第二讲 图像分类

2019年4月20日　　12:23

1.数据驱动方法

1.1 数据驱动概念

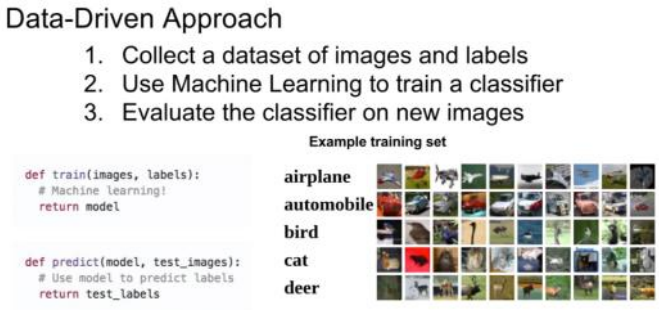

图1.1 数据驱动方法

图像分类整体思想，通过大量数据训练出一个模型，然后用这个模型做预测。

1.2 简单的图像分类方法

1.2.1 使用的数据集

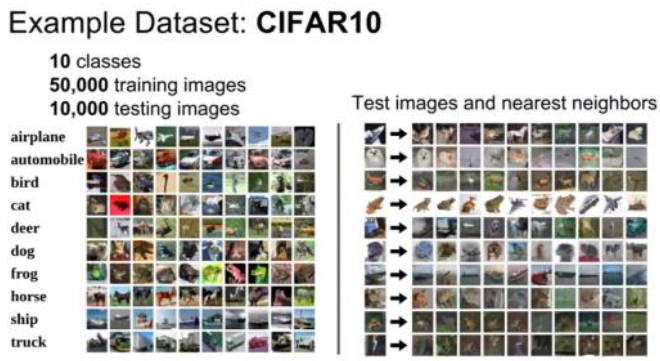

图1.2 CIFAR10数据集

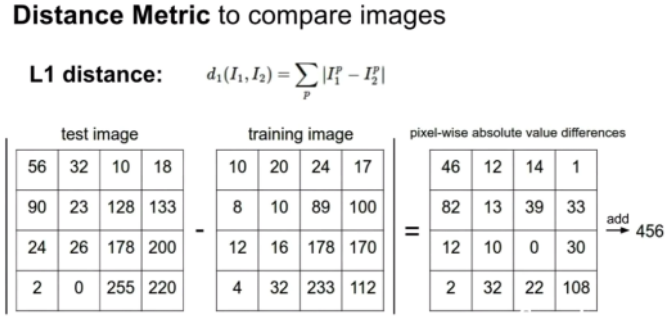

图1.3 曼哈顿距离

曼哈顿距离比较两张图片：像素单元格对应相减，求其所有单元格差的绝对值之

和。和越小证明两张图片越相似。

欧氏距离：像素单元格对应相减平方差。

在KNN中涉及到坐标轴中各个数据有向量意义时 曼哈顿距离比欧式距离好。？？

1.2.2最近邻

通过比较测试对象与已知对象的距离（曼哈顿、欧式距离等），来对该对象进行分类，这种模型叫做最近邻（Nearest Neighbor）。

1.2.3 从NN的噪声问题出发引出KNN

使用单一选取最近邻的点进行分类容易受噪声影响，选取K个点，根据已知的K个点的类别对测试对象进行分类。

2.K最近邻算法

2.1超参数概念

算法在学习之前设置的参数，并非通过训练的得到的，比如KNN中的k值、树的层数和深度、深度神经网络的隐藏层数等。

超参数优化?

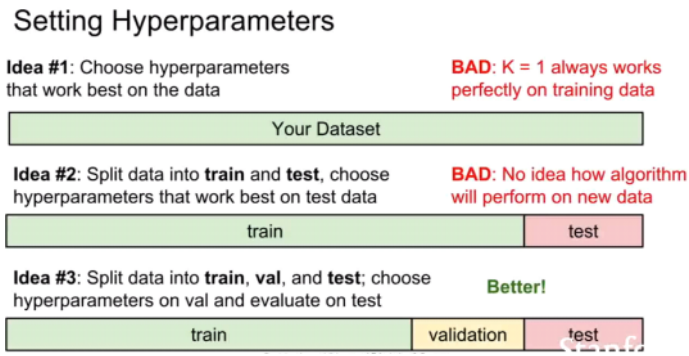

图2.1 超参数选择策略

2.2 小结

k-Nearest Neighbor on images **never used.**

- Very slow at test time
- Distance metrics on pixels are not informative

K-Nearest Neighbors: Summary

In **Image classification** we start with a **training set** of images and labels, and must predict labels on the **test set**

The **K-Nearest Neighbors** classifier predicts labels based on nearest training examples

Distance metric and K are **hyperparameters**

Choose hyperparameters using the **validation set**; only run on the test set once at the very end!

图2.2 KNN小结

3.线性分类I

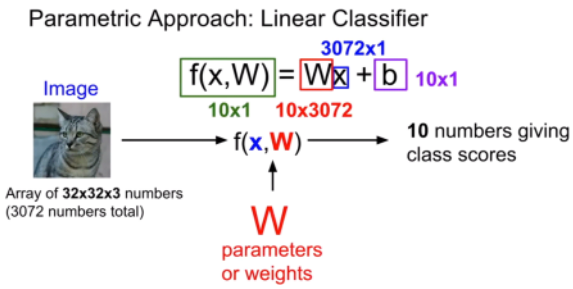

图3.1 参数矩阵行列的计算方法

3.1图片像素数已知(图中的3指rgb3通道)，预测结果分数已知，通过计算可以得到参数矩阵的行数与列数（np.shape）

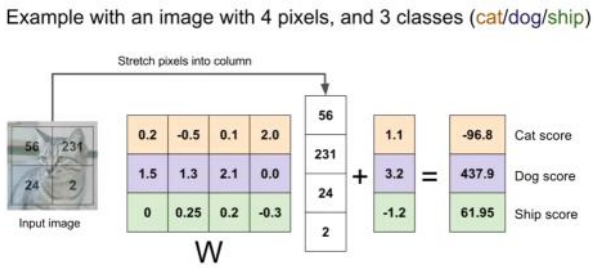

图3.2 分类器的计算方法

这是一个简单的线性分类算法应用在图像上的计算过程

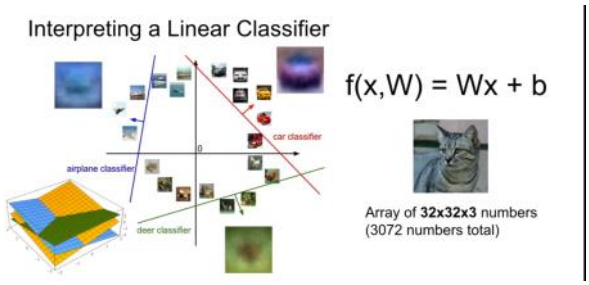

图3.3 线性分类模型

idea

机器与人脑谁的计算能力更强?

人对图片识别过程，比如一只猫，我的猜想：

1.忽略猫的颜色，首先从各个角度观察，建立一种类似于3d的空间猫模型。

2.见过不同颜色的猫，学到了猫可能有各种颜色。

3.见过猫的各种姿态，比如躺、蹲、趴等，学到了猫可能的姿态。

4.见过猫脸的样子，发现所有的猫脸型都是相似的。

5.通过前四点随机组合这些特征与新的图片比较得到是否是一只猫。

机器对图片的识别?

numpy

1.py与numpy基础

课程中给出的教程地址

http://cs231n.github.io/python-numpy-tutorial/

2.numpy常用函数

```
#根据数组产生ndarray对象
#------------------------
a=np.array([1,2,3])
b=np.array([[1,2,3],[4,5,6]])
b.shape
#------------------------


#2行2列全0矩阵
a=np.zeros((2,2))

#1行2列全1矩阵
b=np.ones((1,2))

#2行2列全7矩阵
c=np.full((2,2),7)

#2行2列单位矩阵
d=np.eye(2)

#2行2列随机填充矩阵
e=np.random.random((2,2))

# x为ndarray对象，对其内的数据就行开方操作
np.sqrt(x)

#创建一个xshape的空矩阵
np.empty_like(x)
```

#切片处理

```
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

b =  a[:2,1:3]
```

#切取 第一第二行，第二第三列的矩阵数据 [0:2,1:3]

#切片与索引混合使用

```
b = a[1,:]    #没有冒号是索引，有冒号是切片此时就变成了二维变为了一维数组
b = [5 6 7 8]       b的shape为 （4,） #1行4列 的一维数组

b =  a[1:2,:]     #此操作都是切片，数组不会被降维
b =  [[5 6 7 8]]            b的shape为 (1, 4) #1行4列的二维数组
```

#整数数组索引

```
a = np.array([[1,2], [3, 4], [5, 6]])
#[1,2]
#[3,4]
#[5,6]
a[[0, 1, 2], [0, 1, 0]]     #等价于[a[0,0],a[1,1],a[2,0]]
a[[0, 0], [1, 1]]             #reuse 等价于 a[0,1],a[0,1]
a[np.arange(4), b]          #等价于a[[0,1,2,3],b]
```

#布尔数组索引

```
a = np.array([[1,2], [3, 4], [5, 6]])
bool_idx = (a > 2)  # Find the elements of a that are bigger than 2;

            # Prints "[[False False]
 bool_idx  = #         [ True  True]
             #         [ True  True]]"

a[bool_idx] = [3 4 5 6]
```

#注 索引操作会降维

#np.array().dtype

np.array([1,2]).dtype   # dtype 数据类型 int64

np.array([1,2],dtype=np.int64).dtype   # dtype 数据类型 int64

#ndarray 的加减乘除 操作

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
```

```
np.add(x, y)       等价于  x+y
np.subtract(x, y)等价于  x-y
np.multiply(x, y)等价于  x*y
np.divide(x, y)   等价于  x/y
```

这里的*操作并不是矩阵乘法 而是元素乘法

矩阵乘法使用dot函数x.dot(y) 或者 np.dot(x,y)

矩阵转置 x.T y.T

```
np.sum(x)  #自身元素相加
np.sum(x,axis=0)  #自身列元素相加
np.sum(x,axis=1) #自身行元素相加
```

#广播

广播是一种强大的机制，允许numpy在执行算术运算时使用不同形状的数组。

常用函数np.concatenate()

```
X1.shape = x2.shape = x3.shape
X = [x1,x2,x3]
```

矩阵拼接