《Python机器学习》读书笔记（六）特征抽取——LDA - 没有名字 - CSDN博客

Python机器学习读书笔记（六）特征抽取——LDA

说明：

关于本书：《Python机器学习》

本笔记侧重代码调用，只描述了一些简单概念，本书的公式推导不在这里展示

接上文《Python机器学习》读书笔记（五）特征抽取——PCA

特征抽取 可以将原始数据集变换到一个维度更低的新的特征子空间，在尽可能多地保持相关信息

2. 线性判别分析 Linear Discriminate Analysis, LDA

2.1 简单介绍

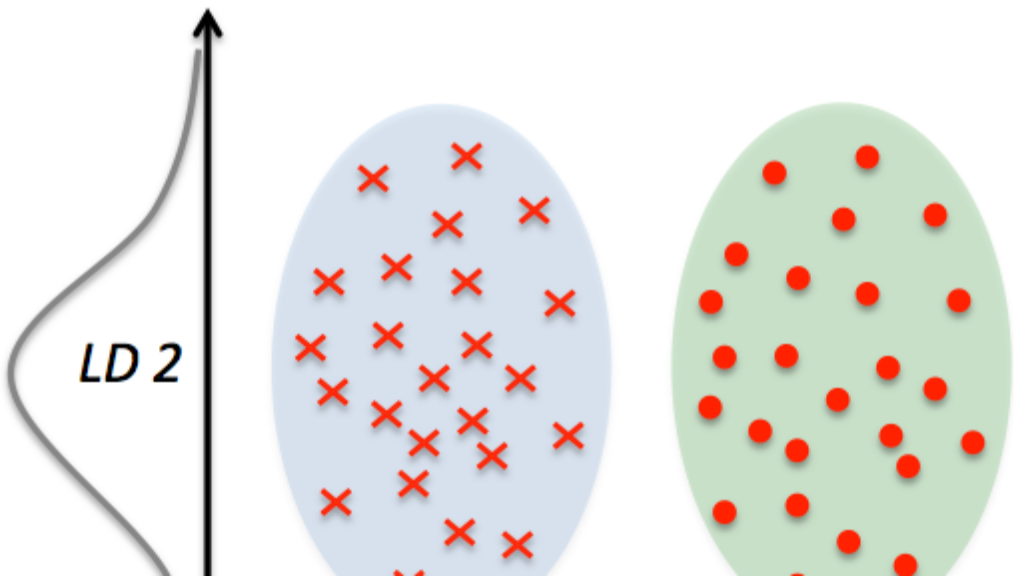
LDA是一种可作为**特征抽取**的技术

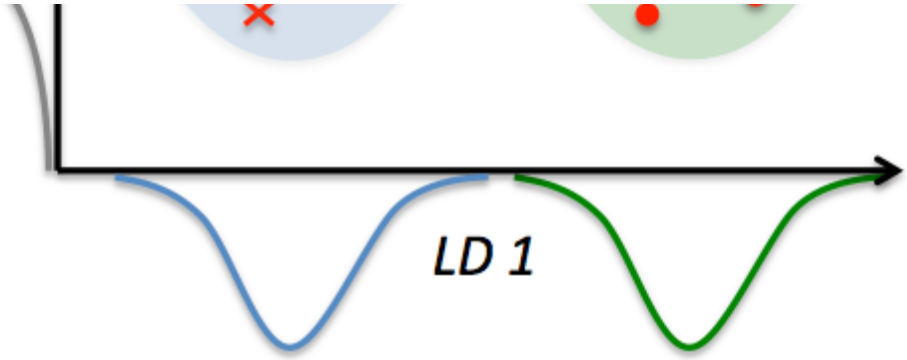
可以提高数据分析过程中的计算效率

对于不适用与正则化的模型，可以降低因维度灾难带来的过拟合

监督算法

目标：发现可以最优化分类的特征子空间





如图所示，在x轴方向，通过线性判定，可以很好的将呈正态分布的两个类分开

虽然沿y轴方向的线性判定保持了数据集的较大方差，但是无法提供关于类别区分的任何信息

2.2 算法

思想：给定训练集样例，设法将样例投影到一条直线上，使得同**类样例的投影尽可能接近**，**异类样例的投**
投影点的位置来确定新样本的类别。（下图截自 周志华《机器学习》）

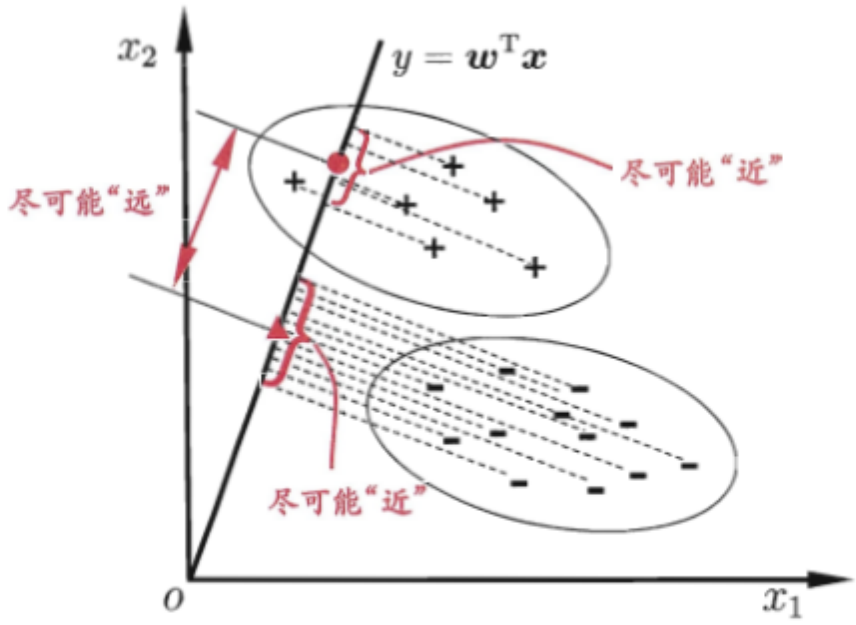


图 3.3 LDA 的二维示意图. “+”、“−”分别代表正例和反例, 椭圆表示数据簇外轮廓, 虚线表示投影, 红色实心圆和实心三角形分别表示两类样本投影后的中心点

假设:

数据呈正态分布

各类别数据具有相同的协方差矩阵

样本的特征从统计上来说相互独立

事实上，即使违背上述假设，LDA仍能正常工作

LDA关键步骤：

> 对d维数据进行标准化处理（d为特征数量）
>
> 对于每一类别，计算d维的均值向量
>
> 构造类间的**散布矩阵**
> S_B
> 以及 **类内散布矩阵**
> S_W
>
> 计算矩阵
> $S_W^{-1}S_B$
> 的特征值以及对应的特征向量
>
> 选取前k个特征值所对应的特征向量，构造一个
> $d * k$
> 维的转换矩阵
> W
> ,其中特征向量以列的形式排列
>
> 使用转换矩阵
> W
> 将样本映射到新的特征子空间上

若将
W
视为一个投影矩阵，则多分类LDA将样本投影到
d
维空间（
$d << d$
），于是达到了降维的目的

在投影过程中用到了类别信息

2.3 示例

加载数据集

```
import pandas as pd

df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/'
                      'machine-learning-databases/wine/wine.data',
                      header=None)
```

划分数据集

```
if Version(sklearn_version) < '0.18':
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.3, random_state=0)
```

标准化

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

计算均值向量

$$m_i = \frac{1}{n_i} \sum_{x \in D_i}^{c} x_m$$

$$m_i = \begin{bmatrix} \mu_{i,alcohol} \\ \mu_{i,malicacid} \\ \cdots \\ \mu_{i,profilr} \end{bmatrix}$$

```
np.set_printoptions(precision=4)

mean_vecs = []
```

```
for label in range(1, 4):
    mean_vecs.append(np.mean(X_train_std[y_train == label], axis=0))
    print('MV %s: %s\n' % (label, mean_vecs[label - 1]))
```

```
MV 1: [ 0.9259 -0.3091  0.2592 -0.7989  0.3039  0.9608  1.0515 -0.6306  0.5354
  0.2209  0.4855  0.798   1.2017]

MV 2: [-0.8727 -0.3854 -0.4437  0.2481 -0.2409 -0.1059  0.0187 -0.0164  0.1095
 -0.8796  0.4392  0.2776 -0.7016]

MV 3: [ 0.1637  0.8929  0.3249  0.5658 -0.01   -0.9499 -1.228   0.7436 -0.7652
  0.979  -1.1698 -1.3007 -0.3912]
```

计算 **类内散布矩阵**
S_W

$$S_w = \sum_{i=1}^{c} S_i$$

$$S_i = \sum_{x \in D_i}^{c} (x - m_i)(x -$$

```
d = 13  # number of features
S_W = np.zeros((d, d))
for label, mv in zip(range(1, 4), mean_vecs):
    class_scatter = np.zeros((d, d))  # scatter matrix for each class
    for row in X_train_std[y_train == label]:
        row, mv = row.reshape(d, 1), mv.reshape(d, 1)  # make column vecto
        class_scatter += (row - mv).dot((row - mv).T)
    S_W += class_scatter                             # sum class scatter matri
```

```
    S_W += class_scatter                                  # sum class scatter matri

print('Within-class scatter matrix: %sx%s' % (S_W.shape[0], S_W.shape[1]))

# Within-class scatter matrix: 13x13
```

计算**散布矩阵**
S_B
，其中
m
是全局均值，在计算时用到了所有类别中的全部样本

$$S_B = \sum_{i=1}^{c} N_i(m - m_i)(m$$

```
mean_overall = np.mean(X_train_std, axis=0)
d = 13  # number of features
S_B = np.zeros((d, d))
for i, mean_vec in enumerate(mean_vecs):
    n = X_train[y_train == i + 1, :].shape[0]
    mean_vec = mean_vec.reshape(d, 1)  # make column vector
    mean_overall = mean_overall.reshape(d, 1)  # make column vector
    S_B += n * (mean_vec - mean_overall).dot((mean_vec - mean_overall).T)
```

求解矩阵
$S_W^{-1}S_B$
的广义特征值

```
eigen_vals, eigen_vecs = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))
```

接着按照降序对特征值进行排序

```
# Make a list of (eigenvalue, eigenvector) tuples
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:, i])
               for i in range(len(eigen_vals))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eigen_pairs = sorted(eigen_pairs, key=lambda k: k[0], reverse=True)

# Visually confirm that the list is correctly sorted by decreasing eigenval

print('Eigenvalues in decreasing order:\n')
for eigen_val in eigen_pairs:
    print(eigen_val[0])
```

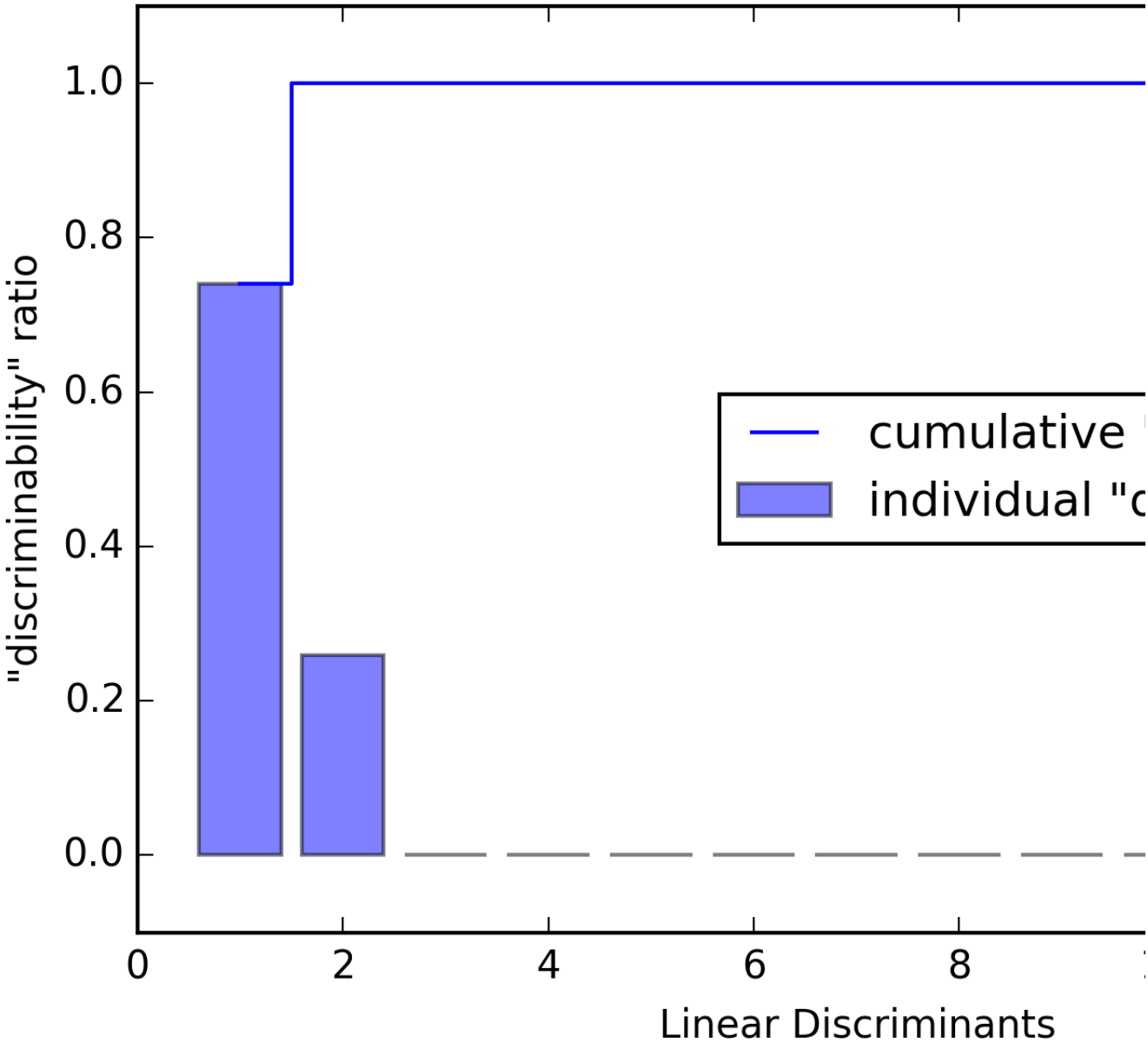
```
Eigenvalues in decreasing order:

452.721581245
156.43636122
1.05646703435e-13
3.99641853702e-14
3.40923565291e-14
2.84217094304e-14
1.4793035293e-14
1.4793035293e-14
1.3494134504e-14
1.3494134504e-14
6.49105985585e-15
6.49105985585e-15
2.65581215704e-15
```

为了度量LDA可以获取多少区分类别的信息，可以按照特征降序绘制出特征对线性判别信息保持程度的图

```
tot = sum(eigen_vals.real)
discr = [(i / tot) for i in sorted(eigen_vals.real, reverse=True)]
cum_discr = np.cumsum(discr)

plt.bar(range(1, 14), discr, alpha=0.5, align='center',
        label='individual "discriminability"')
plt.step(range(1, 14), cum_discr, where='mid',
         label='cumulative "discriminability"')
plt.ylabel('"discriminability" ratio')
plt.xlabel('Linear Discriminants')
plt.ylim([-0.1, 1.1])
plt.legend(loc='best')
plt.tight_layout()
# plt.savefig('./figures/lda1.png', dpi=300)
plt.show()
```



叠加两个判别能力最强的特征向量列构建转换矩阵
W
:

```
w = np.hstack((eigen_pairs[0][1][:, np.newaxis].real,
               eigen_pairs[1][1][:, np.newaxis].real))
print('Matrix W:\n', w)
```

```
Matrix W:
 [[-0.0662 -0.3797]
 [ 0.0386 -0.2206]
 [-0.0217 -0.3816]
 [ 0.184   0.3018]
 [-0.0034  0.0141]
 [ 0.2326  0.0234]
 [-0.7747  0.1869]
 [-0.0811  0.0696]
```

```
[ 0.0875  0.1796]
[ 0.185  -0.284 ]
[-0.066   0.2349]
[-0.3805  0.073 ]
[-0.3285 -0.5971]]
```

将样本映射到新的特征空间

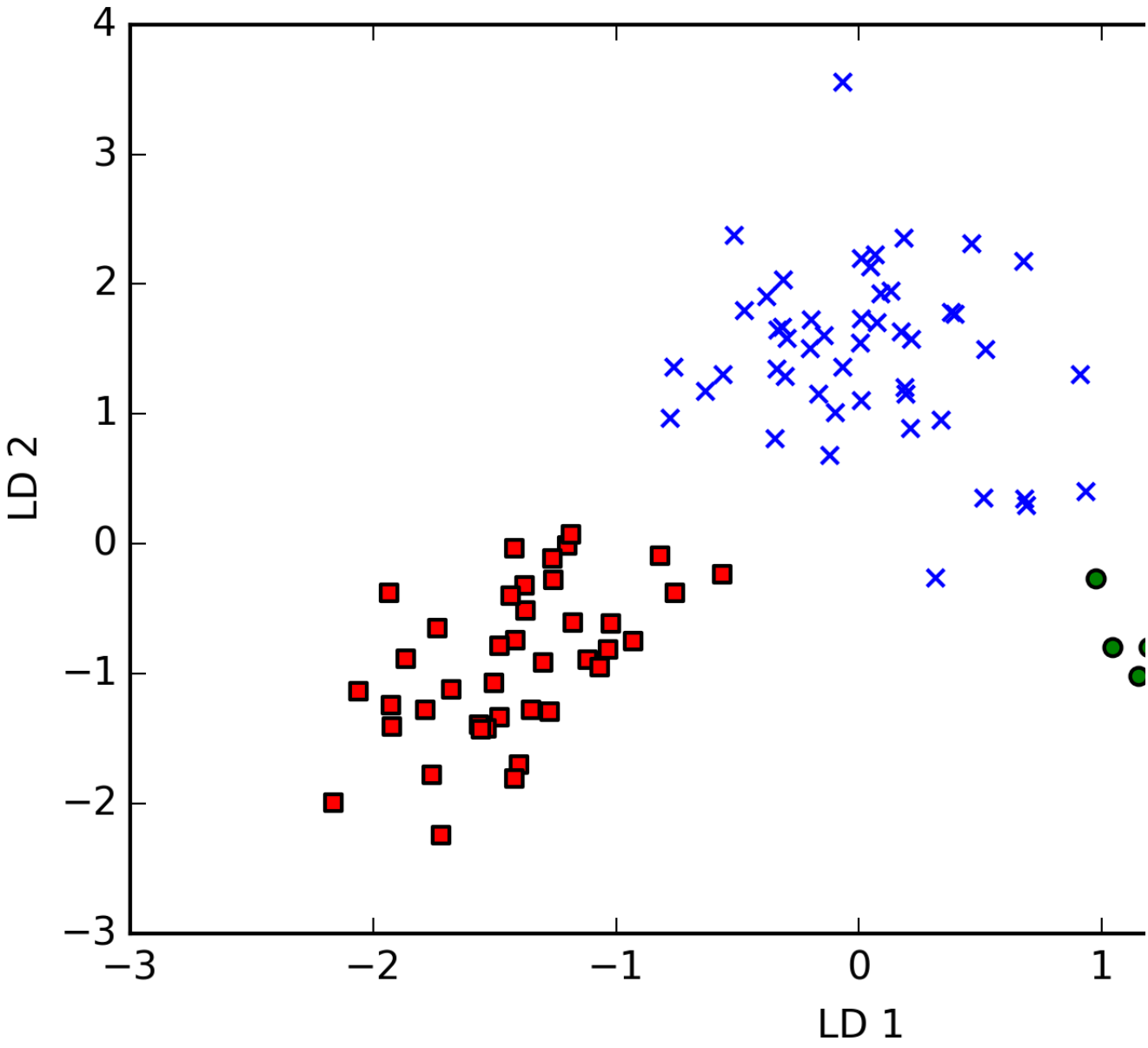
```
X_train_lda = X_train_std.dot(w)
colors = ['r', 'b', 'g']
markers = ['s', 'x', 'o']

for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_lda[y_train == l, 0] * (-1),
                X_train_lda[y_train == l, 1] * (-1),
                c=c, label=l, marker=m)

plt.xlabel('LD 1')
plt.ylabel('LD 2')
plt.legend(loc='lower right')
plt.tight_layout()
# plt.savefig('./figures/lda2.png', dpi=300)
plt.show()
```

```
8
9
10
11
12
13
14
15
```

可以看到，三个葡萄酒类在新的特征子空间上是线性可分的



2.4 使用sklearn进行LDA分析

调用sklearn中的LDA,使用Logistic回归模型：

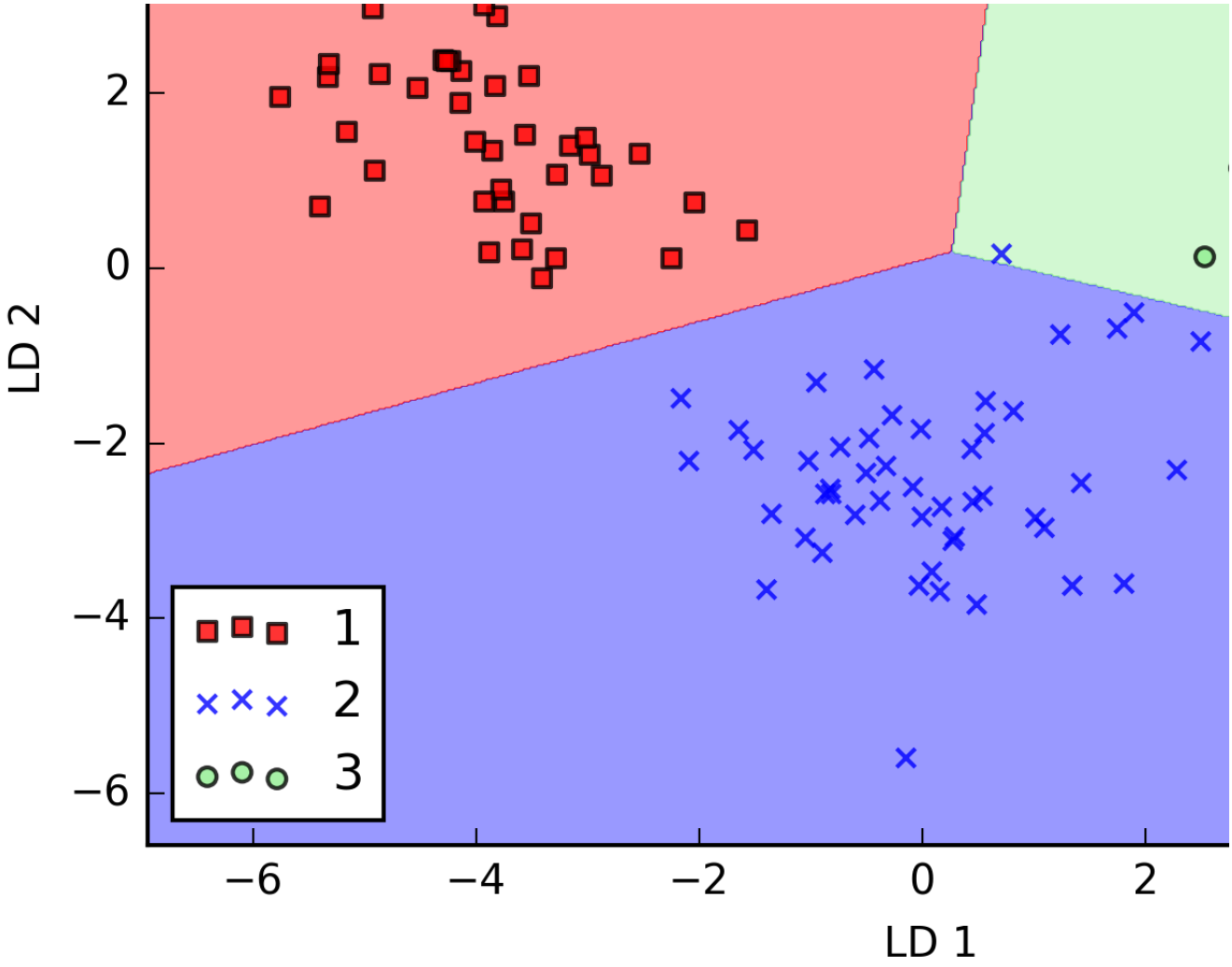
```
if Version(sklearn_version) < '0.18':
    from sklearn.lda import LDA
else:
    from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as

lda = LDA(n_components=2)
X_train_lda = lda.fit_transform(X_train_std, y_train)

from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr = lr.fit(X_train_lda, y_train)

plot_decision_regions(X_train_lda, y_train, classifier=lr)
plt.xlabel('LD 1')
plt.ylabel('LD 2')
plt.legend(loc='lower left')
plt.tight_layout()
# plt.savefig('./images/lda3.png', dpi=300)
plt.show()
```

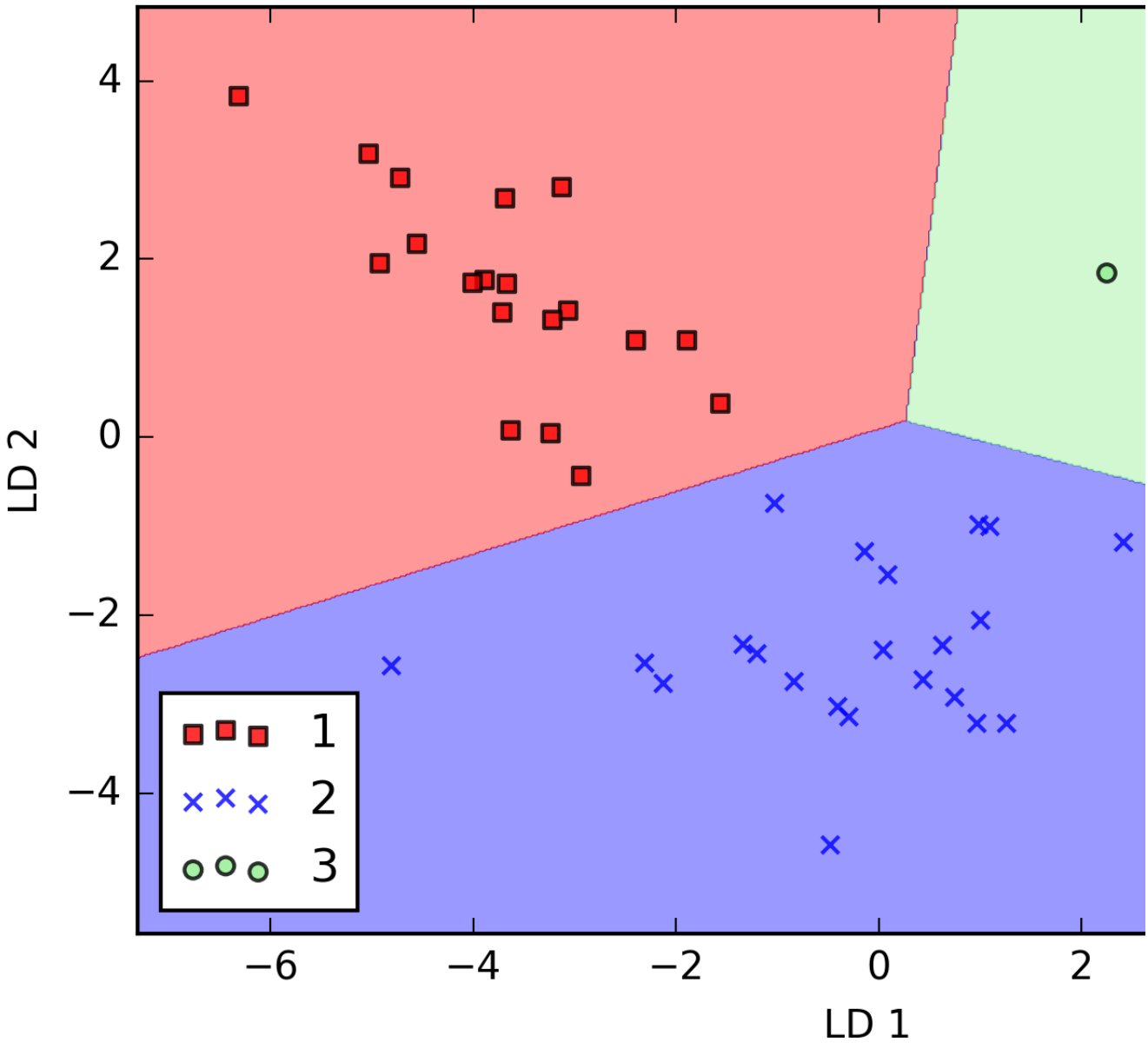
可以看到只有两个样本被误分类。可以通过正则化对决策边界进行调整。



在测试集上的表现

```
X_test_lda = lda.transform(X_test_std)

plot_decision_regions(X_test_lda, y_test, classifier=lr)
plt.xlabel('LD 1')
plt.ylabel('LD 2')
plt.legend(loc='lower left')
plt.tight_layout()
# plt.savefig('./images/lda4.png', dpi=300)
plt.show()
```


4
2
0
−2
−4
LD 2
−6
−4
−2
0
2
LD 1
1
2
3