线性分类器实验报告

1. 实验数据：

行人检测数据库:包含正性（pos (*).png）与负性（neg (*).png）样本分别 1000 张，其中：
正例的图片尺寸：48*96
负例的图片尺寸：64*128

2. 实验环境：

操作系统 ubuntu 14.04
语言 python 2.7.6
依赖包：
矩阵运算：numpy 版本 1.8.1(sudo apt-get install python-numpy)
图像操作：PIL 包(python image library) imagemagick 工具包中包含了这个包，实验中直接安装了 imagemagick(sudo apt-get install imagemagick)
数据可视化：matplotlib(sudo apt-get install python-matplotlib)

3. 代码：

getvec_LBP.py：
从图像数据中提取特征向量，按正例和负例分别输出到两个文件(feature_vects.pos feature_vects.neg)

train_byLDA.py :
模型训练，以(feature_vects.pos feature_vects.neg)为输入获取模型参数，在配置文件 conf 中有一个整数 n（1~999）表示划分给训练集的实例个数（正例与负例相同），本脚本会读取配置文件中的 n，然后读取 feature_vects.*文件中的前 n 行作为训练集。得到的模型参数输出到文件 model(LDA 算法的投影向量)和 limit_point(LDA 算法的分类阀值)中。

test_model.py :
模型测试，读取文件 model 和 limit_point 中的模型参数以及特征向量文件 feature_vects.*(从配置文件 conf 中获取训练集个数 n，从 n+1 行开始读取剩余的数据作为测试集)，最后测试集的识别率直接输出到屏幕。

plot.py :
数据可视化，把测试数据在模型投影方向上的投影值用散点图的形式展示出来

运行顺序：getvec_LBP.py >
(可修改 conf 文件)train_byLDA.py > test_model.py >plot.py

4. 实验过程：

4.1 从样本提取特征（getvec_LBP.py）

采用方法：LBP（参考资料：作业附件的 LBP.docx 和 http://blog.csdn.net/zouxy09/article/details/7929531）

LBP 算子的窗口：3*3；
LBP 的旋转不变性：在一个窗口中可能产生 256 个 LBP 的模式，利用旋转不变性可以计算得到对应 36 个最小化的值（见函数 get_basiclbplist）；

LBP 图谱转化成向量：参考链接中的统计的方法，需要对图像划分为几个区域，分别统计各个区域中 LBP 的 36 个基准值出现的频率。正例与负例的图像大小分别为 48*96 和 64*128，一个合适的划分是分成 4*8 个区域，其中单个区域的尺寸：
正例：12*12
负例：16*16
每个区域的统计信息有 36 维（每一维代表一个 LBP 值出现的频率），因此特征向量一共有 36 ×4×8 = 1152 维

其他：用到 PIL 包获取图像灰度值

4.2 模型训练（train_byLDA.py）

采用方法：LDA（参考资料：教材 4.3、prml 4.1.4）推倒过程略，直接给出计算过程：
先计算投影向量方向，公式如下：

$$w^* = S_w^{-1}(m_1 - m_2)$$

分别利用训练集计算正例与负例特征向量的向量均值 m1、m2 以及协方差矩阵 S1 与 S2：

$$\mathrm{m}_i = \frac{1}{N}\sum_{x_j \in \chi_i} x_j$$

$$\mathrm{S}_i = \sum_{x_j \in \chi_i}(x_j - m_i)(x_j - m_i)^T$$

总的类内协方差矩阵：

$$\mathrm{S}_w = S_1 + S_2$$

计算得到投影向量 w 之后还要求得一个分类的阈值，我简单采用了教材上给出的如下求解方法：

$$w_0 = -\frac{1}{2}(w^T m_1 - w^T m_2)$$

最后采取决策的规则是：

$$若 g(x) = w^T x + w_0 \begin{matrix} > \\ < \end{matrix} 0 则 x \in \begin{cases} \varpi_1 \\ \varpi_2 \end{cases}$$

在训练过程发现，当训练集的变化较大时，正例集与负例集的投影的相对位置可能会变化，因此，每次训练得到模型参数之后需要判断 g(x)>0 时对应的类型是正的还是负的，如果是正的就在文件“limit_point”中添加一个“True”关键字，反之则添加“False”，在测试阶段需要结合这个关键字来判断是否识别正确。

4.3 模型测试（test_model.py）

在测试阶段，计算测试样本的投影值，然后与阈值做比较，统计正确识别率。

遇到问题：测试阶段的成功与否决定了实验的最终结果，前面过程出现的问题也会全部反应在这个阶段。我在测试阶段也遇到了一些问题，最终顺利解决，过程如下：

1、初次测试，设置 train_set : test_set = 7:3，刚开始测试阶段的识别率始终在略高于 50%的水平，而且程序输出的两类的中心点的投影位置相同，这样反而使得 Fisher 判别函数的值为 0 最小，而最后推到出的公式表示的是 Fisher 函数的极值点而并没有证明一定是极大值，因此对算法本身产生了怀疑（求出的极值可能是极小值），但根据矩阵特征向量的性质可知，w 的方向是唯一的，也就是找不出第二个 Fisher 函数的极值向量，并且初步验证 Fisher 函数应该是个上凸函数，问题可能不出在这。

2、接下来我怀疑是 python 的 numpy 包提供的计算协方差矩阵的函数 cov 出了问题，自己重写 cov 函数之后识别效果瞬间提升到了 90%以上，并且两类中心点的投影出现了微小的差别，随后我改变了训练集与测试集的划分比例，依次如下：
5:5、6:4、7:3、8:2、9:1,
识别率依次大概在这个水平：
50+%、 90+%、 90+% 、70+%、 50+%
为什么会出现训练集增加，识别率反而下跌，并且只在当训练测试比例在 5:5 的时候两类中心点投影有明显的区分，其他的都非常接近，这里我怀疑是数据精度的问题影响了最后的分类。后来的实验验证了这个想法，因为测试模型训练完成以后传递给测试程序是通过输出到一个文本文件的，用 str(float)默认的精度在这次实验中会有很严重的影响。通过格式化输出的精度能明显提升识别率，最后的结果如下：

5:5	6:4	7:3	8:2	9:1
0.621	0.95	0.975	0.8975	0.785

随着精度的进一步提升，所有配置下的识别率都能提高，但是在当配置大于 7:3 以后，识别率相对的下降，我猜测有两方面的原因：
1、测试集数量变小，因此统计结果偏差较大。但通过把训练集的一部分也扩充到测试集中，识别率仍然小于 7:3 时候的识别率，因此这应该不是主要影响因素；
2、本实验在最后选取阈值点的时候是按照书上的方法，在样本不是正太分布的时候，这样并不能保证最后的选取是最优的但一般能达到比较好的效果，因此，在找不到其他因素的情况下，我觉得识别率下降的主要因素应该是训练集的变化导致我们选取的阈值点偏离了最优点较远。

最后，本实验的最佳识别精度是 **97.5%**（7:3）

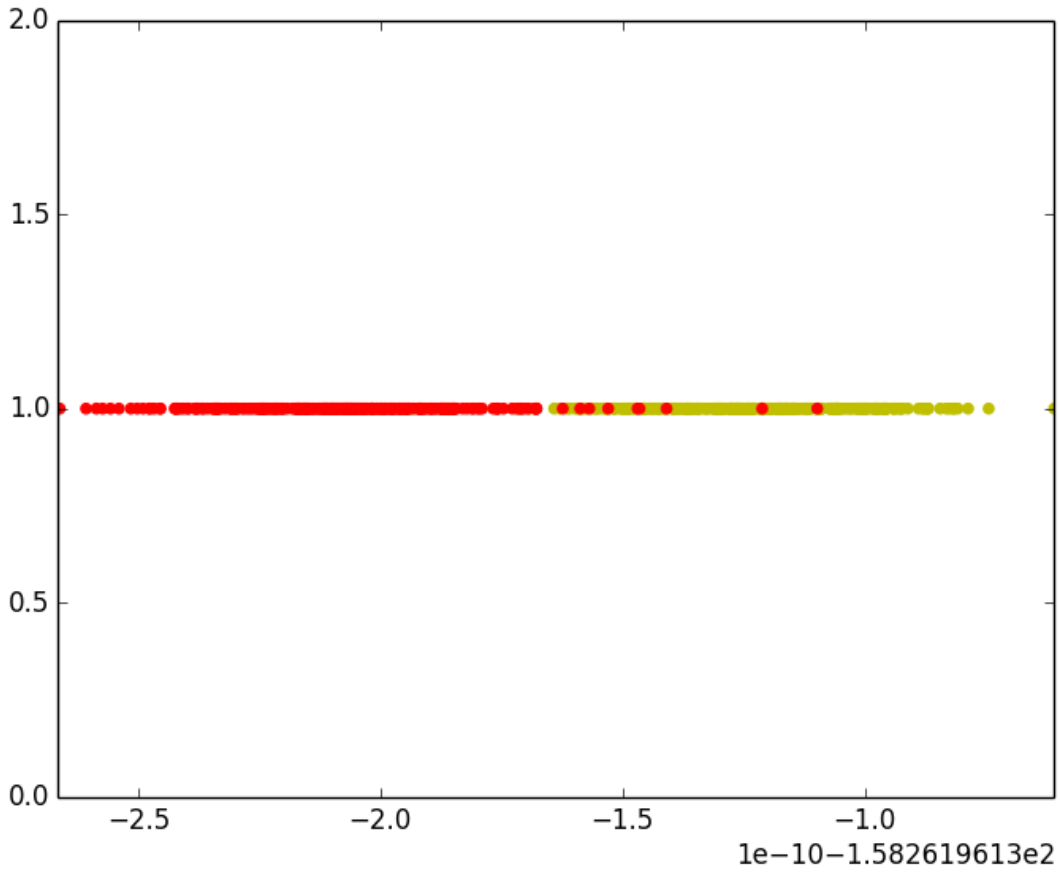
4.4 数据可视化（plot.py）

在模型测试阶段，测试实例在模型投影方向上的投影值按正例与负例被输出到文件 testcase.loc 中，本阶段把这些数据可视化。

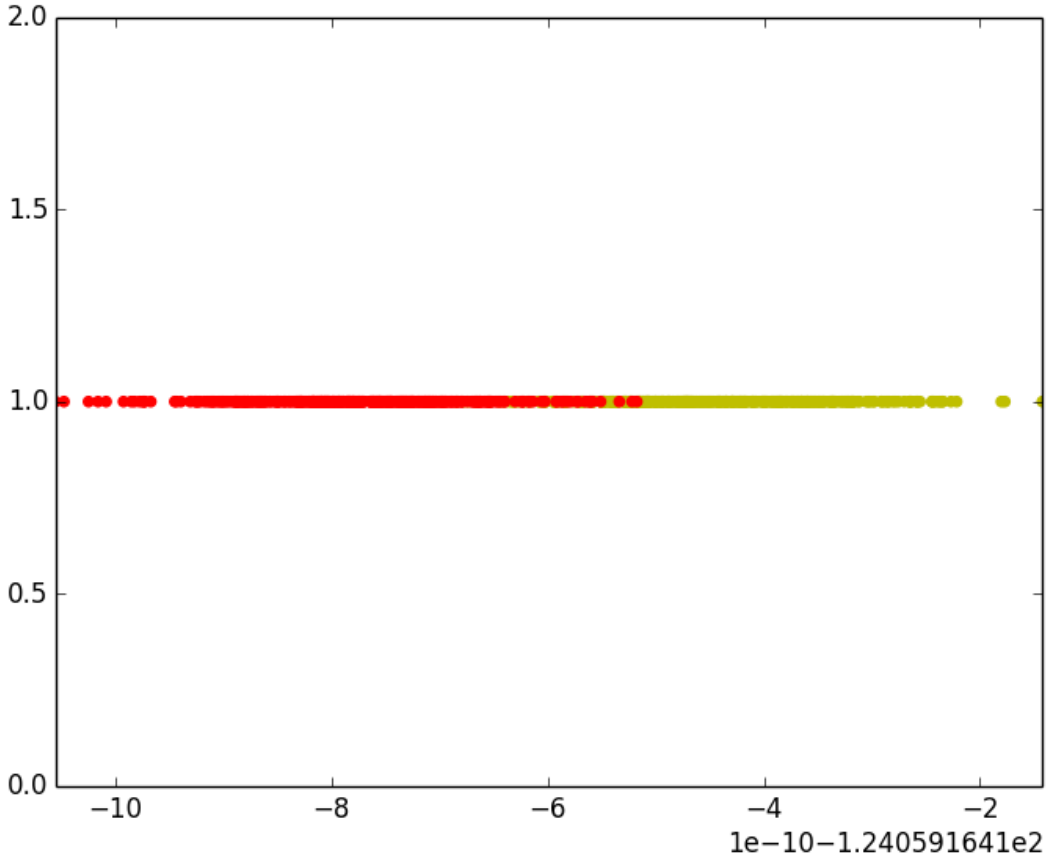
工具：matplotlib（一个基于 numpy 的 python 工具包，用于数据绘图）

可视化的目的是为了体现投影后正例与负例各自的分布情况，因此只需要一个维度，散点图是展示多个类别数据分布比较好的一个方式，最后的解决方案是：

以投影值为 x 轴坐标，每个样本点的纵坐标固定为 1，正例与负例分别用两种颜色标示出来，对 7:3 数据划分情况下的数据可是化的图片结果如下：
其中，红色代表负例，黄色代表正例。



作为对比，训练与测试比为 6:4 情况下的可视化图片为：


2.0
1.5
1.0
0.5
0.0
−10
−8
−6
−4
−2
1e−10−1.240591641e2