CS231n

Liyifan

2023 年 2 月 2 日

目录

第一章　分类问题

相比于硬编码 (hard-code) 的方式，使用训练好的分类器进行测试具有更好的泛化能力。而这种方法的难点在于训练分类器的算法的准确性和高效性，并且训练数据集的构建需要很多的成本。

```
def train(images, labels):
  # Machine learning!
  return model
```

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

图 1.1: data-driven method

1.1　K 邻近算法 (KNearestNeighbour=KNN)

1.1.1　基本方法

对于每个测试向量，计算其与每一个训练向量的距离，找到最近的 k 个的训练样本并取其中出现次数最多的门类作为预测结果。
k 是一个可以调节的超参数。计算两个向量的距离的度量也可以被选择。具体问题具体分析，对于 $CIFAR-10$ 的 KNN 分类方法来说，“k=7” + “L2 度量” 可以达到最佳的效果。

1.1.2　代码框架

train

只是简单地把维度为 $(nums\_train \times dimension\_of\_img)$ 的向量 X 和与之对应的类别向量 y 传入分类器。

computation for distances

$$L_1 = \sum_p |I_a^p - I_b^p|$$

$$L_2 = \sqrt{\sum_p (I_a^p - I_b^p)^2}$$

在编写具体代码时，除了直接使用二重循环一个个地计算 dists 矩阵的元素值之外，还可以充分利用 numpy 库的矢量级加速。将测试数据与训练数据两两之间的距离以矩阵的形式给出，记为 $dists$(大小为 $num\_test \times num\_train$)。可以使用 $dists[i][j] = np.sqrt(np.sum((X[i]-self.X\_train[j])**2))$ 来计算。

一种矢量化的思路是将 X 和 X_train 利用广播机制扩展到 (num_test, num_train, Dimension) 的维度，然后直接求二者的 L_2 度量。这只需要把 X 进行 reshape，到 (num_test, 1 , Dimension) 即可。但是这对于 Dimension 较大的图像特征来说会导致内存超限。另外一种直观的方式是将计算式中的平方项展开。

$$\begin{aligned} & dists[i][j] \\ & \quad = \sqrt{\sum_{k=0}^{D-1} (X_{ik} - X\_train_{jk})^2} \\ & \quad = \sqrt{\sum_{k=0}^{D-1} (X_{ik}^2 + X\_train_{jk}^2 - 2X_{ik}X\_train_{jk})} \end{aligned} \tag{1.1}$$

其中的平方项可以通过把原来 X 和 X_train 按行求和之后利用广播机制求得。交叉项即是矩阵乘积。

```
Two loop version took 53.129210 seconds
One loop version took 33.900654 seconds
No loop version took 0.260223 seconds
```

图 1.2: fully_vectorized is much faster

prediction

dists 的每一行代表着一个测试数据，可以对这一行数进行排序，并找到最小的 k 个数的下标，这些下标对应着不同类别的训练数据。根据之前训练阶段传入的各个训练数据的类别得到离该测试数据最近的 k 个训练数据点中哪一个类别最多，那么就将之分类到那一类中。这里可以使用 np.argsort(list) 函数对 list 进行排序并且返回元素从小到大的下标的向量。之后可以使用 np.bincount 计数每个元素出现的次数。该函数返回一个记录元素出现次数的向量 count_times，下标为元素索引，向量值为出现次数。最后针对 count_times 使用 np.argmax() 函数找到对应的元素值，即为分类结果。

cross-validation

对于训练集，可以将之分成一部分作为训练，另一部分作为验证集，并且可以交叉式地划分，以验证模型的稳定性和有效性。

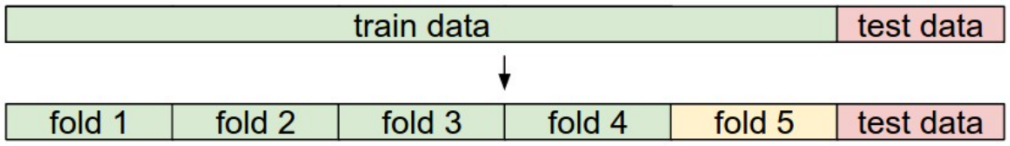

图 1.3: dataset$_split$

代码中，需要注意数据规格，可以使用 np.hstack(vstack) 来处理堆叠的 list，使之成为一个大的连续的 list。

1.1.3　缺点分析

1. 训练过程消耗空间过大

2. 预测过程所需时间过长

1.2　线性分类算法

1.2.1　基本方法

总体思路是：使用 score function 度量类别，使用 loss function 度量与真实类别之间的差距，将问题转换成一个优化问题 (Optimization)，即找到使 loss 最小的 score function 的参数。对于简单的线性分类器 (Linear Classifier)，要求其评分函数为 $f : R^D \longrightarrow R^k$，其中 D 表示数据维度，k 表示类别数量。于是可以使用权重矩阵 (weights)（参数矩阵）和偏置 (bias) 来刻画这一度量，即

$$f(x_i, W, b) = W \cdot x_i + b \tag{1.2}$$

注意到 W 矩阵的每一行都对应一个类别的线性函数，即不同类别的线性度量是分离的。

这种参数化方法比 KNN 的好处在于训练和预测所需成本都有所降低。难点在于找到合适的方法调整权重和偏置使得 loss 尽可能小。

在处理权重和偏置的时候，可以使用一个小技巧将 W 和 b 融合在一起而不是笨重地被分离成两部分。可以将每一个 x_i 最后一行加上一个 1，把 W 最后一列变成 b，这样计算出来的列向量与原来一样。

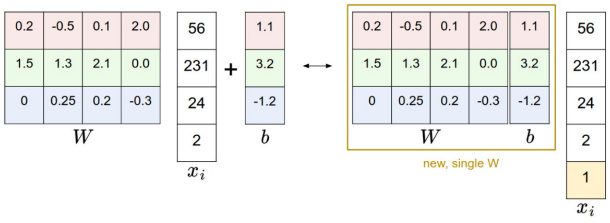

图 1.4: Illustration of bias trick

除此之外，还可以让每个数据减去总体的平均值，使得数据变得更加接近。这被称为数据预处理，即零均值的中心化。

1.2.2　思路解释

高维数据空间划分

将每一个训练数据视为 D 维空间中的一个数据点，线性分类权重矩阵所做的就是用一个线性变换将不同类别的分割面旋转，并用偏置将之平移使其跨过原点，得到该空间的划分。

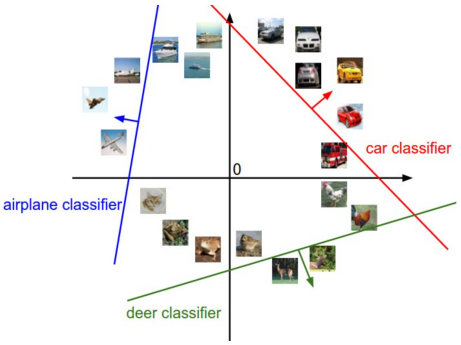

图 1.5: visualized classifier

模式识别

与之前 KNN 想法类似，但是不用 L1 或 L2 度量数据的距离，而是使用 weights 作为中间模板，使用向量内积作为度量。使用 CIFAR-10 训练完成后的 weights 每一行代表一个模板，经过三颜色轨道重塑后可视化为下图:

图 1.6: visualization of template matching

有趣的是，horse 这一分类的模式可以看出两个马头，这说明该模板可以识别左右两个方向的马。但是使用线性模型进行模式的提取有一定局限性，通过上图可以看出，车的颜色和形状比较单一，飞机和轮船的模式图也

很抽象。之后的神经网络可以拥有更庞大的隐藏层，因而可以提取出更丰富、更细节的模式进行比对。

1.2.3 Loss function 的设计

SVM

使用多分类 SVM 损失 (Multiclass Support Vector Machine loss) 对错误的预测进行惩罚。

假设数据 x_i 的分类是 y_i。经过评分函数计算得到的分数向量是 S。我们记 s_i 为预测向量中第 i 个分类的得分。现在要对偏离正确分类的预测做出惩罚，以修正评分函数中的权重矩阵。

基本的想法是，对于 $j \neq y_i$，如果它错的太离谱，即 s_j 比 s_{y_i} 大很多，那么就要计入损失函数。这个“大很多”可以使用 $max(0, \cdot)$ 和一个超参数阈值 Δ 来衡量。于是公式可以写成：

$$\begin{aligned} L_i &= \sum_{j \neq y_i} max(0, s_j - sy_i + \Delta) \\ &= \sum_{j \neq y_i} max(0, w_j \cdot x_i - w_{y_i} \cdot x_i + \Delta) \end{aligned} \tag{1.3}$$

其中 w_k 指权重矩阵中的第 k 行.

需要注意的是，折页损失 (Hinge loss)：$max(0, \cdot)$ 可以使用平方项 (即 $max(0, \cdot)^2$) 来代替，以达到更严厉的惩罚。

除此之外，还可能出现多个 W 权重均满足 $L_i = 0, \forall i \in [1, N]$。这种模糊性使得优化问题变得复杂和不确定。因此可以在设计 Loss 函数的时候加上一个“正则项” (Regularization)，记为 $R(W)$，通常可以使用权重矩阵的 L2 范数，并用 λ 设计其权重。注意，正则项是为了权重矩阵的良好性质设计的，完全基于 W，而与数据无关。

其中最好的性质是抑制 W 中的大数值权重。比如 $[1, 0, 0, 0]$ 和 $[0.25, 0.25, 0.25, 0.25]$ 与 $[1, 1, 1, 1]$ 的内积均是 1，但是前者的 L2 范数较大，因此引起的正则化惩罚 (Regularization penalty) 就较大，在最终训练完成的 W 中就会避免这一大数值权重。

因此总的 Loss 可以写成：

$$\begin{aligned} L &= \frac{1}{N}\sum_i L_i + \lambda R(W) \\ &= \frac{1}{N}\sum_i \sum_{j\neq y_i} max(0, s_j - sy_i + \Delta) + \lambda \sum_k \sum_l W_{k,l}^2 \end{aligned} \tag{1.4}$$

看起来 Δ 和 λ 是在控制两个部分，但是 Δ 通常可以被安全地设置为 1，因为可以通过正则化强度的控制间接地影响数据评分之间的差距。这使得超参数的调整变得简单。

Softmax

SOftmax 分类是二元逻辑回归的多元版本，其直觉地将评分函数 f 给出的结果视为未归一化的对数概率。Softmax 函数

$$g_j(Z) = \frac{e^{Z_j}}{\sum_k e^{Z_k}} \tag{1.5}$$

可以将 $f(W, x_i, b)$ 这一评分向量归一化。
以交叉熵的形式定义损失函数

$$L_i = -\log(\frac{e^{Z_i}}{\sum_k e^{z_k}}) \tag{1.6}$$

总体的损失函数和 SVM 分类器相同，即

$$L = \frac{1}{N}\sum_i L_i + \lambda R(W) \tag{1.7}$$

从信息论的角度来看，用于定量真实分布 p 和估计分布 q 之间的差异的交叉熵 $H(p,q) = -\sum_x p(x)\log q(x)$，因此在损失函数中引入类似交叉熵的计算方式就是在最小化使用 Softmax 函数和线性评分函数估计出来的估计分布和真实分布(就是 N(分类种数)维向量中只有一个 1:$[0, 0, ..., 1, 0, 0, ..., 0]$)的交叉熵。

在实际操作中，经过指数变换的数值可能会很大，这可能会对精度造成影响。因此可以利用指对函数的特殊性质进行如下变换：

$$\frac{e^{f_i}}{\sum_k e^{f_k}} = \frac{C\cdot e^{f_i}}{C\cdot\sum_k e^{f_k}} = \frac{e^{f_i+\log c}}{\sum_k e^{k+\log c}} \tag{1.8}$$

通常会令 $\log C = max(f_i)$，即将 f_i 进行平移，使得最大值为 0。

对比

SVM 的计算是无标定的 (uncalibrated)，并且很难给出直观的解释。Softmax 为每种分类给出了 “可能性”，并且不会像 SVM 中有超参数 Δ，保证会有一个优化程度的上限：Softmax 永远不会对分数满意，只要有错误分类的可能性就会有继续优化的空间。