

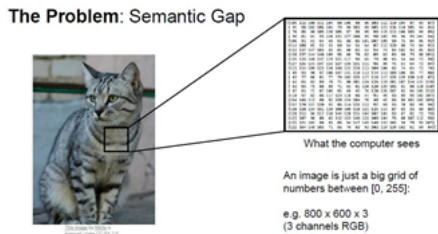
第二章 图像分类

课时1 数据驱动方法

在上一章的内容，我们提到了关于图像分类的任务，这是一个计算机视觉中真正核心的任务，同时也是本课程中关注的重点。

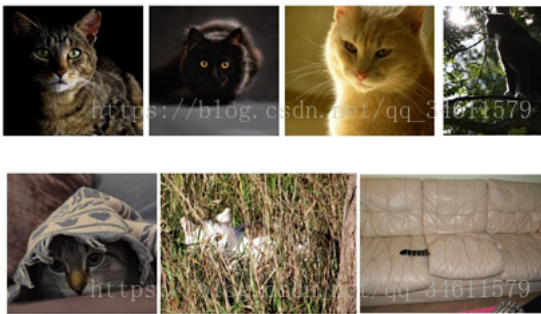
当做图像分类时，分类系统接收一些输入图像，并且系统已经清楚了一些已经确定了分类或者标签的集合，标签可能是猫、狗、汽车以及一些固定的类别标签集合等等；计算机的工作就是观察图片并且给它分配其中一些固定的分类标签。对于人来说这是非常简单的事情，但对计算机来说，却是非常困难的事情。

计算机呈现图片的方式是一大堆数字，图像可能就像下图所示的800*600的像素：



所以，对计算机来说，这就是一个巨大的数字阵列，很难从中提取出猫咪的特征，我们把这个问题称为“语义鸿沟”。对于猫咪的概念或者它的标签，是我们赋予图像的一个语义标签，而猫咪的语义标签和计算机实际看到的像素值之间有很大的差距。

对于同一只猫，从不同的角度拍出来的图像，它的数字阵列是不同，所有要同时让算法对这些变化能够做出调整；其实不仅只有角度的问题，还有光照条件等其他情况，算法都应该具有很好的鲁棒性。

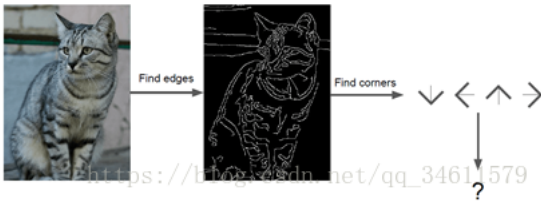


所以算法在处理这些情况的时候是非常有挑战性的，相比人的大脑能够轻易的识别出来，如果让计算机来处理这些事情都会是很难的挑战。

如果使用python写一个图像分类器，定义一个方法，接受图片作为输入参数，经过一系列的操作，最终返回到图片上进行标记是猫还是狗等等。

```
def classify_image(image):  
    # Some magic here?  
    return class_label
```

对于猫来说，它有耳朵、眼睛、鼻子、嘴巴，而通过上一章中Hubel和Wiesel的研究，我们了解到边缘对于视觉识别是十分重要的，所以尝试计算出图像的边缘，然后把边、角各种形状分类好，可以写一些规则来识别这些猫。



但是如果要想识别比如卡车、其他动物等，又需要重新从头再来一遍，所以这不是一种可推演的方法，我们需要的是一种识别算法可以拓展到识别世界上各种对象，由此我们想到了一种数据驱动的方法。

我们并不需要具体的分类规则来识别一只猫或鱼等其他的对象，取而代之的方法是：

- （1）首先收集不同类别图片的示例图，制作成带有标签的图像数据集；
- （2）然后用机器学习的方法来训练一个分类器；
- （3）最后用这个分类器来识别新的图片，看是否能够识别。

所以，如果写一个方法，可以定义两个函数，一个是训练函数，用来接收图片和标签，然后输出模型；另一个数预测函数，接收一个模型，对图片种类进行预测。



这种数据驱动类的算法是比深度学习更广义的一种理念，通过这种过程，对于一个简单的分类器（最近邻分类器），在训练过程中，我们只是单纯的记录所有的训练数据；在预测过程中，拿新的图像与已训练好的训练对比，进行预测。

我们需要知道一个细节问题：给定两幅图片，该怎么对它们进行比较？如果将测试图片和所有训练图片进行比较，将有很多不同的选择来确定需要什么样的比较函数。我们可以使用F1距离（有时称为曼哈顿距离），这是一个简单的比较图片的方法，只是对这些图片中的单个像素进行比较：

Distance Metric to compare images

L1 distance:
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

test image				training image				pixel-wise absolute value differences			
56	32	10	18	10	20	24	17	46	12	14	1
90	23	128	133	8	10	89	100	82	13	39	33
24	26	178	200	12	16	178	170	12	10	0	30
2	0	255	220	4	32	233	112	2	32	22	108

= add → 456

虽然这个方法有些笨，但是有些时候却有它的合理性，它给出了比较两幅图片的具体方法。

下面是最近邻分类器的python代码：

```
1 import numpy as np  
2  
3  
4 class NearestNeighbor:  
5     def __init__(self):  
6         pass  
7  
8     def train(self,X,y):  
9         """X是N*D，y是一维的size是N"""  
10        self.Xtr=X  
11        self.ytr=y  
12  
13    def predict(self,X):  
14        num_test=X.shape[0]  
15        y_pred=np.zeros(num_test,dtype=self.ytr.dtype)  
16  
17        for i in range(num_test):  
18            distances=np.sum(np.abs(self.Xtr-X[i,:]),axis=1)  
19            min_index=np.argmin(distances)  
20            y_pred[i]=self.ytr[min_index]  
21        return y_pred
```

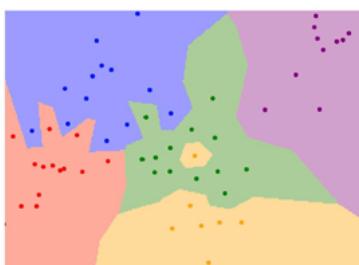
关于最近邻分类器的问题：

如果我们在训练集中有N个实例，训练和测试的过程可以有多快？

答：训练：O(1) 测试：O(N)

由此看来最近邻算法有点落后了，它在训练中花的时间很少，而在测试中花了大量时间；而看卷积神经网络和其他参数模型，则正好相反，它们会花很多时间在训练上，而在测试过程中则非常快。我们希望的是测试能够更快一点，而训练慢一点没有关系，它是在数据中心完成的。

那么在实际应用中，最近邻算法到底表现如何？可以看到下面的图像：

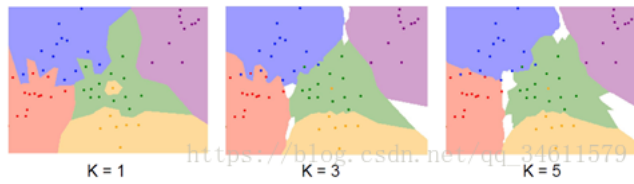


它是最近邻分类器的决策区域，训练集包含二维平面中的这些点，点的颜色代表不同的类别或不同的标签，这里有五种类型的点。对于这些点来说，将计算这些训练数据中最近的实例，然后在这些点的背景上着色，标示出它的类标签，可以发现最近邻分类器是根据相邻的点来切割空间并进行着色。

但是通过上述图片中，可以看到绿色区域中间的黄色区域、蓝色区域中有绿色区域的一部分，这些都说明了最近邻分类器的处理是有问题的。

那么，基于以上问题，产生了K-近邻算法，它不仅是寻找最近的点，还会做一些特殊的操作，根据距离度量，找到最近的K个点，然后在这些相邻点中进行投票，票数多的近邻点预测出结果。

下面用同样的数据集分别使用K=1、K=3、K=5的最近邻分类器：



在K=3时，可以看到绿色区域中的黄色点不再会导致周围的区域被划分成黄色，因为使用了多数投票，中间的这个绿色区域都会被划分成绿色；在K=5时，可以看到蓝色和红色区域间的决策边界变得更加平滑好看。

问题：上图中白色区域代表什么？白色区域表示这个区域没有获得K-最近邻的投票，可以做大胆的假设，把它划分为一个不同的类别。

所以使用最近邻分类器时，总会给K赋一个比较大的值，这会是决策边界变得更加平滑，从而得到更好的结果。

课时2 K-最近邻算法


接下来，继续讨论KNN（K-最近邻算法），回到图片中来，用红色和绿色分别标注了图像分类的正确与否：




取决于它的近邻值，可以看到KNN的表现效果不是很好，但如果可以使用一个更大的K值，那么投票操作的结果就可能会达到很好的分类效果。

当我们使用K-最近邻算法时，确定应该如何比较相对近邻数据距离值。比如，已经讨论过的L1距离，它是像素之间绝对值的总和；另一种常见的选择是L2距离，也就是欧式距离（平方和的平方根）。

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$


L2 (Euclidean) distance

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


这两种方式，L1取决于你选择的坐标系统，所以如果转动坐标轴，将会改变点之间的L1距离；而改变坐标轴对L2距离无影响。

问题：什么情况下L1距离比L2距离表现的更好？主要和解决的问题相关，很难说哪一种更好，但是如果向量中的各个元素有着实际意义，那么L1可能更加好一点。

通过使用不同的距离度量，可以将K-最近邻分类器泛化到许多不同的数据类型上，而不仅仅是向量和图像。例如，假设想对文本进行分类，那么要做的就是对KNN指定一个距离函数，这个函数可以测量两段、两句话之间的距离。

因此，通过指定不同的距离度量，就可以很好地应用这个算法在基本上任何数据类型的数据上。

可以在<http://vision.stanford.edu/teaching/cs231n-demos/knn/>尝试这个KNN，实际上是非常有趣的，可以很好地培养决策边界的直觉。

所以，一旦真的尝试在实践中使用这个算法，有几个选择是需要做的。比如，讨论过的选择K的不同值，选择不同的距离度量，该如何根据问题和数据来选择这些超参数，K值和距离度量称之为超参数，它们不一定能从训练数据中学到。

那么尝试不同的超参数看看哪种更合适该如何去做呢？

错误的策略：

（1）选择能对训练集给出最高的准确率、表现最佳的超参数；

不要这么做，在机器学习中，不是要尽可能拟合训练集，而是要让分类器在训练集以外的未知数据上表现更好。

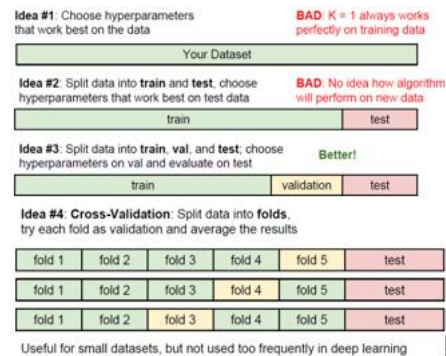
(2) 所有的数据分成两部分：一部分是训练集，另一部分是测试集，然后在训练集上使用不同的超参数来训练算法，将训练好的分类器用在测试集上，再选择一组在测试集上表现最好的超参数；

同样不要这么做，机器学习系统的目的是让我们了解算法表现究竟如何，所以测试集的目的是给我们一种预估方法，如果采用这种方法，只能让我们算法在这组测试集上表现良好，但它无法代表在未见过的数据上的表现。

正确的策略：

(3) 所有数据分成三部分：训练集、验证集和测试集，通常所做的是在训练集上用不同的超参数来训练算法，在验证集上进行评估，然后用一组超参选择在验证集上表现最好的，再把这组验证集上表现最好的分类器拿出来在测试集上运行，这才是正确的方法。

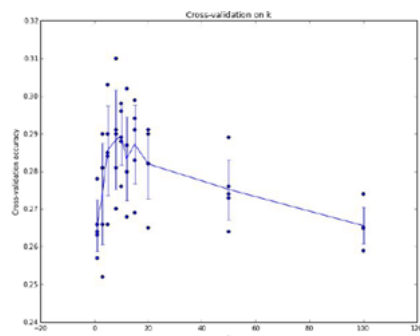
(4) 交叉验证：取出测试集数据，保留部分数据作为最后的测试集，剩余的数据不是分成训练集和验证集，而是将训练数据分成很多份，然后轮流将每一份都当成验证集。



问题：训练集和验证集的区别是什么？

算法可以看到训练集中的各个标签，但是在验证集中，只是用验证集中的标签来检查算法的表现。

那么经过交叉验证可能会得到这样的一张图：



横轴表示K-近邻分类器中的参数K值，纵轴表示分类器对不同K值在数据上的准确率。这里用了5折交叉验证，对每个K值，都对算法进行了5次不同的测试来了解这个算法表现如何；所以当训练一个机器学习的模型时，最后要画这样一张图，从中可以看出算法的表现以及各个超参数之间的关系，最终可以选出在验证集上最好的模型以及相应的超参数。

其实，KNN在图像分类中很少用到。

(1) 它的测试时间非常慢；

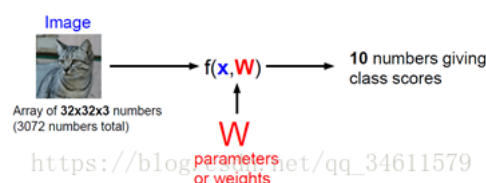
(2) 像欧式距离或者L1距离这样的衡量标准用在比较图像上不太合适；

(3) 维度灾难：KNN有点像把样本空间分成几块，意味着如果希望分类器有好的效果，需要训练数据密集的分布在空间中；而问题在于，想要密集的分布在样本空间中，需要指数倍的训练数据，然而不可能拿到这样高维空间中的像素。

总结：借用KNN介绍了图像分类的基本思路，借助训练集的图片 and 相应的标记可以预测测试集中数据的分类。

课时3 线性分类

在线性分类中，将采用与K-最近邻稍有不同方法，线性分类是参数模型中最简单的例子，以下图为例：



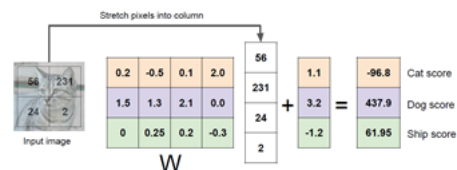
问题：图像中的3指的是什么？指的是3种颜色通道，红绿蓝。因为经常和彩色图像打交道，所以这3种通道信息就是不想丢掉的好信息。

通常把输入数据设为 x ，权重设为 w ，现在写一些函数包含了输入参数 x 和参数 w ，然后就会有10个数字描述的输出，即在CIFAR-10中对应的10个类别所对应的分数。

现在，在这个参数化的方法中，我们总结对训练数据的认知并把它都用到这些参数 w 中，在测试的时候，不再需要实际的训练数据，只需要这些参数我，这使得模型更有效率。

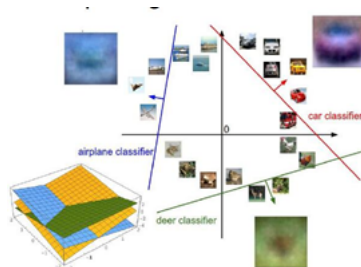
在深度学习中，整个描述都是关于函数 F 正确的结构，可以来编写不同的函数形式用不同的、复杂的方式组合权重和数据，这些对应于不同的神经网络体系结构，将他们相乘是最简单的组合方式，这就是一个线性分类器，。

线性分类器工作的例子如下：



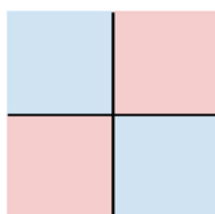
我们把2*2的图像拉伸成一个有4个元素的列向量，在这个例子中，只限制了3类：猫，狗，船；权重矩阵 w 是3行4列（4个像素3个类）；加上一个3元偏差向量，它提供了每个类别的数据独立偏差项；现在可以看到猫的分是图像像素和权重矩阵之间的输入乘积加上偏置项。

线性分类器的另一个观点是回归到图像，作为点和高维空间的概念，可以想像每一张图像都是类似高维空间中的一个点，现在线性分类器尝试在线性决策边界上画一个线性分类面来划分一个类别和剩余其他类别，如下图所示：

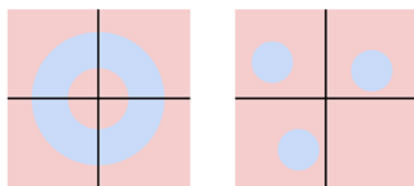


在训练过程中，这些线条会随机地开始，然后快速变化，试图将数据正确区分开，但是从这个高维的角度考虑线性分类器，就能再次看到线性分类器中出现的问题。

假设有一个两类别的数据集，蓝色和红色，蓝色类别是图像中像素数量大于0且都是奇数；红色类别是图像中像素数量大于0且都是偶数，如果去画这些不同的决策，能看到奇数像素点的蓝色类别在平面上有两个象限，所以没有办法能够绘制一条单独的直线来划分蓝色和红色，这就是线性分类器的问题所在。



当然线性分类器还有其他难以解决的情况，比如多分类问题，如下图所示：



因此，线性分类器的确存在很多问题，但它是一个非常简单的算法，易于使用和理解。

总结：本节中讨论了线性分类器对应的函数形式（矩阵向量相乘），对应于模版匹配和为每一类别学习一个单独的模板，一旦有了这个训练矩阵，可以用他得到任何新的训练样本的得分。

思考问题：如何给数据集选择一个正确的权重？这里包括损失函数和一些其他的优化方法，将在下一章中继续讨论。



想对作者说点什么