羽毛等级识别判断

学号：SA20218099 姓名：罗浩楠　先研院电子信息

实验目的

1. 理解机器学习如何应用到实际场景
2. 掌握特征提取方法

实验内容

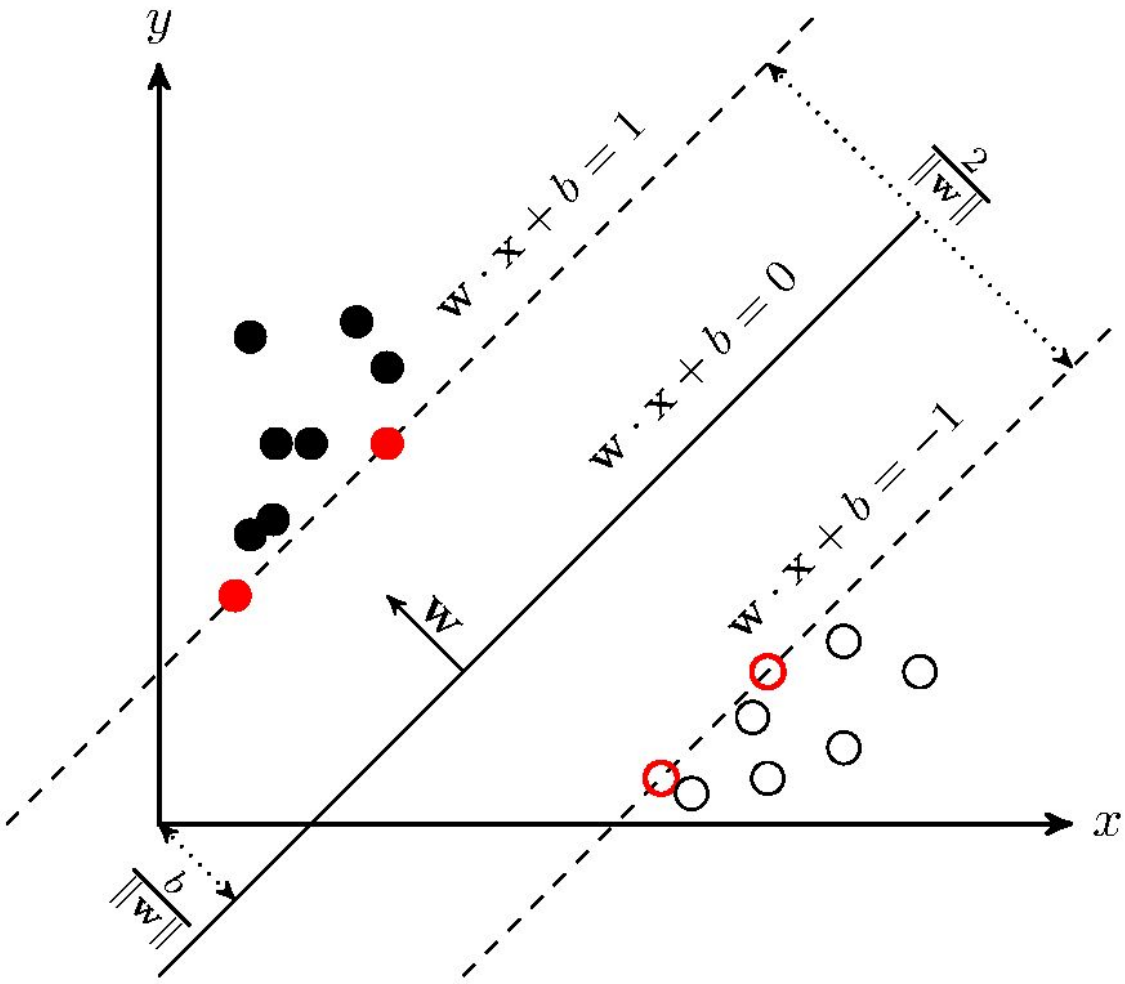
1. 数据集介绍：在制作羽毛球时，羽毛会根据质量的高低而价格不同，因此存在根据羽毛的图片而对其分级的需求。本数据集种羽毛共分为 5 个级别，分别是 1、2、3、4、56 级，其中 1 级有1353 张图片，2 级有 432 张图片，3 级有 163 张图像，4 级有 172 张图像，56 级有 42 张图像。训练集验证集按 7：3 的比例划分，划分结果在 train.txt 与 val.txt 种保存。
2. 在给出的训练图片上训练一个羽毛的等级判断模型，在验证集上进行测试，其中评价指标分为 2 个，第一个是准确率，第二个是每一级别的召回率（如 2 级羽毛共 100 根，实际测试，2 级的100 根中有 80 根识别成了 2 级，则 2 级的召回率为 80%）
3. 提示：可以采用传统计算机视觉方法提取特征，然后利用 SVM 或者决策树进行分类；也可以采用深度学习的方法来自动提取特征然后进行训练；也可以利用在 Imagnet 上训练好的模型来提取特征，然后利用 SVM 或者决策树进行分类；等等

SVM支持向量机

支持向量机（support vector machines, SVM）是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；SVM还包括核技巧，这使它成为实质上的非线性分类器。SVM的的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。SVM的的学习算法就是求解凸二次规划的最优化算法。

SVM算法原理

SVM学习的基本想法是求解能够正确划分训练数据集并且几何间隔最大的分离超平面。如下图所示，$\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b} = \mathbf{0}$即为分离超平面，对于线性可分的数据集来说，这样的超平面有无穷多个（即感知机），但是几何间隔最大的分离超平面却是唯一的。



在推导之前，先给出一些定义。假设给定一个特征空间上的训练数据集

$$T=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_N,y_N)\}$$

其中，$\boldsymbol{x_i}\in\mathbb{R}^n$，$y_i\in\{+1,-1\},i=1,2,\ldots N$，$\boldsymbol{x_i}$ 为第 $\boldsymbol{i}$ 个特征向量，y_i 为类标记，当它等于+1时为正例；为-1时为负例。再假设训练数据集是**线性可分**的。

几何间隔：对于给定的数据集 T 和超平面 $w\cdot x+b=0$，定义超平面关于样本点 (x_i,y_i) 的几何间隔为

$$\gamma_i=y_i\left(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\cdot\boldsymbol{x_i}+\frac{b}{\|\boldsymbol{w}\|}\right)$$

超平面关于所有样本点的几何间隔的最小值为

$$\gamma=\min_{i=1,2\ldots,N}\gamma_i$$

实际上这个距离就是我们所谓的**支持向量**到超平面的距离。

根据以上定义，SVM模型的求解最大分割超平面问题可以表示为以下约束最优化问题

$$\max_{\boldsymbol{w},b}\ \gamma$$

$$s.t.\quad y_i\left(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\cdot\boldsymbol{x_i}+\frac{b}{\|\boldsymbol{w}\|}\right)\geq\gamma\,,i=1,2,\ldots,N$$

将约束条件两边同时除以 γ，得到

$$y_i\left(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|\gamma}\cdot\boldsymbol{x_i}+\frac{b}{\|\boldsymbol{w}\|\gamma}\right)\geq 1$$

因为 $\|\boldsymbol{w}\|$， γ 都是标量，所以为了表达式简洁起见，令

$$\boldsymbol{w}=\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|\gamma}$$

$$b=\frac{b}{\|\boldsymbol{w}\|\gamma}$$

得到

$$y_i\left(\boldsymbol{w}\cdot\boldsymbol{x_i}+b\right)\geq 1,\ i=1,2,\ldots,N$$

又因为最大化 γ，等价于最大化 $\frac{1}{\|\boldsymbol{w}\|}$，也就等价于最小化 $\frac{1}{2}\|\boldsymbol{w}\|^2$（$\frac{1}{2}$ 是为了后面求导以后形式简洁，不影响结果），因此SVM模型的求解最大分割超平面问题又可以表示为以下约束最优化问题

$$\min_{\boldsymbol{w},b}\ \frac{1}{2}\|\boldsymbol{w}\|^2$$

$$s.t.\ \ y_i\left(\boldsymbol{w}\cdot\boldsymbol{x_i}+b\right)\geq 1,\ i=1,2,\ldots,N$$

这是一个含有不等式约束的凸二次规划问题，可以对其使用拉格朗日乘子法得到其对偶问题（dual problem）。

首先，我们将有约束的原始目标函数转换为无约束的新构造的拉格朗日目标函数

$$L\left(\boldsymbol{w},b,\boldsymbol{\alpha}\right)=\frac{1}{2}\|\boldsymbol{w}\|^2-\sum_{i=1}^{N}\alpha_i\left(y_i\left(\boldsymbol{w}\cdot\boldsymbol{x_i}+b\right)-1\right)$$

其中 $\boldsymbol{\alpha_i}$ 为拉格朗日乘子，且 $\boldsymbol{\alpha_i}\geq 0$。现在我们令

$$\theta\left(\boldsymbol{w}\right)=\max_{\alpha_i\geq 0}\ L\left(\boldsymbol{w},b,\boldsymbol{\alpha}\right)$$

当样本点不满足约束条件时，即在可行解区域外：

$$y_i\left(\boldsymbol{w}\cdot\boldsymbol{x_i}+b\right)<1$$

此时，将 $\boldsymbol{\alpha_i}$ 设置为无穷大，则 $\boldsymbol{\theta\left(w\right)}$ 也为无穷大。

当满本点满足约束条件时，即在可行解区域内：

$$y_i\left(\boldsymbol{w}\cdot\boldsymbol{x_i}+b\right)\geq 1$$

此时，$\boldsymbol{\theta\left(w\right)}$ 为原函数本身。于是，将两种情况合并起来就可以得到我们新的目标函数

$$\theta\left(\boldsymbol{w}\right)=\begin{cases}\frac{1}{2}\|\boldsymbol{w}\|^2, \boldsymbol{x}\in\text{可行区域}\\ +\infty\quad, \boldsymbol{x}\in\text{不可行区域}\end{cases}$$

于是原约束问题就等价于

$$\min_{\boldsymbol{w},b}\ \theta\left(\boldsymbol{w}\right)=\min_{\boldsymbol{w},b}\max_{\alpha_i\geq 0}\ L\left(\boldsymbol{w},b,\boldsymbol{\alpha}\right)=p^*$$

看一下我们的新目标函数，先求最大值，再求最小值。这样的话，我们首先就要面对带有需要求解的参数 $\boldsymbol{w}$ 和 $\boldsymbol{b}$ 的方程，而 $\boldsymbol{\alpha_i}$ 又是不等式约束，这个求解过程不好做。所以，我们需要使用拉格朗日函数**对偶性**，将最小和最大的位置交换一下，这样就变成了：

$$\max_{\alpha_i\geq 0}\min_{\boldsymbol{w},b}\ L\left(\boldsymbol{w},b,\boldsymbol{\alpha}\right)=d^*$$

要有 $\boldsymbol{p^*\ =\ d^*}$ ，需要满足两个条件：

① 优化问题是凸优化问题

② 满足KKT条件

首先，本优化问题显然是一个凸优化问题，所以条件一满足，而要满足条件二，即要求

$$\begin{cases}\alpha_i\ \geq 0\\ y_i\left(\boldsymbol{w_i}\cdot\boldsymbol{x_i}+b\right)-1\geq 0\\ \alpha_i\left(y_i\left(\boldsymbol{w_i}\cdot\boldsymbol{x_i}+b\right)-1\right)=0\end{cases}$$

为了得到求解对偶问题的具体形式，令 $\boldsymbol{L\left(w,b,\alpha\right)}$ 对 $\boldsymbol{w}$ 和 $\boldsymbol{b}$ 的偏导为0，可得

$$\boldsymbol{w}=\sum_{i=1}^{N}\alpha_i y_i\boldsymbol{x_i}$$

$$\sum_{i=1}^{N}\alpha_i y_i\ =0$$

将以上两个等式带入拉格朗日目标函数，消去 $\boldsymbol{w}$ 和 $\boldsymbol{b}$， 得

$$L\left(\boldsymbol{w},b,\boldsymbol{\alpha}\right)=\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\left(\boldsymbol{x_i}\cdot\boldsymbol{x_j}\right)-\sum_{i=1}^{N}\alpha_i y_i\left(\left(\sum_{j=1}^{N}\alpha_j y_j\boldsymbol{x_j}\right)\cdot\boldsymbol{x_i}+b\right)+\sum_{i=1}^{N}\alpha_i$$

$$=-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\left(\boldsymbol{x_i}\cdot\boldsymbol{x_j}\right)+\sum_{i=1}^{N}\alpha_i$$

即

$$\min_{\boldsymbol{w},b}\ L\left(\boldsymbol{w},b,\boldsymbol{\alpha}\right)=-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\left(\boldsymbol{x_i}\cdot\boldsymbol{x_j}\right)+\sum_{i=1}^{N}\alpha_i$$

求 $\min_{\boldsymbol{w},b}\ L\left(\boldsymbol{w},b,\boldsymbol{\alpha}\right)$ 对 $\boldsymbol{\alpha}$ 的极大，即是对偶问题

$$\max_{\boldsymbol{\alpha}}\ -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\left(\boldsymbol{x_i}\cdot\boldsymbol{x_j}\right)+\sum_{i=1}^{N}\alpha_i$$

$$s.t. \quad \sum_{i=1}^{N} \alpha_i y_i = 0$$

$$\alpha_i \geq 0,\ i = 1, 2, \ldots, N$$

把目标式子加一个负号，将求解极大转换为求解极小

$$\min_{\boldsymbol{\alpha}} \ \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(\boldsymbol{x_i} \cdot \boldsymbol{x_j}\right) - \sum_{i=1}^{N} \alpha_i$$

$$s.t. \quad \sum_{i=1}^{N} \alpha_i y_i = 0$$

$$\alpha_i \geq 0,\ i = 1, 2, \ldots, N$$

现在我们的优化问题变成了如上的形式。对于这个问题，我们有更高效的优化算法，即序列最小优化（SMO）算法。这里暂时不展开关于使用SMO算法求解以上优化问题的细节，下一篇文章再加以详细推导。

我们通过这个优化算法能得到 $\boldsymbol{\alpha^*}$ ，再根据 $\boldsymbol{\alpha^*}$ ，我们就可以求解出 $\boldsymbol{w}$ 和 $\boldsymbol{b}$ ，进而求得我们最初的目的：找到超平面，即"决策平面"。

前面的推导都是假设满足KKT条件下成立的，KKT条件如下

$$\begin{cases} \alpha_i \geq 0 \\ y_i \left(\boldsymbol{w_i} \cdot \boldsymbol{x_i} + b\right) - 1 \geq 0 \\ \alpha_i \left(y_i \left(\boldsymbol{w_i} \cdot \boldsymbol{x_i} + b\right) - 1\right) = 0 \end{cases}$$

另外，根据前面的推导，还有下面两个式子成立

$$\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x_i}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

由此可知在 $\boldsymbol{\alpha^*}$ 中，至少存在一个 $\alpha_j^* > 0$ （反证法可以证明，若全为0，则 $\boldsymbol{w} = 0$，矛盾），对此 $\boldsymbol{j}$ 有

$$y_j \left(\boldsymbol{w}^* \cdot \boldsymbol{x_j} + b^*\right) - 1 = 0$$

因此可以得到

$$\boldsymbol{w}^* = \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i$$

$$b^* = y_j - \sum_{i=1}^{N} \alpha_i^* y_i \left(\boldsymbol{x_i} \cdot \boldsymbol{x_j}\right)$$

对于任意训练样本 $(\boldsymbol{x_i}, y_i)$，总有 $\alpha_i = 0$ 或者 $y_j\left(\boldsymbol{w} \cdot \boldsymbol{x}_j + b\right) = 1$。若 $\alpha_i = 0$，则该样本不会在最后求解模型参数的式子中出现。若 $\alpha_i > 0$，则必有 $y_j\left(\boldsymbol{w} \cdot \boldsymbol{x_j} + b\right) = 1$，所对应的样本点位于最大间隔边界上，是一个支持向量。这显示出支持向量机的一个重要性质：**训练完成后，大部分的训练样本都不需要保留，最终模型仅与支持向量有关。**

到这里都是基于训练集数据线性可分的假设下进行的，但是实际情况下几乎不存在完全线性可分的数据，为了解决这个问题，引入了“软间隔”的概念，即允许某些点不满足约束

$$y_j\left(\boldsymbol{w} \cdot \boldsymbol{x_j} + b\right) \geq 1$$

采用hinge损失，将原优化问题改写为

$$\min_{\boldsymbol{w}, b, \xi_i} \ \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{m}\xi_i$$

$$s.t. \ \ y_i\left(\boldsymbol{w} \cdot \boldsymbol{x_i} + b\right) \geq 1 - \xi_i$$

$$\xi_i \geq 0 \ , \ i = 1, 2, \ldots, N$$

其中 $\boldsymbol{\xi_i}$ 为“松弛变量”，$\boldsymbol{\xi_i} = \max\left(0, 1 - y_i\left(\boldsymbol{w} \cdot \boldsymbol{x_i} + b\right)\right)$，即一个hinge损失函数。每一个样本都有一个对应的松弛变量，表征该样本不满足约束的程度。$C > 0$ 称为惩罚参数，C 值越大，对分类的惩罚越大。跟线性可分求解的思路一致，同样这里先用拉格朗日乘子法得到拉格朗日函数，再求其对偶问题。

综合以上讨论，我们可以得到**线性支持向量机学习算法**如下：

输入：训练数据集 $T = \left\{\left(\boldsymbol{x}_1, y_1\right), \left(\boldsymbol{x}_1, y_1\right), \ldots, \left(\boldsymbol{x}_N, y_N\right)\right\}$ 其中，$\boldsymbol{x_i} \in \mathbb{R}^n$，$y_i \in \left\{+1, -1\right\}, i = 1, 2, \ldots N$；

输出：分离超平面和分类决策函数

（1）选择惩罚参数 $C > 0$，构造并求解凸二次规划问题

$$\min_{\boldsymbol{\alpha}} \ \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\left(\boldsymbol{x_i} \cdot \boldsymbol{x_j}\right) - \sum_{i=1}^{N}\alpha_i$$

$$s.t. \quad \sum_{i=1}^{N}\alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, \ i = 1, 2, \ldots, N$$

得到最优解 $\boldsymbol{\alpha}^* = \left(\alpha_1^*, \alpha_2^*, \ldots, \alpha_N^*\right)^T$

（2）计算

$$\boldsymbol{w}^* = \sum_{i=1}^{N}\alpha_i^* y_i \boldsymbol{x_i}$$

选择 $\boldsymbol{\alpha}^*$ 的一个分量 α_j^* 满足条件 $0 < \alpha_j^* < C$，计算

$$b^* = y_j - \sum_{i=1}^{N} \alpha_i^* y_i \left(\boldsymbol{x_i} \cdot \boldsymbol{x_j} \right)$$

（3）求分离超平面

$$\boldsymbol{w}^* \cdot \boldsymbol{x} + b^* = 0$$

分类决策函数：

$$f(\boldsymbol{x}) = sign\left(\boldsymbol{w}^* \cdot \boldsymbol{x} + b^* \right)$$

非线性SVM算法原理

对于输入空间中的非线性分类问题，可以通过非线性变换将它转化为某个维特征空间中的线性分类问题，在高维特征空间中学习线性支持向量机。由于在线性支持向量机学习的对偶问题里，目标函数和分类决策函数都**只涉及实例和实例之间的内积，所以不需要显式地指定非线性变换，而是用核函数替换当中的内积**。核函数表示，通过一个非线性转换后的两个实例间的内积。具体地，$K\left(x, z \right)$ 是一个函数，或正定核，意味着存在一个从输入空间到特征空间的映射 $\phi\left(x \right)$，对任意输入空间中的 x, z，有

$$K\left(x, z \right) = \phi\left(x \right) \cdot \phi\left(z \right)$$

在线性支持向量机学习的对偶问题中，用核函数 $K\left(x, z \right)$ 替代内积，求解得到的就是非线性支持向量机

$$f\left(x \right) = sign\left(\sum_{i=1}^{N} \alpha_i^* y_i K\left(x, x_i \right) + b^* \right)$$

综合以上讨论，我们可以得到**非线性支持向量机学习算法**如下：

输入：训练数据集 $T = \left\{ \left(\boldsymbol{x}_1, y_1 \right), \left(\boldsymbol{x}_1, y_1 \right), \ldots, \left(\boldsymbol{x}_N, y_N \right) \right\}$ 其中，$\boldsymbol{x_i} \in \mathbb{R}^n$，$y_i \in \left\{ +1, -1 \right\}, i = 1, 2, \ldots N$；

输出：分离超平面和分类决策函数

（1）选取适当的核函数 $K\left(x, z \right)$ 和惩罚参数 $C > 0$，构造并求解凸二次规划问题

$$\min_{\boldsymbol{\alpha}} \ \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K\left(\boldsymbol{x_i}, \boldsymbol{x_j} \right) - \sum_{i=1}^{N} \alpha_i$$

$$s.t. \quad \sum_{i=1}^{N} \alpha_i y_i = 0$$

$$0 \le \alpha_i \le C, \ i = 1, 2, \ldots, N$$

得到最优解 $\boldsymbol{\alpha}^* = \left(\alpha_1^*, \alpha_2^*, \ldots, \alpha_N^* \right)^T$

（2）计算

选择 $\boldsymbol{\alpha}^*$ 的一个分量 α_j^* 满足条件 $0 < \alpha_j^* < C$，计算

$$b^* = y_j - \sum_{i=1}^{N} \alpha_i^* y_i K\left(\boldsymbol{x_i}, \boldsymbol{x_j}\right)$$

（3）分类决策函数：

$$f(x) = sign\left(\sum_{i=1}^{N} \alpha_i^* y_i K\left(x, x_i\right) + b^*\right)$$

介绍一个常用的核函数——**高斯核函数**

$$K(x, z) = \exp\left(-\frac{\|x - z\|^2}{2\sigma^2}\right)$$

对应的SVM是高斯径向基函数分类器，在此情况下，分类决策函数为

$$f(x) = sign\left(\sum_{i=1}^{N} \alpha_i^* y_i \exp\left(-\frac{\|x - z\|^2}{2\sigma^2}\right) + b^*\right)$$

HOG

Histogram of Oriented Gridients，缩写为HOG，是目前计算机视觉、模式识别领域很常用的一种描述图像局部纹理的特征。

分割图像

因为HOG是一个局部特征，因此如果你对一大幅图片直接提取特征，是得不到好的效果的。原理很简单。从信息论角度讲，例如一幅640×480的图像，大概有30万个像素点，也就是说原始数据有30万维特征，如果直接做HOG的话，就算按照360度，分成360个bin，也没有表示这么大一幅图像的能力。从特征工程的角度看，一般来说，只有图像区域比较小的情况，基于统计原理的直方图对于该区域才有表达能力，如果图像区域比较大，那么两个完全不同的图像的HOG特征，也可能很相似。但是如果区域较小，这种可能性就很小。最后，把图像分割成很多区块，然后对每个区块计算HOG特征，这也包含了几何（位置）特性。例如，正面的人脸，左上部分的图像区块提取的HOG特征一般是和眼睛的HOG特征符合的。

接下来说HOG的图像分割策略，一般来说有overlap和non-overlap两种，如下图所示。overlap指的是分割出的区块（patch）互相交叠，有重合的区域。non-overlap指的是区块不交叠，没有重合的区域。这两种策略各有各的好处。

non-overlap

overlap

先说overlap，这种分割方式可以防止对一些物体的切割，还是以眼睛为例，如果分割的时候正好把眼睛从中间切割并且分到了两个patch中，提取完HOG特征之后，这会影响接下来的分类效果，但是如果两个patch之间overlap，那么至少在一个patch会有完整的眼睛。overlap的缺点是计算量大，因为重叠区域的像素需要重复计算。

再说non-overlap，缺点就是上面提到的，有时会将一个连续的物体切割开，得到不太"好"的HOG特征，优点是计算量小，尤其是与Pyramid（金字塔）结合时，这个优点更为明显。

计算每个区块的方向梯度直方图

将图像分割后，接下来就要计算每个patch的方向梯度直方图。步骤如下：

A.利用任意一种梯度算子，例如：sobel，laplacian等，对该patch进行卷积，计算得到每个像素点处的梯度方向和幅值。具体公式如下：

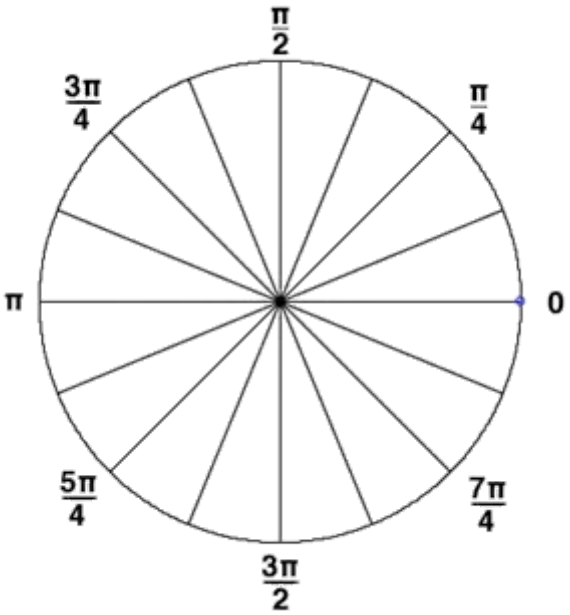
$$M(x,y) = \sqrt{I_x^2 + I_y^2} \qquad (1)$$

$$\theta(x,y) = tan^{-1}\frac{I_y}{I_x} \in [0,360^\circ) or \in [0,180^\circ) \qquad (2)$$

Paste_Image.png

其中，Ix和Iy代表水平和垂直方向上的梯度值，M(x,y)代表梯度的幅度值，θ(x,y)代表梯度的方向。

B.将360度（2×PI）根据需要分割成若干个bin，例如：分割成12个bin，每个bin包含30度，整个直方图包含12维，即12个bin。然后根据每个像素点的梯度方向，利用双线性内插法将其幅值累加到直方图中。



C.（可选）将图像分割成更大的Block，并利用该Block对其中的每个小patch进行颜色、亮度的归一化，这一步主要是用来去掉光照、阴影等影响的，对于光照影响不剧烈的图像，例如很小区域内的字母，数字图像，可以不做这一步。而且论文中也提及了，这一步的对于最终分类准确率的影响也不大。

组成特征

将从每个patch中提取出的"小"HOG特征首尾相连，组合成一个大的一维向量，这就是最终的图像特征。可以将这个特征送到分类器中训练了。例如：有*44=16个patch，每个patch提取12维的小HOG，那么最终特征的长度就是：16*12=192维。

准确率与召回率

混淆矩阵

True Positive(真正，TP)：将正类预测为正类数

True Negative(真负，TN)：将负类预测为负类数

False Positive(假正，FP)：将负类预测为正类数误报 (Type I error)

False Negative(假负，FN)：将正类预测为负类数→漏报 (Type II error)

	Positive	Negative
True	True Positive (TP)	True Negative (TN)
False	False Positive (FP)	False Negative (FN)

		预测类别		
		Yes	No	总计
实际类别	Yes	TP	FN	P（实际为Yes）
	No	FP	TN	N（实际为No）
	总计	P'（被分为Yes）	N'（被分为No）	P+N

准确率（Accuracy）

准确率(accuracy)计算公式为：

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

召回率（recall）

召回率是覆盖面的度量，度量有多个正例被分为正例，recall=TP/(TP+FN)=TP/P=sensitive，可以看到召回率与灵敏度是一样的。

实验代码

由于图片格式不确定，首先对所有图片操作得到同一大小的图片，之后对train.txt和val.txt进行处理并将图片放入同一个文件夹，之后根据train.txt和val.txt将图片分成train和test两个文件夹并对其中的图片进行特征化，之后使用支持向量机对其进行训练并分类。

```
#coding=utf-8
from PIL import Image
import os
from tqdm import tqdm
import time
import glob
import platform
from skimage.feature import hog
import numpy as np
import joblib
from sklearn.svm import LinearSVC
import shutil


def fixed_size(filePath,savePath):
    """
    按照固定尺寸处理图片
    """
    im = Image.open(filePath)
    out = im.resize((image_width, image_height), Image.ANTIALIAS)
    out.save(savePath)


def changeSize():
    filePath = './featherdataset'
```

```
    destPath = './featherdataset/imgSizeChanged'
    #for eachfilePath
    print("正在进行图像初始化...")
    if not os.path.exists(destPath):
        os.makedirs(destPath)
    for root, dirs, files in os.walk(filePath):
        for file in files:
            if file[-1]=='g':
                fixed_size(os.path.join(filePath, file), os.path.join(destPath,
file))
    print('图像初始化已完成')

def datasetmade(target_path,filenames):
    for file in filenames:
        # 所有的子文件夹
        sonDir = "featherdataset/" + file
        print("正在对{}数据集进行初始化".format(file))
        time.sleep(0.5)
        # 遍历子文件夹中所有的文件
        for root, dirs, files in os.walk(sonDir):
            #如果文件夹中有文件
            if len(files) > 0:
                for f in tqdm(files):
                    newDir = sonDir + '/' + f
                    # 将文件移动到新文件夹中
                    im = Image.open(newDir)
                    out = im.resize((image_width, image_height),
Image.ANTIALIAS)
                    # out = out.crop((0,0,600,900))
                    # out.show()
                    if not os.path.exists(target_path):
                        os.mkdir(target_path)
                        # print("数据集合并文件夹新建完成")
                    out.save(target_path+'/'+f)
                    # 将文件改名改成训练集中带\1格式
                    # print(target_path+'/'+f)
                    # print(target_path + '/' +file+'-'+ f)
                    os.rename(target_path+'/'+f,target_path + '/' +file+'-'+ f)

                    time.sleep(0.01)
            else:
                print(sonDir + "文件夹是空的")
        time.sleep(1)


def txtProcess():
    # 对train.txt和val.txt操作 使其与newdataset中的文件名相同
    txtChange = ["featherdataset/train.txt","featherdataset/val.txt"]
    for eachtxtChange in txtChange:
        with open(eachtxtChange,'r+') as f:
            print("---------正在对{}进行初始化---------
".format(eachtxtChange.split('/')[-1]))
            time.sleep(0.5)
            newf = open("featherdataset/new"+eachtxtChange.split('/')[-1] ,'w')
            Alllines = f.readlines()
            for eachstringline in tqdm(Alllines):
                eachstringline = eachstringline.replace('/', '-')
                newf.write(eachstringline)
```

```
                time.sleep(0.005)
            newf.close()

def makedirs():
    os.mkdir("train")
    os.mkdir("test")

# 获得图片列表
def get_image_list(filePath, nameList):
    print('正在进行图像读取... ',filePath)
    img_list = []
    for name in nameList:
        temp = Image.open(os.path.join(filePath,name))
        img_list.append(temp.copy())
        temp.close()
    return img_list

# 提取特征并保存
def get_feat(image_list, name_list, label_list, savePath):
    i = 0
    for image in image_list:
        try:
            # 如果是灰度图片  把3改为-1
            image = np.reshape(image, (image_height, image_width, 3))
        except:
            print('发送了异常，图片大小size不满足要求：',name_list[i])
            continue
        gray = rgb2gray(image) / 255.0
        fd = hog(gray, orientations=12,block_norm='L1', pixels_per_cell=[13,
13],
                 cells_per_block=[4, 4], visualize=False,transform_sqrt=True)
        fd = np.concatenate((fd, [label_list[i]]))
        fd_name = name_list[i] + '.feat'
        fd_path = os.path.join(savePath, fd_name)
        joblib.dump(fd, fd_path)
        i += 1
    print("features are extracted and saved.")


# 变成灰度图片
def rgb2gray(im):
    gray = im[:, :, 0] * 0.2989 + im[:, :, 1] * 0.5870 + im[:, :, 2] * 0.1140
    return gray


# 获得图片名称与对应的类别
def get_name_label(file_path):
    print("正在从文件载入...",file_path)
    name_list = []
    label_list = []
    with open(file_path) as f:
        for line in f.readlines():
            #一般是name label  三部分，所以至少长度为3  所以可以通过这个忽略空白行
            if len(line)>=3:
                name_list.append(line.split(' ')[0])
                label_list.append(line.split(' ')
[1].replace('\n','').replace('\r',''))
                if not str(label_list[-1]).isdigit():
```

```
                    print("label必须为数字，得到的是：",label_list[-1],"程序终止，请检查文件")
                    exit(1)
    return name_list, label_list


# 提取特征
def extra_feat():
    train_name, train_label = get_name_label(train_label_path)
    test_name, test_label = get_name_label(test_label_path)

    train_image = get_image_list(train_image_path, train_name)
    test_image = get_image_list(test_image_path, test_name)
    get_feat(train_image, train_name, train_label, train_feat_path)
    get_feat(test_image, test_name, test_label, test_feat_path)


# 创建存放特征的文件夹
def mkdir():
    """

    Returns
    -------
    None.

    """
    if not os.path.exists(train_feat_path):
        os.mkdir(train_feat_path)
    if not os.path.exists(test_feat_path):
        os.mkdir(test_feat_path)


# 训练和测试
def train_and_test():
    features = []
    labels = []
    correct_number = 0
    total = 0
    for feat_path in glob.glob(os.path.join(train_feat_path, '*.feat')):
        data = joblib.load(feat_path)
        features.append(data[:-1])
        labels.append(data[-1])
    print("SVM Training......")
    clf = LinearSVC()
    clf.fit(features, labels)
    # 下面的代码是保存模型的
    if not os.path.exists(model_path):
        os.makedirs(model_path)
    joblib.dump(clf, model_path + 'model')
    # 下面的代码是加载模型  可以注释上面的代码   直接进行加载模型  不进行训练
    # clf = joblib.load(model_path+'model')
    print("Model has saved......")
    # exit()
    result_list = []
    for feat_path in glob.glob(os.path.join(test_feat_path, '*.feat')):
        total += 1
        if platform.system() == 'Windows':
            symbol = '\\'
```

```
        else:
            symbol = '/'
        image_name = feat_path.split(symbol)[1].split('.feat')[0]
        data_test = joblib.load(feat_path)
        data_test_feat = data_test[:-1].reshape((1, -1)).astype(np.float64)
        result = clf.predict(data_test_feat)
        result_list.append(image_name + ' ' + label_map[int(result[0])] + '\n')
        if int(result[0]) == int(data_test[-1]):
            correct_number += 1
    write_to_txt(result_list)
    rate = float(correct_number) / total
    print('准确率为： %f' % rate)


def write_to_txt(list):
    with open('result.txt', 'w') as f:
        f.writelines(list)
    print('Result has been saved in result.txt')


def trainandtestdatasetmade():
    txtlist = ["featherdataset/newtrain.txt","featherdataset/newval.txt"]
    for eachtxtlist in txtlist:
        with open(eachtxtlist,"r") as f:
            allLines = f.readlines()
            load = "featherdataset_"+eachtxtlist.split('/')[-1].split('.')[0]
            os.mkdir(load)
            print("正在创建{} 集".format(eachtxtlist.split('new')[-1]).split('.')
[0])
            time.sleep(0.5)
            for eachline in tqdm(allLines):
                fileload =
seek_files('featherdataset/newdataset',eachline.split(' ')[0])
                shutil.copy(fileload,load)
                time.sleep(0.05)
            shutil.copy(eachtxtlist,load)


def seek_files(id1, name):
    """根据输入的文件名称查找对应的文件夹有无改文件，有则输出文件地址"""
    for root, dirs, files in os.walk(id1):
        if name in files:
            # 当层文件内有该文件，输出文件地址
            return root+'/'+name

if __name__ == '__main__':

    t0 = time.time()
    makedirs()

    label_map = {1: '1',
             2: '2',
             3: '3',
             4: '4',
             56:'56'
             }

    train_feat_path = 'train/'
```

```
test_feat_path = 'test/'
model_path = 'model/'


# 设定图片的统一尺寸
image_width = 128
image_height = 100

targetpath =r"featherdataset/newdataset"
# 原文件夹
old_path = r"featherdataset"
# 查看原文件夹下所有的子文件夹
filenames = os.listdir(old_path)

datasetmade(targetpath,filenames=filenames)
print("数据集生成完成")

txtProcess()
trainandtestdatasetmade()


# 训练集图片的位置
train_image_path = 'featherdataset_newtrain'
# 测试集图片的位置
test_image_path = 'featherdataset_newval'

# 训练集标签的位置
train_label_path = train_image_path+'/newtrain.txt'
# 测试集标签的位置
test_label_path = test_image_path+'/newval.txt'


shutil.rmtree(train_feat_path)
shutil.rmtree(test_feat_path)
mkdir()
extra_feat()  # 获取特征并保存在文件夹
print("特征提取成功")

train_and_test()

t1 = time.time()
print('耗时: %f' % (t1 - t0))
```

输出

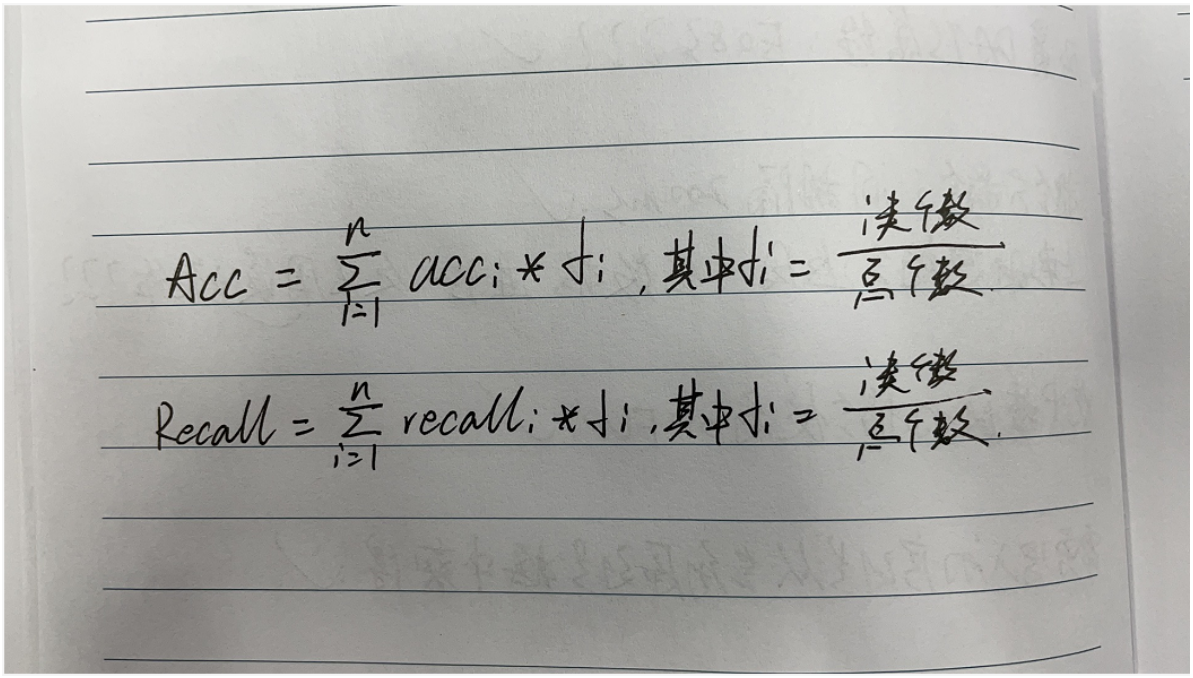
```
In [51]: runfile('F:/temp/ailab2.py', wdir='F:/temp')
正在对1数据集进行初始化
100%|██████████| 1353/1353 [00:33<00:00, 40.11it/s]
正在对2数据集进行初始化
100%|██████████| 432/432 [00:10<00:00, 40.24it/s]
正在对3数据集进行初始化
100%|██████████| 163/163 [00:04<00:00, 39.58it/s]
正在对4数据集进行初始化
100%|██████████| 172/172 [00:04<00:00, 40.08it/s]
正在对56数据集进行初始化
100%|██████████| 42/42 [00:01<00:00, 40.15it/s]
正在对train.txt数据集进行初始化
正在对val.txt数据集进行初始化
数据集生成完成
---------正在对train.txt进行初始化---------
100%|██████████| 1295/1295 [00:07<00:00, 181.27it/s]
---------正在对val.txt进行初始化---------
100%|██████████| 867/867 [00:04<00:00, 182.46it/s]
正在创建train
100%|██████████| 1295/1295 [01:10<00:00, 18.50it/s]
正在创建val
100%|██████████| 867/867 [00:46<00:00, 18.55it/s]
正在从文件载入... featherdataset_newtrain/newtrain.txt
正在从文件载入... featherdataset_newval/newval.txt
正在进行图像读取...  featherdataset_newtrain
正在进行图像读取...  featherdataset_newval
features are extracted and saved.
features are extracted and saved.
特征提取成功
SVM Training......
Model has saved......
Result has been saved in result.txt
混淆矩阵为:
[[536.   5.   0.   1.   0.]
 [160.  12.   0.   1.   0.]
 [ 57.   9.   0.   0.   0.]
 [ 56.   9.   0.   4.   0.]
 [ 15.   1.   0.   1.   0.]]
准确率为: 0.6366782006920415
召回率为 0.5186370759992556
耗时: 221.783990
```

Note

多分类下的准确率和召回率：

多分类准确率和召回率		预测				
		A	B	C	D	E
实际	A	TP1	FP1	FP2	FP3	FP4
	B	FP5	TP	FP	FP	FP
	C	FP6	FP	TP	FP	FP
	D	FP7	FP	FP	TP	FP
	E	FP8	FP	FP	FP	TP
A类准确率 = TP1/（TP1+FP1+FP2+FP3+FP4）						
A类召回率 = TP1/（TP1+FP5+FP6+FP7+FP8）						

基于此上述求准确率和召回率的公式如下：



参考

Navneet Dalal and Bill Triggs，《Histograms of Oriented Gradients for Human Detection》，2005
A. Bosch, A. Zisserman, and X. Munoz,《Representing shape with a spatial pyramid kernel》，2007

《统计学习方法》 李航

《机器学习》周志华