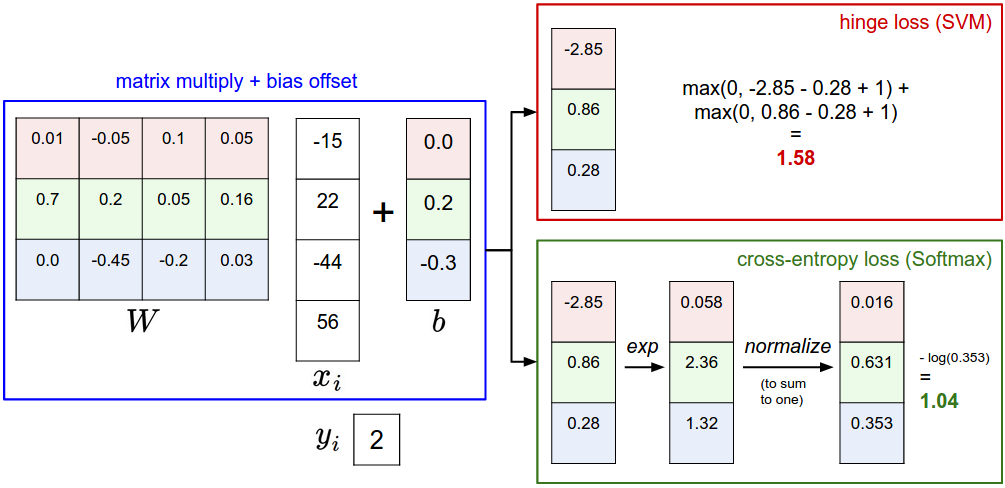
*# instead: first shift the values of f so that the highest number is 0:*

f **-=** np.max(f) *# f becomes [-666, -333, 0]*

p **=** np.exp(f) **/** np.sum(np.exp(f)) *# safe to do, gives the correct answer*

# SVM vs. Softmax

总结一下： SVM 分类器使用的是 *hinge loss ，*Softmax分类器使用的是 *corss-entropy loss*。Softmax分类器的命名是从 *softmax 函数*那里得来的，softmax 函数将原始分类评分变成正的归一化数值，所有数值和为 1 ，这样处理后交叉熵损失才能应用。注意从技术上说 softmax 映射本身是没有意义的，因为 softmax 只是一个压缩数值的函数。但是在这个说法常常被用来做简称，下图有助于区分这 Softmax 和 SVM 这两种分类器：



针对一个数据点，SVM 和 Softmax 分类器的不同处理方式的例子。两个分类器都计算了同样的分值向量 f（本节中是通过矩阵乘来实现）。不同之处在于对 f中分值的解释：SVM 分类器将它们看做是分类评分，它的损失函数鼓励正确的分类（本例中是蓝色的类别 2 的分值比其他分类的分值高出至少一个边界值。Softmax 分类器将这些数值看做是每个分类没有归一化的**对数概率**，鼓励正确分类的归一化的对数概率变高，其余的变低。SVM 的最终的损失值是 1.58 ，Softmax 的最终的损失值是 0.452 ，但要注意这两个数值没有可比性。只在给定同样数据，在同样的分类器的损失值计算中，它们才有意义。

**Softmax 分类器为每个分类提供了“可能性” Softmax classifier provides “probabilities” for each class.**

SVM 难以针对所有分类的评分值给出直观解释。Softmax 分类器则不同，它允许计算出对于所有分类标签的可能性。举个例子，针对给出的图像，SVM 分类器可能给你的是一个 对应分类“猫”，“狗”，“船”。而 softmax 分类器可以计算出这三个标签的”可能性“是 [0.9,0.09,0.01]，这就让你能看出对于不同分类准确性的把握。为什么我们要在”可能性“上面打引号呢？这是因为可能性分布的集中或离散程度是由正则化参数 λλ 直接决定的， λ 是你能直接控制的一个输入参数。举个例子，假设 3 个分类的原始分数是 ，那么 softmax 函数就会计算：

现在，如果正则化参数λ 更大，那么权重 W 就会被惩罚的更多，然后他的权重数值就会更小。这样算出来的分数也会更小，假设小了一半，那么softmax 函数的计算就是：

现在看起来，概率的分布就更加分散了。还有，随着正则化参数 λλ 不断增强，权重数值会越来越小，最后输出的概率会接近于均匀分布。这就是说，softmax 分类器算出来的概率最好是看成一种对于分类正确性的自信。和 SVM 一样，数字间相互比较得出的大小顺序是可以解释的，但其绝对值则难以直观解释**。**

**SVM 和 Softmax 经常是相似的 In practice, SVM and Softmax are usually comparable.**

通常说来，两种分类器的表现差别很小，不同的人对于哪个分类器更好有不同的看法。相对于 Softmax 分类器，SVM 更加“局部目标化（local objective）”，这既可以看做是一个特性，也可以看做是一个劣势。考虑一个评分是 [10,-2,3] 的数据，其中第一个分类是正确的。那么一个 SVM（Δ=1）会看到正确分类相较于不正确分类，已经得到了比边界值还要高的分数，它就会认为损失值是 0 。SVM 对于数字个体的细节是不关心的：如果分数是 [10,-100,-100] 或者 [10,9,9]，对于 SVM 来说没设么不同，只要满足超过边界值1 ，那么损失值就等于0。

对于 softmax 分类器，情况则不同。对于[10,9,9]来说，计算出的损失值就远远高于[10,-100,-100]的。换句话来说， softmax 分类器对于分数是永远不会满意的：正确分类总能得到更高的可能性，错误分类总能得到更低的可能性，损失值总是能够更小。但是，SVM 只要边界值被满足了就满意了，不会超过限制去细微地操作具体分数。这可以被看做是 SVM 的一种特性。举例说来，一个汽车的分类器应该把他的大量精力放在如何分辨小轿车和大卡车上，而不应该纠结于如何与青蛙进行区分，因为区分青蛙得到的评分已经足够低了。

# **Summary**

* 对于图像分类，首先定义从图像像素映射到不同类别的分类评分的评分函数。在本节中，评分函数是一个基于权重 W 和偏差 b 的线性函数。
* 与 KNN 分类器不同，参数方法的优势在于一旦通过训练学习到了参数，就可以将训练数据丢弃了。同时该方法对于新的测试数据的预测非常快，因为只需要与权重 W 进行一个矩阵乘法运算。
* 介绍了偏差技巧，让我们能够将偏差向量和权重矩阵合二为一，然后就可以只跟踪一个矩阵。
* 定义了损失函数（介绍了  SVM 和 Softmax 线性分类器最常用的 2 个损失函数）。损失函数能够衡量给出的参数集与训练集数据真实类别情况之间的一致性。在损失函数的定义中可以看到，对训练集数据做出良好预测与得到一个足够低的损失值这两件事是等价的。

现在我们知道了如何基于参数，将数据集中的图像映射成为分类的评分，也知道了两种不同的损失函数，它们都能用来衡量算法分类预测的质量。但是，如何高效地得到能够使损失值最小的参数呢？这个求得最优参数的过程被称为最优化，将在下篇文章中进行介绍。