【译】cs231n第一课:图片分类、数据驱动模型，k近邻算法，训练集、验证集、测试集的划分

2016-07-13 | 104

> 本来是要学习cs224d的，但cs224d要求学几节cs231n的课程作为预备课程，故有如此翻译。
> 原文题目: 图片分类：数据驱动模型，k近邻算法，训练集、验证集、测试集的划分

这篇课件主要是为了引导那些不了解计算机视觉的人学习图片分类问题和数据驱动模式。
大致内容请见目录。

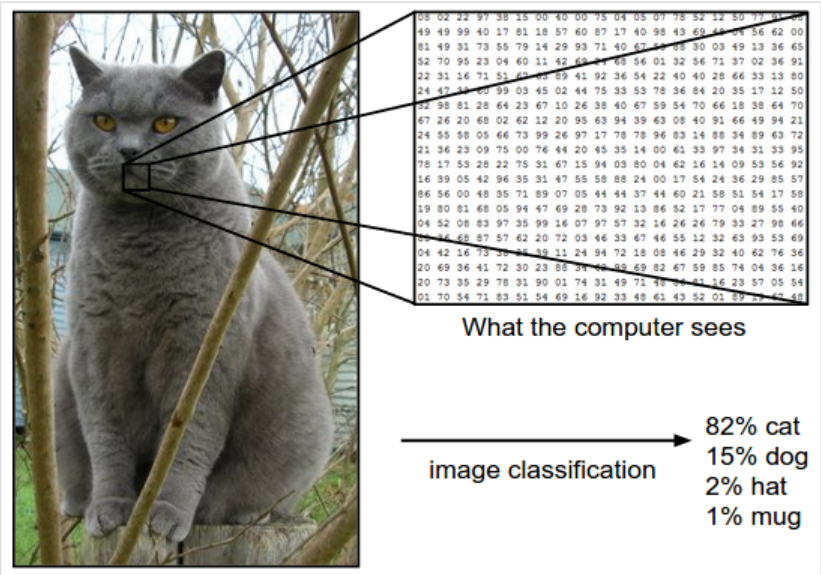
图片分类、数据驱动模式和处理流程

动机

这一小节我们主要引入图片分类问题，主要任务是对一张图片指定一个分类，而可选分类往往是固定的几个。这就是计算机视觉的核心问题，尽管表述非常简洁，但是却有着广泛的应用。此外，在该课后续的内容中，我们会发现很多不同的计算机视觉任务，比如物体识别和分割，都可以简化为图片分类问题。

举例

下面的图片分类任务是为一张图片算出四种所属类别(猫、狗、帽子、被子)的概率，正如下图所示。对于计算机来说，一个图片将会被表示为一个巨大的三维矩阵。在本例中，这只猫的图片是一个宽为248像素，高为400像素，并且拥有三个颜色通道（红、绿、蓝，简称RGB）。因此，每张图片由一个248 x 400 x 3，一共297,600个数字组成。每一个数字的范围在0~255，0代表黑色，255代表白色。我们的任务就是将这个巨大的数字转变为一个标签，如'猫'。

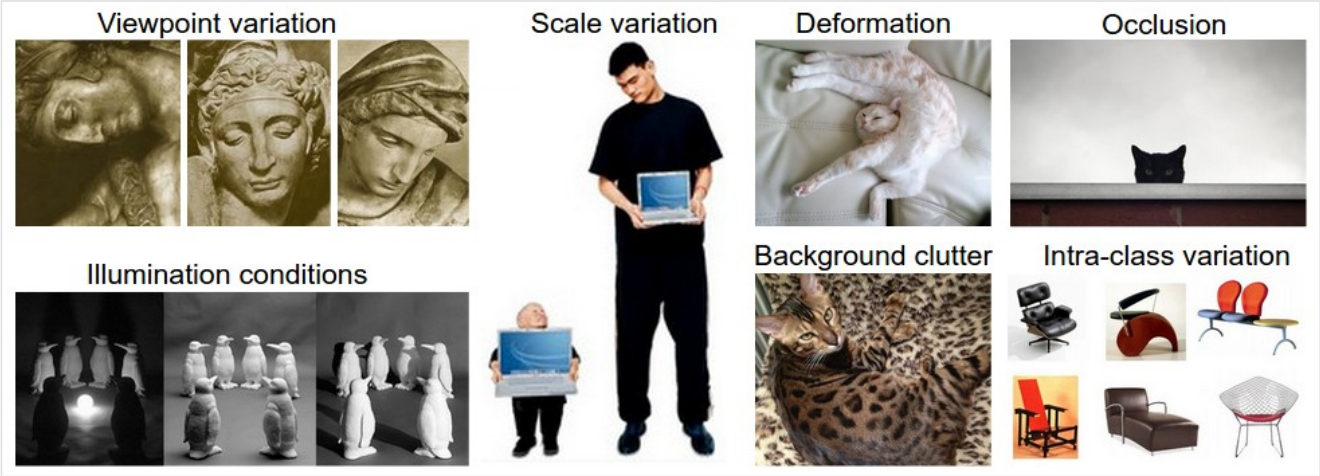


图片分类任务是为一张图片预测一个分类标签，也可以预测所有分类标签的概率分布，我们再依据其做出更多的判断。图片是由一个三维的矩阵组成，矩阵里每个数的范围都是0~255，矩阵的大小即为图片的 宽 x 高 x 3，其中3代表着红、绿、蓝三种颜色通道。

挑战

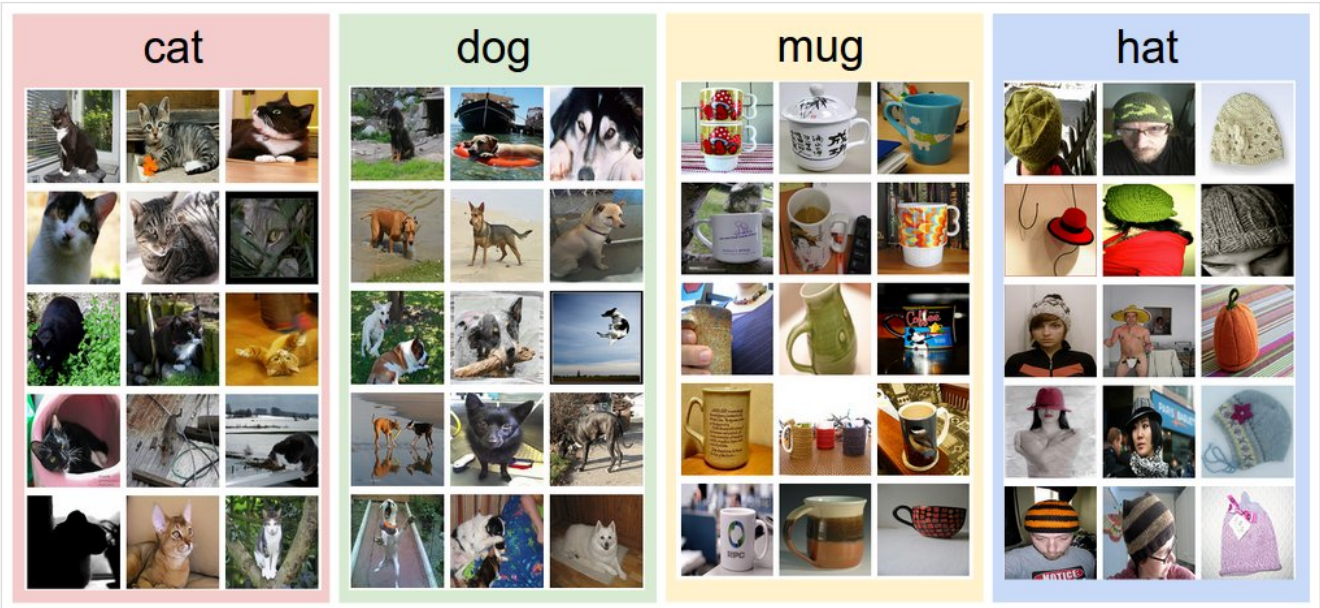
虽然识别一个视觉主体对人类来说是一个非常简单的任务，但是从计算机视觉算法来看却是一个非常具有挑战的任务。如下我们列举出了一些难点，另外注意我们是用一个三维矩阵（每个数字代表不同颜色通道的亮度）来表示图片。

- 物体方向的转变：举例来说就是我们可以转动相机来拍摄同一个物体。
- 大小的改变：同一个物体（比如远近）经常展现出大小的不同。这不仅仅只出现在图片上，现实生活中也是如此。
- 变形：许多我们感兴趣的物体不是刚体，而会发生极端的变形。
- 遮挡：很多我们感兴趣的物体会被遮挡，有时只有一小部分（小到几个像素）能被看见。
- 照明条件：在像素级别上，物体将会受到照明极大的影响。
- 杂乱的背景：物体有时候会极大的受到背景的影响，使其更难以辨认。
- 类型内的变化：一类物体通常有不同的外观（比如椅子）。



数据驱动模式

我们如何设计一个算法将图片分配到不同的种类中呢？不同于写一个排序算法，比如，我们不太清楚如何写一个算法用于识别一只猫的图片。因此，试着去写一份一份代码去识别图中每一种我们感兴趣的物体，还不如采用一种我们教小孩儿的办法：我们向电脑提供许多不同种类的例子图片，然后开发一种会自己观察这些图片，并且学习每个种类的外观特征的算法。这种方法被称为数据驱动模式，因为其依赖第一次提供给算法的带有类别标签的训练集。下图就是这样的训练集。



一个含有四种类别的训练集的例子，实际上我们有几千个类别，每个类别都有成千上万的图片。

图片分类流程

我们已经知道图片分类任务是将一个含有像素的数组作为图片，并且对这个数组标记了分类类别。我们完整的流量如下所示：

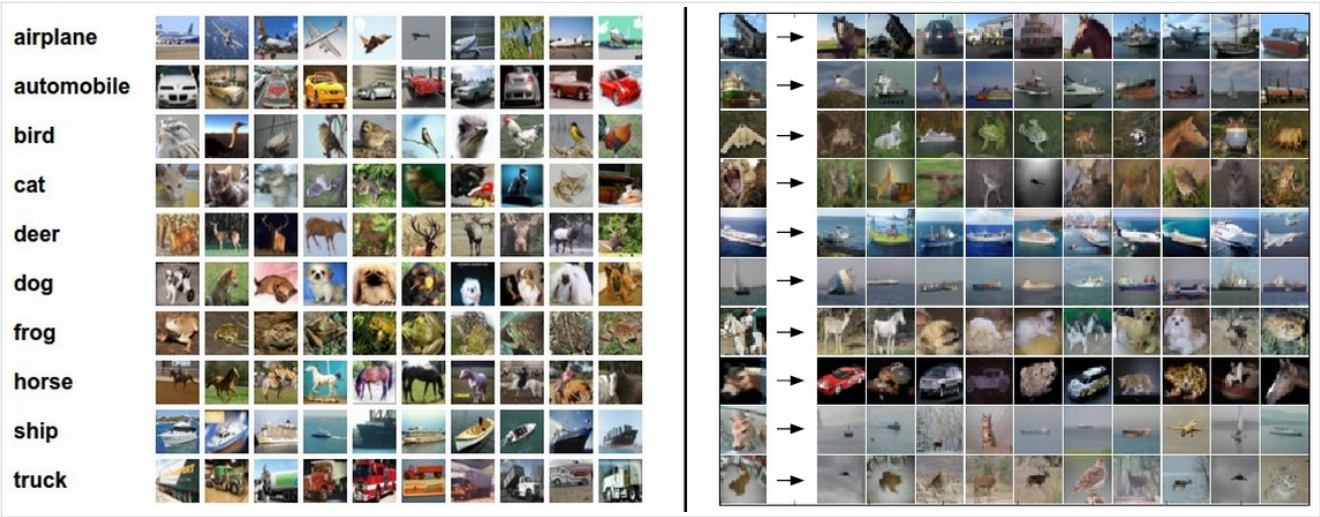
- 输入：我们的输入包含N个图片，每个图片都被标记一个分类。我们称这份数据为训练集。
- 学习：我们的任务是用这份训练集学习每个类型的图片都像什么。这个过程被称为训练一个分类器或者学习一个模型。
- 评价：最后，我们通过让分类器预测一个从未见过的图片来评价分类器的质量。然后我们会将这个图片真实的分类和预测的分类做对比。通常，我们希望更多的预测分类和图片的真是分类一致（被称为ground truth）。

最近邻分类器

作为我们的第一个方案，我们会开发一个被称为最近邻分类器。这个分类器和CNN没有任何关系，并且在实际上也没有多少使用，但是可以让我们熟悉一下处理图像分类问题的基本方法。

图片分类训练集:CIFAR-10

CIFAR-10是一个经常被使用的图片分类数据集。这份数据集含有60,000张小图片，长宽只有32个像素。每一个图片都被标记为十个分类之一，比如飞机、汽车、鸟等等。这60,000张图片被分为了含有50,000的训练集和10,000的训练集。下图中你可以看到从10个分类中随机挑选的10张图片。



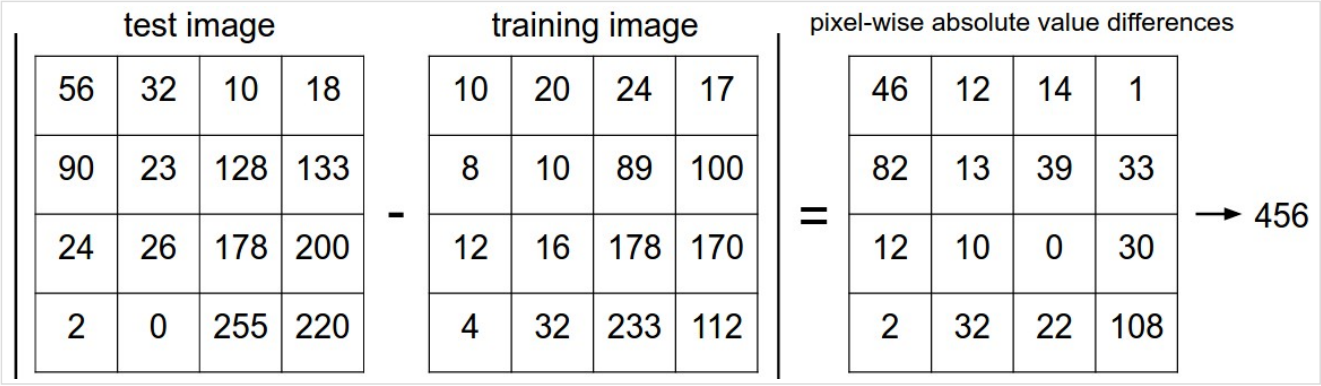
左:CIFAR-10数据集中的样例图片。右：第一列我们展示了一些测试图片，在每张图片旁边我们展现了十张来自训练集的最近邻图片(根据像素差之和来挑选)

假设现在有50,000张图片作为训练集（每一张图片都被标记了分类），然后我们希望能够标记剩下的10,000张图片的分类。最近邻分类器将需要预测的图片与训练集中的每一个图片对比，并将图片预测为与之最近的训练集中的图片的分类。在上图右侧，我们可以看到有十张图片使用这种方法来找出的训练集中的图片。然而十张里面只有三张预测成功了，其他七张都错误了。比如，第八张图片的最近邻图片是一辆车，然而需要预测的图片是马的头部，这也许是受到了杂乱的背景的影响。因此，这张实际上马的图片会被预测为一辆车。

你也许已经发现了我们还没详细说明如何比较两张图片，在本例子中就是比较两个 32 x 32 x 3 的三维数组。其中最简单的方法就是一个像素一个像素的对比，然后把所有差值加起来。也就是说，对于两张图片并且表示他们为向量 I_1，I_2，另外一个合理的选择就是用L1距离去比较两张图片。

$$d_1(I_1, I_2) = \sum_P |L_1^p - L_2^p|$$

其中加和是表示将所有的像素差给加起来。下图是展示了这个过程。



一个使用像素差再加和(L1距离)去比较两张图片的例子（这里只是显示了一个颜色通道），两个图片在像素级别上相减，然后把所有的差值加起来作为一个数。如果两个图片一样的，那么加起来的数应该是0。但是如果两个图片有非常大的区别，那么这个值将会非常大。

。首先让我们以四个数组的形式加载CIFAR-10数据进内存，这四个数组分别是：训练集、训练集的分类标记、测试集、测试集的分类标记。下面的代码中，Xtr以一个大小为50,000 x 32 x 32 x 3的数组装载了全部的训练集图片，而对应的分类标记是用一维数组 Ytr (长为50,000)装载（都是0到9的数字）。

```
Xtr, Ytr, Xte, Yte = load_CIFAR10('data/cifar10/') # 本课程的作业1提供的代码里面有这个方法
#Xtr.shape:(50000, 32, 32, 3)  Ytr.shape:(50000,) 上面四个的变量都是numpy.ndarray
Xtr_rows = Xtr.reshape((Xtr.shape[0], 32 * 32 * 3)) # 变为一维数组：50000 x 3072 参数最好带一个()，表示变形后矩阵的大小。
Xte_rows = Xte.reshape((Xte.shape[0], 32 * 32 * 3)) # 变为一维数组：10000 x 3072
```

现在我们把所有图片都变成了一行一行的，下面是我们如何训练和评价分类器：

```
nn = NearestNeighbor() # 创建一个最近邻分类器的类
nn.train(Xtr_rows, Ytr) # 用训练集训练分类器
Yte_predict = nn.predict(Xte_rows) # 对测试集上的数据预测分类
# 现在计算准确率，也就是说预测正确数所占的比例
print 'accuracy: %f' % ( np.mean(Yte_predict == Yte) )
```

注意准确率常常作为评价的常用指标，它表示正确预测的占所有预测的比例。注意对于所有的分类器都会建立统一的API：train(X,y)将以训练集和其分类标记训练分类器。而在预测器这个类里面，会用一个变量保存学习到的模型和如何去预测新的图片。在本例子中也就是predict(X) 方法，它会取新的图片并且预测其分类。当然，在作业中，会只有API，然后你们需要去实现分类器的代码。下面是用L1距离实现简单的一个最近邻分类器，并且也满足我们的模板需求。

```
import numpy as np

class NearestNeighbor(object):
  def __init__(self):
    pass

  def train(self, X, y):
    """ X 是 N x D的数组，每一行都是一个图片. Y 是 一个大小为N的一维数组"""
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X 是 N x D的数组，每一行都是一个图片. 每一行也都是我们需要预测的图片"""
    num_test = X.shape[0]
    # 预测集的输出类型和训练集的分类标记保持一致
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # 对于需要预测的图片，找到最近邻
      # 使用L1距离（是差值的绝对值之和）
```

```
        distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
        min_index = np.argmin(distances) # 获得最近邻的下标
        Ypred[i] = self.ytr[min_index] # 预测其分类为最近邻

      return Ypred

```

下面是对上面代码的一些解读，主要是两个矩阵的加减和np.sum中axis的含义：

```
In [49]: a
Out[49]:
array([[1, 2],
       [2, 3]])

In [50]: b
Out[50]: array([3, 4])

In [51]: a-b  #a的每一行和b相减
Out[51]:
array([[-2, -2],
       [-1, -1]])

In [52]: np.sum(a-b,axis=0) #axis = 0表示按列相加
Out[52]: array([-3, -3])

In [53]: np.sum(a-b,axis=1) #axis = 1表示按行相加
Out[53]: array([-4, -2])

```

如果我们运行代码，我们得到准确率为 38.6%。这只是比猜测高一点（因为有10个类型，所以猜的话准确率应该为10%），但是这份结果比人类判断的准确率低太多了（被估计为 94% ），也比现在如日中天的卷积神经网络的 95% 差远了(Kaggle比赛的记分牌有这项结果)。

选择距离

有多种方式可以计算两个向量（矩阵）的距离。另一个常见的选择就是L2距离，从图形上讲L2距离就是两个向量（矩阵）的欧几里得距离。公式如下:

$$d_2(I_1, I_2) = \sqrt{\sum_P (L_1^p - L_2^p)^2}$$

也就是说，这次我们也会算出每个像素的差值，但是这次我们会平方差值之后再求和，最后开方。在numpy中我们只用一行代码就可以计算出L2距离:

```
distances = np.sqrt(np.sum(np.square(self.Xtr - X[i,:]), axis = 1))
```

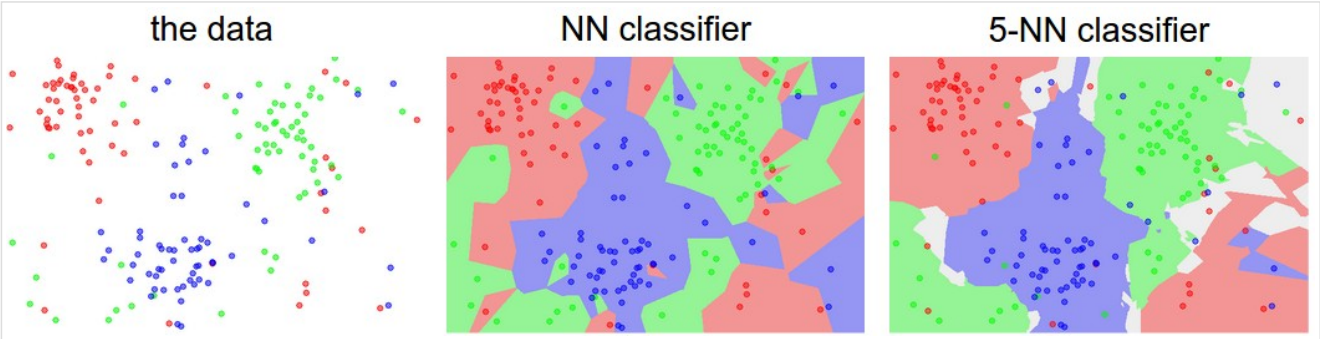
注意我加了一个np.sqrt，但是在这个最近邻分类器的应用中实际上不需要开方，因为开方是一个单调函数，不会改变数组的排序，从而并不会改变所取的最小值。如果你以L2为距离再运行上述代码，那么我们得到的准确率为35.4%（略微比L1距离低了一些）。

L1 vs L2

比较这两个指标是十分有趣和必要的。实际上，当出现差异时，L2距离比L1距离会有更高的值（对差异有更大的惩罚）。That is, the L2 distance prefers many medium disagreements to one big one。实际上 L1 距离和L2距离（包括 L1/L2 模，也可以用于比较两个图片的距离）是p-norm的特殊形式，也是最常用的形式。

k近邻分类器

可能我们已经发现了在预测的时候只参考最近的图片是不合理的。确实如此，实际上使用K近邻确实能得到更好的结果。这个想法非常简单：不是从训练集中找到与预测图片最近的图片，而且找到最近的几张图片，然后选取这几张图片中类别最多的分类作为预测图片的分类。实际上，到k=1的时候，这就是一个最近邻分类器。直觉告诉我们，越高的k值有更好的平滑作用：更能抵抗异常数据的影响。



一个例子展示了最近邻和5-邻近之间的区别，数据是二维的点，一共有三种分类（红、蓝、绿）。颜色区域展现了使用L2距离的不同分类器的分类结果。白色的区域是指无法分类的区域（发生的主要原因是选择的k个近邻的图片中不同分类的数量相等）。注意在NN分类器中，异常点（比如在蓝色区域内的绿色点）创造了一小片区域可能会导致预测不准，然而5近邻分类器却处理了这样的异常的情况，因此可以认为5近邻分类器对测试集有更好的泛化作用。最后注意在5近邻分类器中会出现一些灰色的区域，这主要是因为5个近邻图片不同分类数相等造成的（比如有2个是红色、2个是蓝色、一个是绿色，此时无法确定到底是哪个分类）

在实践中，我们经常使用K近邻分类器，但是k到底取值多少呢？我们立刻来谈谈这个问题。

用于超参数调整的验证集

超参数

因为k近邻算法需要设置k值，具体多少是合适的呢？此外，我们也看到了很多种计算距离的方式：比如L1距离和L2距离，除此之外还有很多别的距离计算方式（如点积）。这些参数都被称为超参数，在设计机器学习算法时会经常遇见，而这些参数的值（或选择）也是需要从数据中学习得来的。所以对于一个具体的问题，往往不能一下子确定是哪一个参数或者选择。

也许我们已经感受到了，尽可能的测试更多的参数和选择，然后看看哪个更好。这确实是正确的，我们也正是这么做的，但是具体操作时有些需要考虑的地方。实际上，我们不能使用测试集来完成超参数的调整。无论何时，当你设计一下机器学习算法时，都要注意测试集是非常珍贵的资源，在训练的时候往往不能使用，只能到最后一步测试预测效果时才能使用。否则，如果用测试集调整超参数，那么模型在测试集的效果非常好，但是当把模型布置到生产环境时，效果会非常差。我们称这样现象为过度拟合。从另一个角度来看这个问题，就是我们已经把测试集当做训练集在使用了。所以在测试集上的效果会远远好于生产环境中的效果。但是如果我们只能在最后预测效果时才用测试集，那么测试集相当于一种生产环境数据的替代品，让我们看到了模型泛化的能力（关于泛化，以后的课程会有更多的讨论）。

> 一个模型只能在测试集上评价一次，这种评价也是在最后一个环节。

调整超参数

当然，我们有一套成熟的方案来调整超参数，并且不会用到测试集的数据。这个办法是将训练集分割成两部分：其中有一个较小的训练集被称为验证集。使用CIFAR-10作为例子，我们可以用49,000张图片作为训练集用于训练，剩下的1,000张作为验证。其实这个验针集认为像一个测试集，只是为了调整超参数而已。

下面说具体的代码：

```
# 用前面的代码准备好了:Xtr_rows, Ytr, Xte_rows, Yte
# 将 Xtr_rows 变为 50,000 x 3072 的矩阵
Xval_rows = Xtr_rows[:1000, :] # 前1000行作为验证集
Yval = Ytr[:1000]
Xtr_rows = Xtr_rows[1000:, :] # 后49,000作为训练集
Ytr = Ytr[1000:]

# 用验证集找到表现最好的性能最好的超参数
validation_accuracies = []
for k in [1, 3, 5, 10, 20, 50, 100]:

```

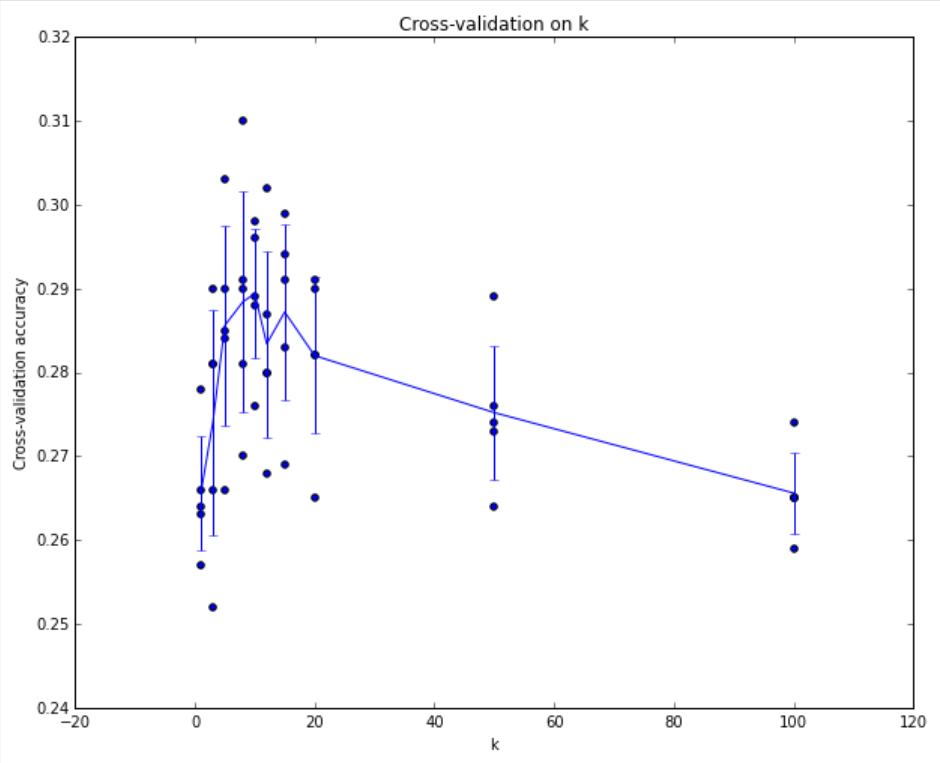
```
# 用一个k值，测试模型在验证集上的效果
nn = NearestNeighbor()
nn.train(Xtr_rows, Ytr)
# 假设修改过的k近邻能够接受k作为一个参数
Yval_predict = nn.predict(Xval_rows, k = k)
acc = np.mean(Yval_predict == Yval)
print 'accuracy: %f' % (acc,)

# 记录不同k值在测试集上的表现
validation_accuracies.append((k, acc))

```

最后，我们可以画个点图来展示所有k值的准确率，这样一下就能看出哪个k值效果最好。然后我们就选择有着最高准确率的k值，再在测试集上测试模型的最终性能。

> 将训练集分成训练集和验针集，使用验证集调整超参数。最后在测试集上运行一次得到最终效果。

交叉验证

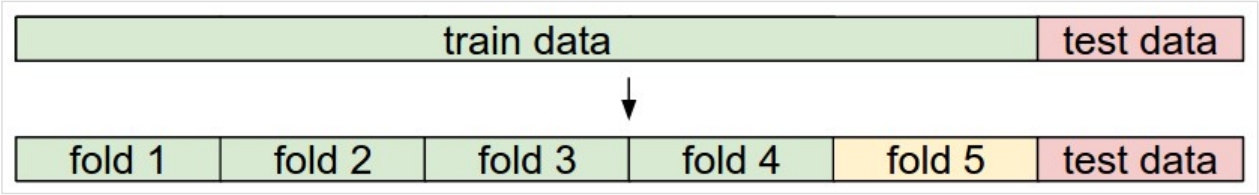
那么在分割训练集的时候分割哪一部分数据作为验证集了，有一种更为巧妙的办法解决这个问题：交叉验证。在我们先前的例子中，不是随意选择前1000个图片作为验证集，剩下的作为训练集。能减少错误评估的好办法是对于同一个k值，迭代使用不同的训练集和验证集算出性能，再求均值，这个均值就是这个k值的性能。比如说，在5-交叉验证中，我们会将训练集切分成5份，其中4份用于训练，1份用于验证。我们迭代使用每一份作为验证集，剩下的训练集，算出每一份验证集的性能，最后求均值，就得到一个k值的性能。



使用5-交叉验证调整超参数k。对于每一个k值，我们会使用4份作为训练集，剩下的一下作为验证集。于是，我们会得到5个准确率（图中准确率是Y轴，每一个点是一份结果）。趋势线是穿过了每个k值的准确率的均值，而误差棒（error bars）指名了标准差。在这个特例中，当k=7时能得到最好的性能（对应着趋势线的峰值）。如果我们把训练集切成更多份，那么我们能够得到更平滑的曲线（更少的噪点）

实践时的建议

在实践中，因为交叉验证需要耗费巨大的计算资源，所以人们通常不使用交叉验证，而使用一个验证集。一般人们会使用50%~90%的数据作为新的训练集，剩下的作为验证集。具体的数值依赖很多因素：如果超参数非常多，那么验证集通常会比较大（但只有一个，只验证一次，这样节约了计算资源和时间）；如果超参数很少（比如几百个），也可以使用交叉验证。另外，人们通常使用交叉验证时所切的份数是3、5、10。



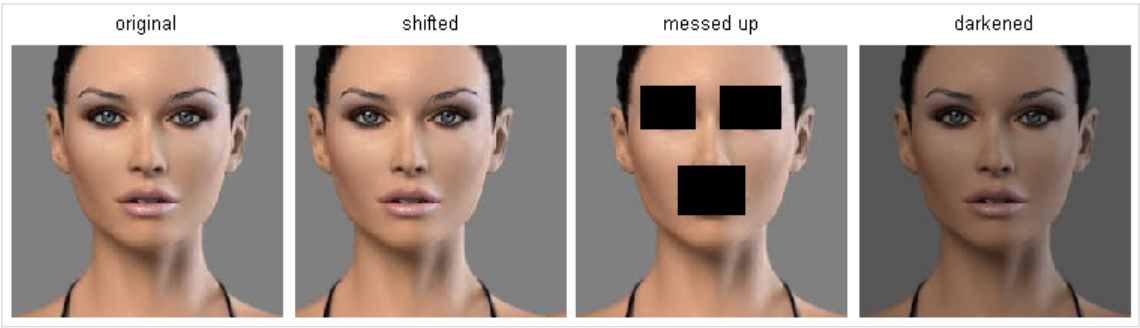
常见的数据切分。训练集和测试集如图所示。训练集将会被切分为几份（图中是5份）。第1~4份即为新的训练集，第5份（黄色）就是用于调整超参数的验证集。交叉验证是一种更为先进的方式，会迭代使用每一份作为验证集（从第一份到第五份）。这可以被称为5-交叉验证。在最后模型训练完毕后，所有的超参数已经确定了，此时模型用测试集评估一次（红色部分）以确定最后的泛化能力

最近邻算法的优缺点

近邻算法的优点是算法简单易懂、实现也比较简单。此外，近邻分类器不需要训练，因为它会保存所有的训练集的数据。然而，我们在测试性能的时候却要花费巨大的时间，因为对每一个测试样例分类都需要与训练集中每一个样本作比较。这是一个严重的缺点，因为在实践中，我们非常关心预测的时效性，而不在乎训练所需的时间。实际上，课程后续开发的深度神经网络是另一个极端：训练时需要极多的时间，但是一旦训练完成了，预测的速度非常快，这样的模型在实践中能发挥更大的效用。

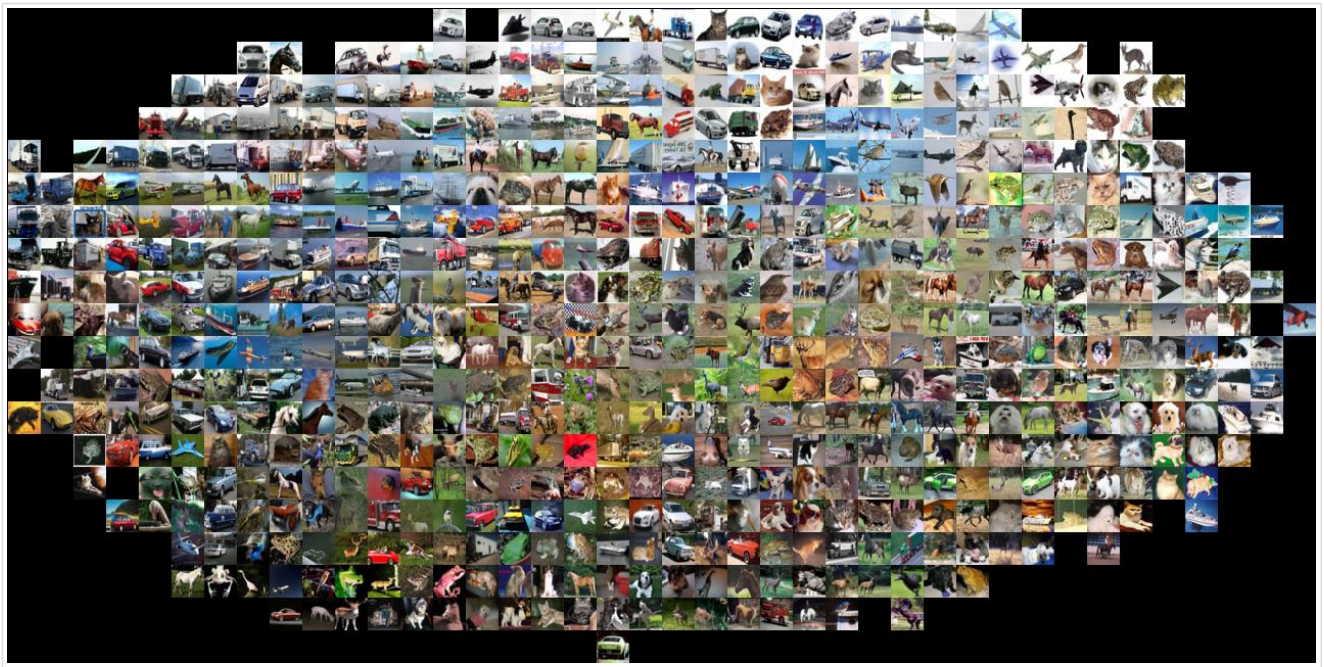
另一方面，最近邻算法的计算复杂度也是一个活跃的研究领域，一些近似近邻算法和库确定加速了该算法在训练集中的查询（例如：FLANN）。这些算法允许在计算复杂度和准确率（召回率）之间做一个权衡，这些算法的原理一般会依赖索引和预处理，比如建立kd-树，或者是k均值算法。

近邻算法也许在数据的维度比较低的时候是一个不错的算法，但是通常不会在实践中用于图片分类。主要原因是图片是高维度的数据（因为含有太多像素），并且计算出来的距离也是违反实际情况的。下图演示了这样的情况，虽然图片是相似的，但是L2距离非常大。



基于像素的距离对于高维度的数据（特别是图片）来说是违反实际情况的。最左边的原图和右边三图应该是相似的，但是L2距离却非常大。所以基于像素的距离实际不符合人类的知觉相似和语义相似。

下面是一个形象化的例子清楚表明了用基于像素的距离来比较图片是非常不现实的。我们可以使用可视化的工具（t-SNE）将CIFAR-10的图片全部嵌入一个两维的大图片中，大图片中每对小图片的距离非常容易观察出来，并且每两张图片的相对距离也是使用了基于像素算出来的距离（类似我们开发的L2距离）。



CIFAR-10的图片在t-SNE的工具帮助下被嵌入了一个两维的大图中。大图中距离近的两张小图可以认为是L2距离小的图片。注意背景对图片的影响非常大，甚至大于图片内物体种类的影响。点击(此处)[http://cs231n.github.io/assets/pixels_embed_cifar10_big.jpg]此处获得大图

此外，图片的相似也不是仅仅靠一个计算颜色分布的公式就判定的，有时背景比物体对相似的判断影响要大的多。比如当一只狗在白色的背景中时和雾也是非常像的。理想的状况下，我们会挖掘同一类图片的特点、特质，并且忽略掉无关的特质或者变量（比如背景）。然而，为了做到这一点，我们不能仅仅在像素级别研究。

总结

- 简介了图片分类，也就是在我们拥有一系列的图片和其类别的情况下，对新的图片预测分类，然后计算预测的准确率。
- 介绍了简单的分类器：最近邻分类器。并且有一些超参数（k值、比较图片的距离的选择）与k近邻分类器相关，然而我们没有很简单、明确的办法确定这些参数。
- 正确选择超参数的办法就是将训练集分成两类：新的训练集和验证集。再尝试用不同的超参数和新的训练集训练，并且选择在验证集中性能最佳的超参数组合作为模型的参数。
- 如果缺乏训练集，我们可以使用交叉验证，即有助于我们降低噪点，也会使用我们选出最佳的超参数组合。
- 一旦最佳超参数确定，可以考虑修正一下，然后在测试集上做最后的模型性能测试。
- 最近邻算法在CIFAR-10数据集上只能得到40%的准确率，它虽然实现简单，但是它要求保存全部的训练集，并且在预测一张图片的时候非常耗时。
- 最后，我们发现L1和L2距离这样基于像素的计算其实并不适用于图片分类，因为这样计算会极大地受到图片背景和颜色分布的影响，而忽略图片中具体的物体。

下一节，我们将解决一些问题，达到90%的准确率，同时允许我们在训练完成之后不再使用训练集，并在毫秒之间预测一张图片的类别。

总结: 实际应用kNN

（略）流程和上面本课内容差不多。

进一步阅读

-
- A Few Useful Things to Know about Machine Learning：虽然第六节与图片分类相关，但是都强烈推荐阅读整篇论文。
- Recognizing and Learning Object Categories： 一个对ICCV 2005的物体分类的快速课程

#cs231n

【译】cs231n第二课：线性分类 - SVM与Softmax >