4+GaussianDescriminativeAnalysis

2019 年 3 月 9 日

1 高斯判别分析（Gaussian Discriminant Analysis）

1.1 一、概念

高斯判别分析是用于分类的有监督学习算法，但是略不同于 logistic regress。

Logistic regression 本质上是学习在给定自变量的情况下因变量的分布，也即条件分布 $P(y|x)$，学习的是从从样本空间 X 到标签 $\{0,1\}$ 的直接映射，这种算法被称为**判别学习算法（discriminative learning algorithm）**。

高斯判别分析学习的不是 $P(y|x)$，由 $P(y|x)=\frac{P(x,y)}{P(x)}=\frac{P(x|y)P(y)}{P(x)}$，其中 $P(y),P(x)$ 可以进行假定，比如高斯判别分析假定 X 属于多元正态分布，Y 属于伯努利分布，因此需要学习的分布就是 $P(x|y)$，这种算法被称为**生成学习算法（generative learning algorithms）**

高斯判别分析有如下假设：

1. 自变量 Y 服从伯努利分布：$y \sim Bernoulli(\phi)$
2. 在 Y=0 的条件下 X 服从多元正态分布：$x|y=0 \sim N(\mu_0, \Sigma)$
3. 在 Y=1 的条件下 X 服从多元正态分布：$x|y=1 \sim N(\mu_1, \Sigma)$

写成概率密度函数的形式则是：

$$
\begin{aligned}
p(y) &= \phi^y (1-\phi)^{1-y} \\
p(x|y=0) &= \frac{1}{(2\pi)^{\frac{p}{2}}|\Sigma|^{\frac{1}{2}}} exp\left(-\frac{1}{2}(x-\mu_0)^T \Sigma^{-1}(x-\mu_0)\right) \\
p(x|y=1) &= \frac{1}{(2\pi)^{\frac{p}{2}}|\Sigma|^{\frac{1}{2}}} exp\left(-\frac{1}{2}(x-\mu_1)^T \Sigma^{-1}(x-\mu_1)\right)
\end{aligned}
$$

其中，$y \in \{0,1\}$，$x, \mu_0, \mu_1 \in R^p$ ，X 是 p 维列向量，μ_0, μ_1 分别是对应的均值向量，$\Sigma \in R^{p\times p}$ 是 $p \times p$ 维协方差矩阵。

模型参数为概率分布中的参数：$\phi, \mu_0, \mu_1, \Sigma$。

1.2 二、优化

由于我们要学习的是概率分布的参数，而非函数映射，因此优化的目标不再是通过损失函数求导获得，而是通过求解极大似然估计。

$$\begin{aligned} l(\phi,\mu_0,\mu_1,\Sigma) &= log\prod_{i=1}^{n} p(x_i,y_i;\phi,\mu_0,\mu_1,\Sigma) \\ &= log\prod_{i=1}^{n} p(x_i|y_i;\mu_0,\mu_1,\Sigma)p(y_i;\phi) \end{aligned}$$

$$p(x,y) = \left[p\left(x|y=0\right)p\left(y=0\right)\right]^{1-y}\left[p\left(x|y=1\right)p\left(y=1\right)\right]^{y}$$

于是对数似然函数：

$$\begin{aligned} l(\phi,\mu_0,\mu_1,\Sigma) &= log\prod_{i=1}^{n} p(x_i|y_i;\mu_0,\mu_1,\Sigma)p(y_i;\phi) \\ &= \sum_{i=1}^{n}[(1-y_i)log(p(x_i|y_i=0)p(y_i=0)) + y_i log(p(x_i|y_i=1)p(y_i=1))] \\ &= \sum_{i=1}^{n}[(1-y_i)log(p(x_i|y_i=0)) + (1-y_i)log(p(y_i=0)) + y_i log(p(x_i|y_i=1)) + y_i log(p(y_i=1))] \\ &= \sum_{i=1}^{n}(1-y_i)[-\frac{n}{2}log(2\pi) - \frac{1}{2}log(|\Sigma|) - \frac{1}{2}(X-\mu_0)^T\Sigma^{-1}(X-\mu_0) + log(1-\phi)] \\ &+ \sum_{i=1}^{n} y_i[-\frac{n}{2}log(2\pi) - \frac{1}{2}log(|\Sigma|) - \frac{1}{2}(X-\mu_1)^T\Sigma^{-1}(X-\mu_1) + log(\phi)] \end{aligned}$$

对各个参数分别求导再令其等于零：

$$\begin{aligned} &\frac{\partial l(\phi,\mu_0,\mu_1,\Sigma)}{\partial\phi} = \sum_{i=1}^{n} -\frac{(1-y_i)}{1-\phi} + \frac{y_i}{\phi} = 0 \\ &\Leftrightarrow \sum_{i=1}^{n} -(1-y_i)\phi + y_i(1-\phi) = 0 \\ &\Leftrightarrow \phi = \frac{1}{n}\sum_{i=1}^{n} y_i \end{aligned}$$

对于 μ_0,μ_1,Σ 的极大似然估计可以参考《应用多元统计分析（北京大学出版社，高慧璇编著）》P38-P40，书中给到的公式是对于单一正态总体：

$$\begin{aligned} &\hat{\mu} = \bar{X} \\ &\Sigma = \frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{X})(x_i-\bar{X})^T \end{aligned}$$

对于高斯判别分析，两类对应不同的分布函数，因此分类进行极大似然估计即可：

$$\hat{\mu}_0 = \frac{\sum_{i=1}^{n} I(y_i = 0)x_i}{\sum_{i=1}^{n} I(y_i = 0)}$$

$$\hat{\mu}_1 = \frac{\sum_{i=1}^{n} I(y_i = 1)x_i}{\sum_{i=1}^{n} I(y_i = 1)}$$

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu}_{y_i})(x_i - \hat{\mu}_{y_i})^T$$

1.3 三、应用

当我们求到两个条件分布的概率密度函数之后，只需要将新样本带入两个分布的概率密度函数，函数值较大者，即是样本更有可能属于的那一类。

```
In [44]: import numpy as np
         import pandas as pd

         iris = pd.read_csv('Iris.data', header=None, names = ['sepal.l','sepal.w','petal.l','peta
         iris.index = iris['class']
         iris = iris.loc[['Iris-setosa', 'Iris-versicolor'],:]
         iris['class'].unique()

Out[44]: array(['Iris-setosa', 'Iris-versicolor'], dtype=object)

In [45]: n = iris.shape[0] # 样本数
         p = iris.shape[1]-1 # 变量数

         np.random.seed(2099)
         index = np.random.permutation(n) # 打乱样本索引

         train_index = index[0: int(0.7*n)]
         test_index = index[int(0.7*n): n]

         y = iris['class']
         y = y=='Iris-setosa'
         x = iris.drop(['class'], axis=1)

         train_x = x.iloc[train_index]
         train_y = y.iloc[train_index]
         test_x = x.iloc[test_index]
         test_y = y.iloc[test_index]
```

```
    train_y = np.array(train_y).reshape([int(0.7*n), 1])
    test_y = np.array(test_y).reshape([n-int(0.7*n), 1])
```

```
In [46]: versicolor = np.where(train_y==0)
         setosa = np.where(train_y==1)

         phi = train_y.mean()

         mu_0_hat = np.dot(train_x.T, 1-train_y)/np.sum(1-train_y)
         mu_1_hat = np.dot(train_x.T, train_y)/np.sum(train_y)

         train_x = np.array(train_x)
         sigma_x = train_x.copy()
         sigma_x[versicolor[0]] = sigma_x[versicolor[0]] - mu_0_hat.reshape([1,p])
         sigma_x[setosa[0]] = sigma_x[setosa[0]] - mu_1_hat.reshape([1,p])
         sigma = np.dot(train_x.T, train_x)/train_x.shape[0]

         print(mu_0_hat)
         print(mu_1_hat)
         print(sigma)
```

```
[[5.91470588]
 [2.75882353]
 [4.23823529]
 [1.32352941]]
[[5.02777778]
 [3.475     ]
 [1.49444444]
 [0.25277778]]
[[30.16985714 16.99785714 16.11471429  4.48242857]
 [16.99785714 10.02442857  8.39214286  2.24971429]
 [16.11471429  8.39214286  9.97014286  2.95457143]
 [ 4.48242857  2.24971429  2.95457143  0.90814286]]
```

```
In [47]: print(train_x)
         print(sigma_x)
```

```
[[6.6 2.9 4.6 1.3]
 [5.2 4.1 1.5 0.1]
 [5.  3.5 1.3 0.3]
 [5.2 3.5 1.5 0.2]
```

```
[5.5 2.5 4.  1.3]
[6.1 3.  4.6 1.4]
[5.7 4.4 1.5 0.4]
[5.1 3.8 1.5 0.3]
[5.6 3.  4.5 1.5]
[5.6 2.7 4.2 1.3]
[6.  2.2 4.  1. ]
[5.  3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[6.6 3.  4.4 1.4]
[4.8 3.  1.4 0.1]
[5.8 4.  1.2 0.2]
[6.  2.7 5.1 1.6]
[5.1 3.5 1.4 0.2]
[6.7 3.1 4.7 1.5]
[5.6 3.  4.1 1.3]
[6.2 2.9 4.3 1.3]
[5.  3.4 1.5 0.2]
[6.3 3.3 4.7 1.6]
[5.6 2.5 3.9 1.1]
[4.8 3.4 1.9 0.2]
[6.1 2.8 4.  1.3]
[5.  3.  1.6 0.2]
[5.6 2.9 3.6 1.3]
[4.4 3.  1.3 0.2]
[5.5 2.3 4.  1.3]
[4.9 3.1 1.5 0.2]
[4.7 3.2 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.5 4.2 1.4 0.2]
[5.4 3.9 1.7 0.4]
[5.2 2.7 3.9 1.4]
[6.7 3.1 4.4 1.4]
[4.6 3.1 1.5 0.2]
[5.7 2.6 3.5 1. ]
[4.3 3.  1.1 0.1]
[6.5 2.8 4.6 1.5]
[5.  2.  3.5 1. ]
[5.1 3.5 1.4 0.3]
```

```
 [6.3 2.3 4.4 1.3]
 [5.4 3.7 1.5 0.2]
 [5.5 2.4 3.7 1. ]
 [4.8 3.1 1.6 0.2]
 [5.7 3.8 1.7 0.3]
 [6.2 2.2 4.5 1.5]
 [6.  2.9 4.5 1.5]
 [5.3 3.7 1.5 0.2]
 [4.9 2.4 3.3 1. ]
 [5.1 3.4 1.5 0.2]
 [5.7 3.  4.2 1.2]
 [5.1 3.7 1.5 0.4]
 [4.9 3.  1.4 0.2]
 [5.4 3.  4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [5.7 2.9 4.2 1.3]
 [5.7 2.8 4.1 1.3]
 [4.7 3.2 1.3 0.2]
 [4.8 3.  1.4 0.3]
 [5.4 3.4 1.5 0.4]
 [4.6 3.4 1.4 0.3]
 [4.8 3.4 1.6 0.2]
 [5.8 2.7 4.1 1. ]
 [5.1 3.8 1.6 0.2]
 [5.9 3.2 4.8 1.8]
 [6.4 2.9 4.3 1.3]
 [5.  3.4 1.6 0.4]]
[[ 0.68529412  0.14117647  0.36176471 -0.02352941]
 [ 0.17222222  0.625       0.00555556 -0.15277778]
 [-0.02777778  0.025      -0.19444444  0.04722222]
 [ 0.17222222  0.025       0.00555556 -0.05277778]
 [-0.41470588 -0.25882353 -0.23823529 -0.02352941]
 [ 0.18529412  0.24117647  0.36176471  0.07647059]
 [ 0.67222222  0.925       0.00555556  0.14722222]
 [ 0.07222222  0.325       0.00555556  0.04722222]
 [-0.31470588  0.24117647  0.26176471  0.17647059]
 [-0.31470588 -0.05882353 -0.03823529 -0.02352941]
 [ 0.08529412 -0.55882353 -0.23823529 -0.32352941]
 [-0.02777778  0.025       0.10555556  0.34722222]
```

```
[ 0.07222222  0.325       0.40555556  0.14722222]
[ 0.68529412  0.24117647  0.16176471  0.07647059]
[-0.22777778 -0.475      -0.09444444 -0.15277778]
[ 0.77222222  0.525      -0.29444444 -0.05277778]
[ 0.08529412 -0.05882353  0.86176471  0.27647059]
[ 0.07222222  0.025      -0.09444444 -0.05277778]
[ 0.78529412  0.34117647  0.46176471  0.17647059]
[-0.31470588  0.24117647 -0.13823529 -0.02352941]
[ 0.28529412  0.14117647  0.06176471 -0.02352941]
[-0.02777778 -0.075       0.00555556 -0.05277778]
[ 0.38529412  0.54117647  0.46176471  0.27647059]
[-0.31470588 -0.25882353 -0.33823529 -0.22352941]
[-0.22777778 -0.075       0.40555556 -0.05277778]
[ 0.18529412  0.04117647 -0.23823529 -0.02352941]
[-0.02777778 -0.475       0.10555556 -0.05277778]
[-0.31470588  0.14117647 -0.63823529 -0.02352941]
[-0.62777778 -0.475      -0.19444444 -0.05277778]
[-0.41470588 -0.45882353 -0.23823529 -0.02352941]
[-0.12777778 -0.375       0.00555556 -0.05277778]
[-0.32777778 -0.275       0.10555556 -0.05277778]
[-0.42777778 -0.275      -0.09444444 -0.05277778]
[ 0.47222222  0.725      -0.09444444 -0.05277778]
[ 0.37222222  0.425       0.20555556  0.14722222]
[-0.71470588 -0.05882353 -0.33823529  0.07647059]
[ 0.78529412  0.34117647  0.16176471  0.07647059]
[-0.42777778 -0.375       0.00555556 -0.05277778]
[-0.21470588 -0.15882353 -0.73823529 -0.32352941]
[-0.72777778 -0.475      -0.39444444 -0.15277778]
[ 0.58529412  0.04117647  0.36176471  0.17647059]
[-0.91470588 -0.75882353 -0.73823529 -0.32352941]
[ 0.07222222  0.025      -0.09444444  0.04722222]
[ 0.38529412 -0.45882353  0.16176471 -0.02352941]
[ 0.37222222  0.225       0.00555556 -0.05277778]
[-0.41470588 -0.35882353 -0.53823529 -0.32352941]
[-0.22777778 -0.375       0.10555556 -0.05277778]
[ 0.67222222  0.325       0.20555556  0.04722222]
[ 0.28529412 -0.55882353  0.26176471  0.17647059]
[ 0.08529412  0.14117647  0.26176471  0.17647059]
[ 0.27222222  0.225       0.00555556 -0.05277778]
```

```
 [-1.01470588 -0.35882353 -0.93823529 -0.32352941]
 [ 0.07222222 -0.075       0.00555556 -0.05277778]
 [-0.21470588  0.24117647 -0.03823529 -0.12352941]
 [ 0.07222222  0.225       0.00555556  0.14722222]
 [-0.12777778 -0.475      -0.09444444 -0.05277778]
 [-0.51470588  0.24117647  0.26176471  0.17647059]
 [ 0.98529412  0.34117647  0.66176471  0.17647059]
 [-0.21470588  0.14117647 -0.03823529 -0.02352941]
 [-0.21470588  0.04117647 -0.13823529 -0.02352941]
 [-0.32777778 -0.275      -0.19444444 -0.05277778]
 [-0.22777778 -0.475      -0.09444444  0.04722222]
 [ 0.37222222 -0.075       0.00555556  0.14722222]
 [-0.42777778 -0.075      -0.09444444  0.04722222]
 [-0.22777778 -0.075       0.10555556 -0.05277778]
 [-0.11470588 -0.05882353 -0.13823529 -0.32352941]
 [ 0.07222222  0.325       0.10555556 -0.05277778]
 [-0.01470588  0.44117647  0.56176471  0.47647059]
 [ 0.48529412  0.14117647  0.06176471 -0.02352941]
 [-0.02777778 -0.075       0.10555556  0.14722222]]
```

```python
In [48]: def multi_norm(x, mu, sigma):
             p = sigma.shape[0]
             pdf = np.exp(-np.dot(np.dot(x - mu.T, np.linalg.inv(sigma)), x.T-mu)/2)/np.power(2*np
             return np.diag(pdf)

         print(multi_norm(train_x, mu=mu_0_hat, sigma=sigma))
         print(multi_norm(train_x, mu=mu_1_hat, sigma=sigma))
```

```
[0.05158901 0.00071774 0.00684229 0.01363433 0.10481722 0.06113685
 0.00119583 0.00915141 0.01488632 0.09046647 0.00198879 0.00050439
 0.00628861 0.06557128 0.00694552 0.0017596  0.03971691 0.01374556
 0.10110418 0.04544751 0.11081438 0.01494656 0.03028553 0.06698615
 0.00074869 0.05341568 0.00926372 0.01447576 0.02068535 0.0755752
 0.01407197 0.01128818 0.01744431 0.00503466 0.00975464 0.02501363
 0.06469796 0.01640875 0.0545105  0.01508242 0.05888063 0.04887789
 0.00996067 0.0104076  0.011924   0.04287573 0.01368134 0.01322966
 0.0059869  0.07935464 0.01220885 0.10763214 0.01439774 0.01953495
 0.00598368 0.00921308 0.00431031 0.10156933 0.06861011 0.10397956
 0.01657193 0.00958558 0.00167008 0.01291809 0.00843288 0.00432163
 0.00473276 0.00132238 0.08790556 0.01420927]
```

```
[6.82145974e-03 9.41447425e-03 5.48385881e-02 1.03099458e-01
 1.47588959e-02 7.36247380e-03 1.62449514e-02 7.82431323e-02
 1.65550454e-03 1.25868332e-02 3.16592263e-04 2.34399683e-03
 3.41785424e-02 1.02391409e-02 4.03174530e-02 2.81405131e-02
 1.98331107e-03 1.12000920e-01 1.20493382e-02 8.71010007e-03
 1.80457304e-02 1.01436993e-01 3.60534114e-03 1.25882520e-02
 3.42851064e-03 1.04038107e-02 4.22756888e-02 4.03947900e-03
 1.12648290e-01 9.12996701e-03 7.47071601e-02 5.72674304e-02
 1.04390575e-01 7.46950956e-02 7.19122221e-02 3.91249043e-03
 1.10804410e-02 8.30864459e-02 1.76740808e-02 1.06137068e-01
 5.91500352e-03 8.96228628e-03 7.40861682e-02 9.93341840e-04
 1.08462016e-01 9.64407135e-03 6.53127294e-02 1.03761861e-01
 3.96574604e-04 8.70669407e-03 1.09315009e-01 3.16252943e-02
 9.92660289e-02 3.80626848e-03 4.32564016e-02 4.95945153e-02
 4.64461899e-04 1.04251885e-02 1.13031420e-02 1.73692347e-02
 1.10285593e-01 4.63645497e-02 1.00594736e-02 8.22482361e-02
 5.06573229e-02 8.93092907e-04 4.04944640e-02 1.04205060e-04
 1.47739463e-02 7.34038100e-02]
```

```
In [50]: is_versicolor = multi_norm(train_x, mu=mu_0_hat, sigma=sigma)
         is_setosa = multi_norm(train_x, mu=mu_1_hat, sigma=sigma)
         train_y_hat = is_versicolor < is_setosa
         print(train_y_hat)
```

```
[False  True  True  True False False  True  True False False False  True
  True False  True  True False  True False False False  True False False
  True False  True False  True False  True  True  True  True  True False
 False  True False  True False False  True False  True False  True  True
 False False  True False  True False  True  True False False False False
  True  True  True  True  True False  True False False  True]
```

```
In [53]: def accuracy(y, y_hat):
             n = y.shape[0]
             return np.sum(y==y_hat)/n

         train_y_hat = np.array(train_y_hat).reshape([len(train_y_hat), 1])
         train_acc = accuracy(train_y, train_y_hat)
         print(train_acc)
```

```
1.0
```

```
In [54]: is_versicolor = multi_norm(test_x, mu=mu_0_hat, sigma=sigma)
         is_setosa = multi_norm(test_x, mu=mu_1_hat, sigma=sigma)
         test_y_hat = is_versicolor < is_setosa
         print(test_y_hat)
```

```
[False False  True False  True  True False  True False False False  True
  True False  True False False False False  True  True False False  True
 False False  True  True  True  True]
```

```
In [55]: test_y_hat = np.array(test_y_hat).reshape([len(test_y_hat), 1])
         test_acc = accuracy(test_y, test_y_hat)
         print(test_acc)
```

```
1.0
```

```
In [ ]:
```