15.1 线性判别分析LDA

线性判别式分析(Linear Discriminant Analysis,LDA)，也叫作Fisher线性判别(Fisher Linear Discriminant,FLD)，最开始用于处理机器学习中的分类任务，但是由于其对数据特征进行了降维投影， 使其成为一种经典的降维方法。

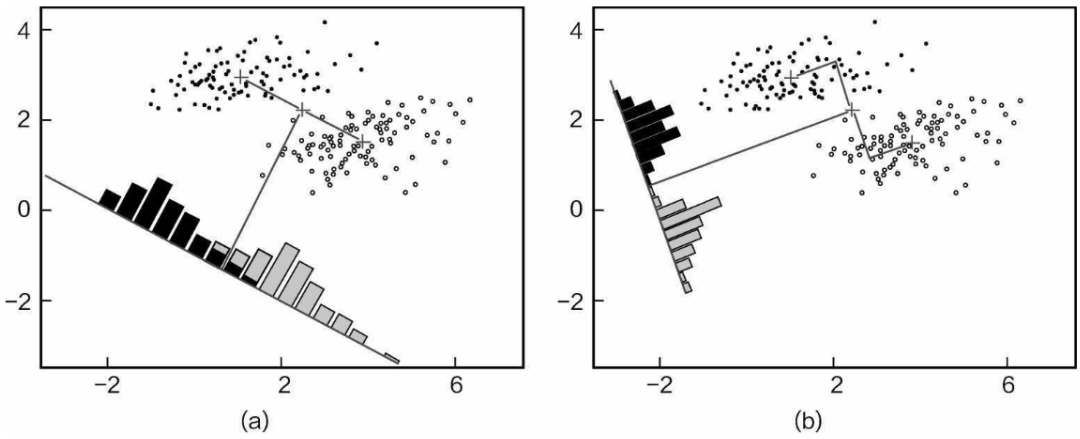
15.1.1 降维原理概述

背景：怎么降维好?

线性判别分析属于**有监督学习算法**，也就是数据中必须要有明确的类别标签，它不仅能用来降维， 还可以处理分类任务，不过，更多用于降维。下面通过一个小例子来感受下降维的作用。这个游戏需要 通过不断地寻找最合适的投影面，来观察原始物体的形状，如图15-1所示。降维任务与之非常相似，也是 通过找到最合适的投影方向，使得原始数据更容易被计算机理解并利用。

图15-1 投影的意义

接下来，通过实例说明降维的过程。假设有两类数据点，如图15-2所示。由于数据点都是二维特征， 需要将其降到一维，也就是需要找到一个最合适的投影方向把这些数据点全部映射过去。图15-2(a)、 (b)分别是选择两个投影方向后的结果，那么，究竟哪一个更好呢?



从投影结果上观察，图15-2(a)中的数据点经过投影后依旧有一部分混在一起，区别效果有待提 高。图15-2(b)中的数据点经过投影后，没有混合，区别效果比图15-2(a)更好一些。因此，我们当然 会选择图15-2(b)所示的降维方法，由此可见，降维不仅要压缩数据的特征，还需要寻找最合适的方 向，使得压缩后的数据更有利用价值。

由图15-2可知，线性判别分析的原理可以这样理解:任务目标就是要找到最合适的投影方向，这个方 向可以是多维的。

分类降维的目的以及目标

为了把降维任务做得更圆满，实现“尽可能使两类数据点区别得越明显越好，不要混在一起”的目的，提出了**两个目标**:

目标1.对于不同类别的数据点，希望其经过投影后能离得越远越好。（不同类分散）

目标2.对于同类别的数据点，希望它们能更集中，离组织的中心越近越好。（同类集中）

接下来的任务就是完成这两个目标，这也是线性判别分析的核心优化目标，降维任务就是找到能同 时满足这两个目标的投影方向。

15.1.2 目标的求解

投影就是通过矩阵变换的方式把数据映射到最适合做分类的方向上:

$$y = w^{\mathrm{T}} x$$

其中，x表示当前数据所在空间，也就是原始数据;y表示降维后的数据。最终的目标也很明显，就 是找到最合适的变换方向，即求解出参数W。除了降维任务中提出的两个目标，还需要定义一下距离这个 概念，例如“扎堆”该怎么体现呢?这里用数据点的均值来表示其中心位置，如果每一个数据点都离中心 很近，它们就“扎堆”在一起了。中心点位置计算方法如下:

$$\mu_i = \frac{1}{N_i} \sum_{x \in \omega_i} x$$

对于多分类问题，也可以得到各自类别的中心点，不同类别需要各自计算，这就是为什么要强调线 性判别分析是一个有监督问题，因为需要各个类别分别进行计算，所以每一个数据点是什么类别，必须 在标签中给出。

在降维算法中，其实我们更关心的并不是原始数据集中数据点的扎堆情况，而是降维后的结果，因此，可知投影后中心点位置为:

$$\tilde{\mu}_i = \frac{1}{N_i} \sum_{x \in w_i} w^{\mathrm{T}} x = w^{\mathrm{T}} \mu_i$$

衡量不同类远离度

可以得到投影后的中心点计算方法，按照之前制定的目标，对于一个二分类任务来说， 应当使得这两类数据点的中心离得越远越好，这样才能更好地区分它们:

$$J(w) = \left| \tilde{\mu}_1 - \tilde{\mu}_2 \right| = \left| w^{\mathrm{T}} (\mu_1 - \mu_2) \right|$$

现在可以把 当作目标函数，目标是希望其值能够越大越好，但是只让不同类别投影后的中心点 越远可以达到我们期望的结果吗?

对于图15-3所示的样本数据，假设只能在x1和x2两方向进行投影，如果按照之前定义的 ，显然x1 方向更合适，但是投影后两类数据点依旧有很多重合在一起，而x2方向上的投影结果是两类数据点重合较 少;因此，x2方向更好。

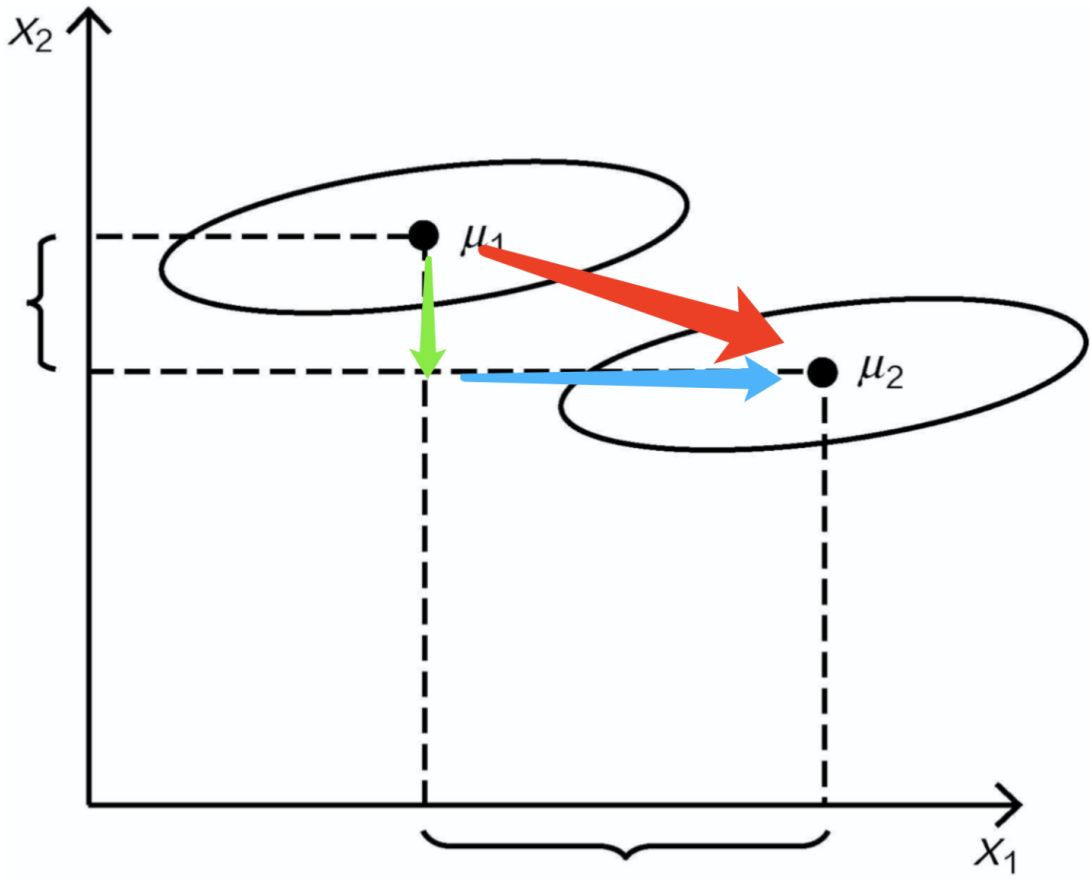


图15-3 投影方向选择

这个问题就涉及要优化的另一个目标，**不仅要考虑不同类别之间要区分开，还要考虑同类样本点应当尽可能聚集在一起**。显然在图15-3中，x1方向不满足这个条件，因为在x1方向上，同类样本变得更分散，不够集中。

衡量同类集中度

我们还可以使用另一个度量指标——散列值(scatter)，表示同类数据样本点的离散程度，定义如下:

$$\tilde{s}_t^2 = \sum_{y \in Y_i} (y - \tilde{\mu}_i)^2$$

其中，y表示经过投影后的数据点，从式(15.5)中可以看出，散列值表示样本点的密集程度，其值越大，表示越分散;反之，则越集中。定义好要优化的两个目标后，接下来就是求解了。

15.1.3 线性判别分析求解

上一小节已经介绍了降维后想要得到的目标，现在把它们综合在一起，但是优化的目标有两个，那 么如何才能整合它们呢?

既然要最大化不同类别之间的距离，那就把它当作分子;最小化同类样本之间的离散程度，那就把 它当作分母，最终整体的J(W)依旧求其极大值即可。

$$J(w)=\frac{\left|\tilde{\mu}_1-\tilde{\mu}_2\right|^2}{\tilde{s}_1^{\,2}+\tilde{s}_2^{\,2}}$$

在公式推导过程中，牢记最终的要求依旧是寻找最合适的投影方向，先把散列值公式展开：

$$\begin{aligned}\tilde{S}_t^{\,2}&=\sum_{y\in Y_i}(y-\mu_i)^2=\sum_{y\in Y_i}(w^{\mathrm{T}}x-w^{\mathrm{T}}\mu_i)^2\\&=\sum\nolimits_{y\in Y_i}w^{\mathrm{T}}(x-\mu_i)(x-\mu_i)^{\mathrm{T}}w\end{aligned}\qquad(15.7)$$

为了化简方便，则令：

$$S_i=\sum\nolimits_{x\in X_i}(x-\mu_i)(x-\mu_i)^{\mathrm{T}}\qquad(15.8)$$

式（15.8）称为散布矩阵（scatter matrices）。

由此可以定义类内散布矩阵为：

$$S_w=S_1+S_2\qquad(15.9)$$

将式（15.9）代入式（15.7），可得：

$$\tilde{s}_1^2+\tilde{s}_2^2=w^{\mathrm{T}}S_ww\qquad(15.10)$$

对式（15.6）分子进行展开可得：

$$(\tilde{\mu}_1-\tilde{\mu}_2)^2=(w^{\mathrm{T}}\mu_1-w^{\mathrm{T}}\mu_2)^2=w^{\mathrm{T}}(\mu_1-\mu_2)(\mu_1-\mu_2)^{\mathrm{T}}w=w^{\mathrm{T}}S_Bw\qquad(15.11)$$

其中，SB为类间散布矩阵。

目标函数$J(w)$最终可以表示为：

$$J(w)=\frac{w^{\mathrm{T}}S_Bw}{w^{\mathrm{T}}S_ww} \tag{15.12}$$

对于散列矩阵SB和SW，只要有数据和标签即可求解。但是如何求解最终的结果呢？如果对分子和分母同时求解，就会有无穷多解，通用的解决方案是先固定分母，经过放缩变换后，将其值限定为1，则令：

$$w^{\mathrm{T}}S_ww=1 \tag{15.13}$$

在此条件下求$w^{\mathrm{T}}S_Bw$的极大值点，利用拉格朗日乘子法可得：

$$\begin{aligned} c(w)&=w^{\mathrm{T}}S_Bw-\lambda(w^{\mathrm{T}}S_ww-1)\\ \Rightarrow\frac{\mathrm{d}c}{\mathrm{d}w}&=2S_Bw-2\lambda S_ww=0\\ \Rightarrow S_Bw&=\lambda S_ww \end{aligned} \tag{15.14}$$

既然目标是找投影方向，也就是w，将式（15.14）左右两边同时乘以S_w^{-1}可得：

$$S_w^{-1}S_Bw=\lambda w \tag{15.15}$$

观察一下式（15.15），它与线性代数中的特征向量有点像，如果把$S_w^{-1}S_B$当作一个整体，那么w就是其特征向量，问题到此迎刃而解。在线性判别分析中，其实只需要得到类内和类间散布矩阵，然后求其特征向量，就可以得到投影方向，然后，只需要对数据执行相应的矩阵变换，就完成全部降维操作。

15.1.4 Python实现线性判别分析降维

要在非常经典的“鸢尾花”数据集上使用线性判别分析完成降维任务。鸢尾花数据集可从该链接 下载:https://archive.ics.uci.edu/ml/datasets/Iris (https://archive.ics.uci.edu/ml/datasets/Iris) 也可以从sklearn库内置的数据集中获取。数据集中含有3 类、共150条鸢尾花基本数据，其中山鸢尾、变色鸢尾、维吉尼亚鸢尾各有50条数据，每条数据包括萼片 长度(单位:厘米)、萼片宽度、花瓣长度、花瓣宽度4种特征。

首先读取数据集，代码如下:

In [1]:

```
# 自己来定义列名
feature_dict = {i:label for i,label in zip(
                range(4),
                  ('sepal length in cm',
                  'sepal width in cm',
                  'petal length in cm',
                  'petal width in cm', ))}

label_dict = {i:label for i,label in zip(
                range(1,4),
                  ('Setosa',
                  'Versicolor',
                  'Virginica'
                  ))}

import pandas as pd
# 数据读取，大家也可以先下载下来直接读取
df = pd.io.parsers.read_csv(
    filepath_or_buffer='https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',
    header=None,
    sep=',',
    )
# 指定列名
df.columns = [l for i,l in sorted(feature_dict.items())] + ['class label']

df.head()
```

Out[1]:

	sepal length in cm	sepal width in cm	petal length in cm	petal width in cm	class label
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

数据集共有150条数据，每条数据有4个特征，现在需要将四维特征降成二维。观察输出结果可以发 现，其特征已经是数值数据，不需要做额外处理，但是需要转换一下标签:

In [2]:

```
from sklearn.preprocessing import LabelEncoder

X = df[['sepal length in cm','sepal width in cm','petal length in cm','petal width in cm']].values
y = df['class label'].values

# 制作标签{1: 'Setosa', 2: 'Versicolor', 3:'Virginica'}
enc = LabelEncoder()
label_encoder = enc.fit(y)
y = label_encoder.transform(y) + 1
```

上述代码使用了sklearn工具包中的LabelEncoder用于快速完成标签转换，可以发现基本上所有sklearn 中的数据处理操作都是分两步走，先fit再transform，变换结果如图15-4所示。

$$y = \begin{bmatrix} \text{setosa} \\ \text{setosa} \\ \cdots \\ \text{virginica} \end{bmatrix} \Rightarrow \begin{bmatrix} 1 \\ 1 \\ \cdots \\ 3 \end{bmatrix}$$

在计算过程中需要基于均值来判断距离，因此先要对数据中各个特征求均值，但是只求4个特征的均 值能满足要求吗?不要忘记任务中还有3种花，相当于3个类别，所以也要对每种花分别求其各个特征的均值:

$$m_i = \begin{bmatrix} \mu_{wi(\text{sepal length})} \\ \mu_{wi(\text{sepal width})} \\ \mu_{wi(\text{petal length})} \\ \mu_{wi(\text{petal width})} \end{bmatrix}, \text{ with } i = 1, 2, 3$$

In [3]:

```
import numpy as np
#设置小数点的位数
np.set_printoptions(precision=4)
#这里会保存所有的均值
mean_vectors = []
# 要计算3个类别
for cl in range(1,4):
    # 求当前类别各个特征均值
    mean_vectors.append(np.mean(X[y==cl], axis=0))
    print('均值类别 %s: %s\n' %(cl, mean_vectors[cl-1]))
```

```
均值类别 1: [5.006 3.418 1.464 0.244]

均值类别 2: [5.936 2.77  4.26  1.326]

均值类别 3: [6.588 2.974 5.552 2.026]
```

接下来计算类内散布矩阵:

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

$$S_i = \sum_{z \in D_i}^{n} (x - m_i)(x - m_i)^{\mathrm{T}}$$

$$m_i = \frac{1}{n_i} \sum_{x \in D_i}^{n} x_k$$

In [4]:

```
# 原始数据中有4个特征
S_W = np.zeros((4,4))
# 要考虑不同类别，自己算自己的
for cl,mv in zip(range(1,4), mean_vectors):
    class_sc_mat = np.zeros((4,4))
    # 选中属于当前类别的数据
    for row in X[y == cl]:
        # 这里相当于对各个特征分别进行计算，用矩阵的形式
        row, mv = row.reshape(4,1), mv.reshape(4,1)
        # 跟公式一样
        class_sc_mat += (row-mv).dot((row-mv).T)
    S_W += class_sc_mat
print('类内散布矩阵:\n', S_W)
```

类内散布矩阵:

```
 [[38.9562 13.683  24.614   5.6556]
 [13.683  17.035   8.12    4.9132]
 [24.614   8.12   27.22    6.2536]
 [ 5.6556  4.9132  6.2536  6.1756]]
```

继续计算类间散布矩阵:

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

式中，m为全局均值;mi为各个类别的均值;Ni为样本个数。

In [5]:

```
# 全局均值
overall_mean = np.mean(X, axis=0)
# 构建类间散布矩阵
S_B = np.zeros((4,4))
# 对各个类别进行计算
for i,mean_vec in enumerate(mean_vectors):
    #当前类别的样本数
    n = X[y==i+1,:].shape[0]
    mean_vec = mean_vec.reshape(4,1)
    overall_mean = overall_mean.reshape(4,1)
    # 如上述公式进行计算
    S_B += n * (mean_vec - overall_mean).dot((mean_vec - overall_mean).T)

print('类间散布矩阵:\n', S_B)
```

```
类间散布矩阵:
 [[ 63.2121 -19.534  165.1647  71.3631]
 [-19.534   10.9776 -56.0552 -22.4924]
 [165.1647 -56.0552 436.6437 186.9081]
 [ 71.3631 -22.4924 186.9081  80.6041]]
```

得到类内和类间散布矩阵后，还需将它们组合在一起，然后求解矩阵的特征向量:

求解矩阵的特征值：$S_W^{-1}S_B$

In [6]:

```
#求解矩阵特征值，特征向量
eig_vals, eig_vecs = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))
# 拿到每一个特征值和其所对应的特征向量
for i in range(len(eig_vals)):
    eigvec_sc = eig_vecs[:,i].reshape(4,1)
    print('\n特征向量 {}: \n{}'.format(i+1, eigvec_sc.real))
    print('特征值 {:}: {:.2e}'.format(i+1, eig_vals[i].real))
```

```
特征向量 1:
[[ 0.2049]
 [ 0.3871]
 [-0.5465]
 [-0.7138]]
特征值 1: 3.23e+01

特征向量 2:
[[-0.009 ]
 [-0.589 ]
 [ 0.2543]
 [-0.767 ]]
特征值 2: 2.78e-01

特征向量 3:
[[-0.8379]
 [ 0.1696]
 [ 0.1229]
 [ 0.5041]]
特征值 3: -4.13e-15

特征向量 4:
[[ 0.2   ]
 [-0.3949]
 [-0.4567]
 [ 0.7717]]
特征值 4: 1.20e-14
```

输出结果得到4个特征值和其所对应的特征向量。特征向量直接观察起来比较麻烦，因为投影方向在 高维上很难理解;特征值还是比较直观的，这里可以认为特征值代表的是其所对应特征向量的重要程 度，也就是特征值越大，其所对应的特征向量就越重要，所以接下来可以对特征值按大小进行排序，排 在前面的越重要，排在后面的就没那么重要了。

In [7]:

```
#特征值和特征向量配对
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

# 按特征值大小进行排序
eig_pairs = sorted(eig_pairs, key=lambda k: k[0], reverse=True)

print('特征值排序结果:\n')
for i in eig_pairs:
    print(i[0])
```

特征值排序结果:

```
32.27195779972981
0.27756686384004264
1.1953730364935478e-14
4.1311796919088535e-15
```

In [8]:

```
print('特征值占总体百分比:\n')
eigv_sum = sum(eig_vals)
for i,j in enumerate(eig_pairs):
    print('特征值 {0:}: {1:.2%}'.format(i+1, (j[0]/eigv_sum).real))
```

特征值占总体百分比:

```
特征值 1: 99.15%
特征值 2: 0.85%
特征值 3: 0.00%
特征值 4: 0.00%
```

可以看出，打印出来的结果差异很大，第一个特征值占据总体的99.15%，第二个特征值只占0.85%， 第三和第四个特征值，看起来微不足道。这表示对鸢尾花数据进行降维时，可以把特征数据降到二维甚 至一维，但没必要降到三维。

然已经有结论，选择把数据降到二维，只需选择特征值1、特征值2所对应的特征向量即可:

In [9]:

```
W = np.hstack((eig_pairs[0][1].reshape(4,1), eig_pairs[1][1].reshape(4,1)))
print('矩阵W:\n', W.real)
```

矩阵W:

```
 [[ 0.2049 -0.009 ]
 [ 0.3871 -0.589 ]
 [-0.5465  0.2543]
 [-0.7138 -0.767 ]]
```

这也是最终所需的投影方向，只需和原始数据组合，就可以得到降维结果:

In [10]:

```
# 执行降维操作
X_lda = X.dot(W)
X_lda.shape
```

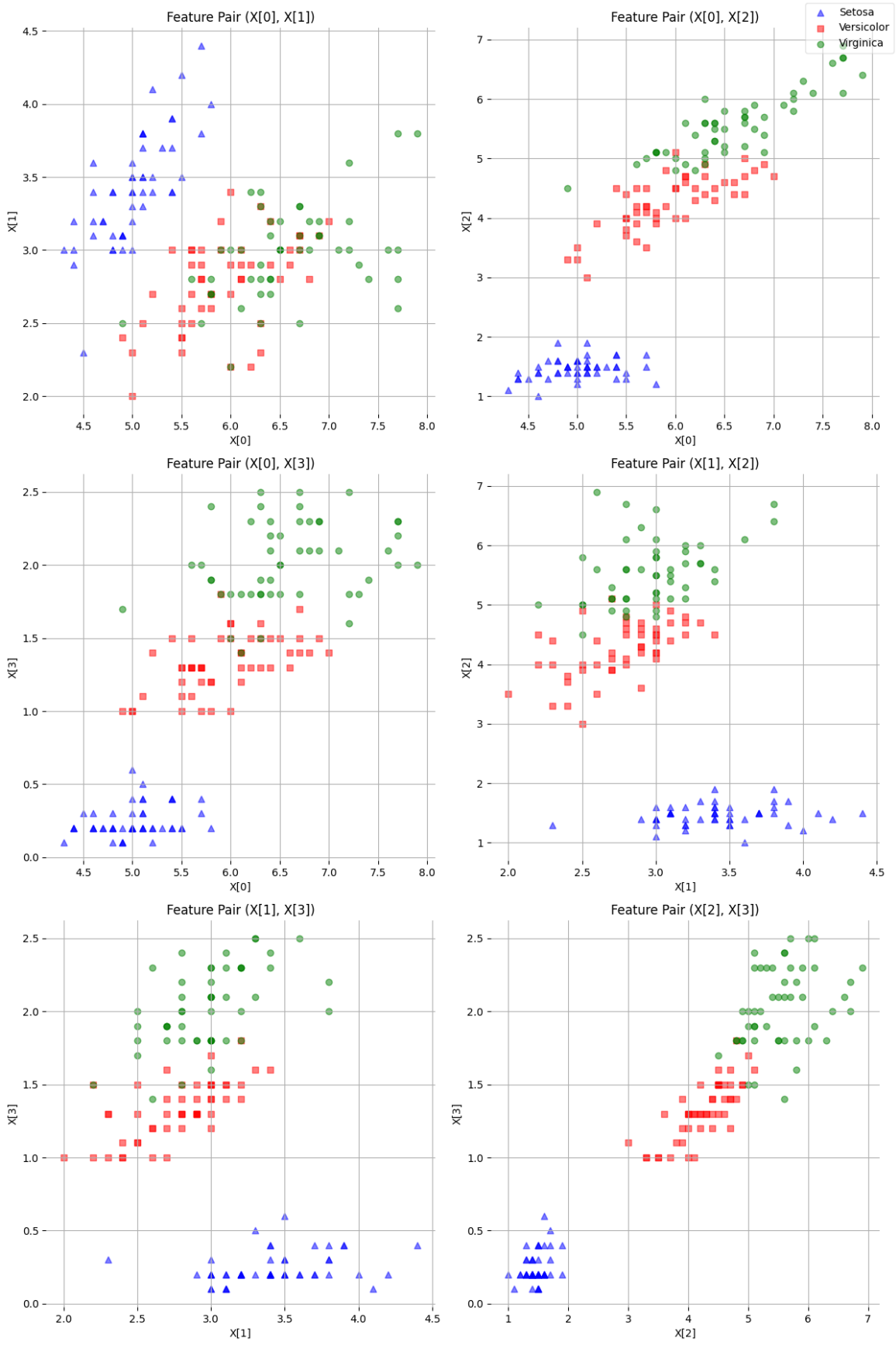
Out[10]:

```
(150, 2)
```

现在可以看到数据维度从原始的(150,4)降到(150,2)，到此就完成全部的降维工作。接下来对比 分析一下降维后结果，为了方便可视化展示，在原始四维数据集中随机选择两维进行绘图展示:

In [15]:

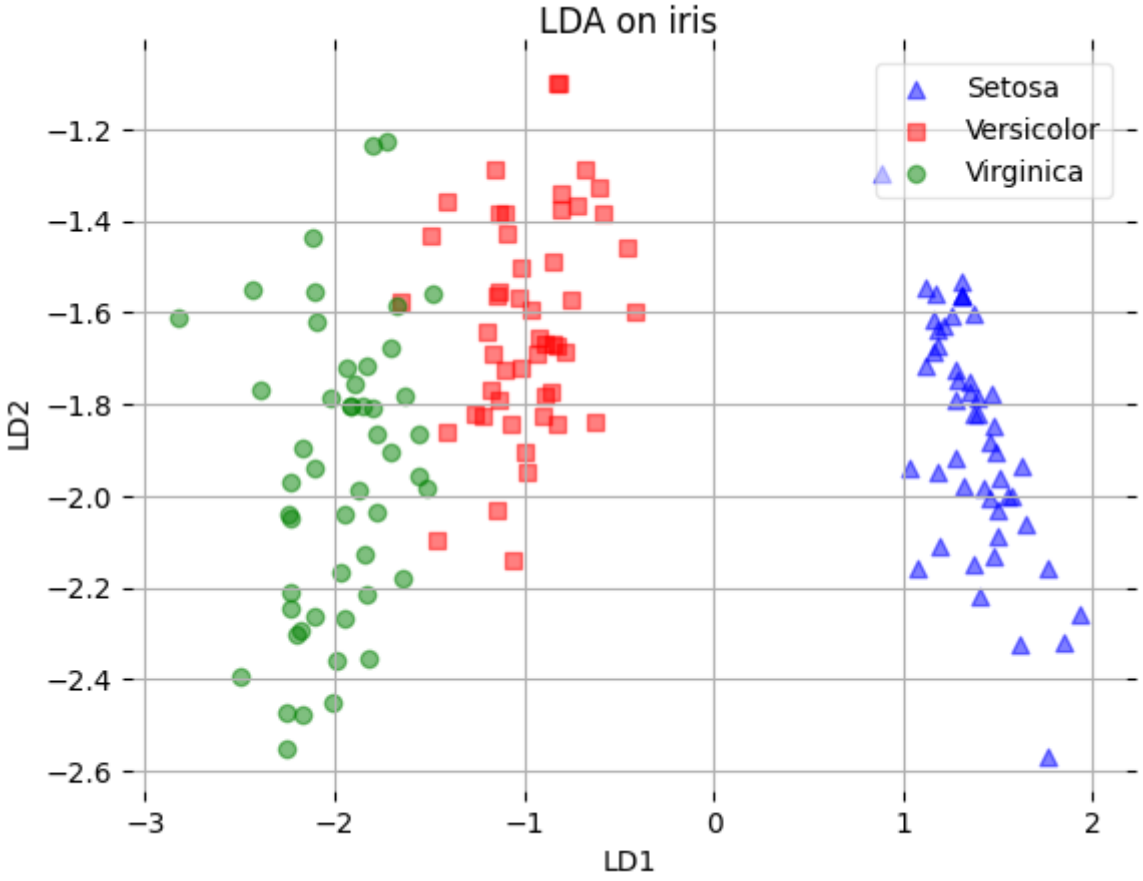
```
import matplotlib.pyplot as plt
import numpy as np

# 假设我们已加载鸢尾花数据集和处理好的特征矩阵 X 和标签 y
# 这里 X 是形状为 (150, 4) 的特征矩阵，y 是形状为 (150,) 的标签
# label_dict 是类别标签的字典，例如：{1: '山鸢尾', 2: '变色鸢尾', 3: '维吉尼亚鸢尾'}

# 可视化展示
def plot_step_lda(X, y, label_dict):
    feature_pairs = [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]  # 特征组合
    fig, axes = plt.subplots(3, 2, figsize=(12, 18))  # 创建 3x2 的子图
    axes = axes.ravel()  # 将子图数组展平，方便索引

    for i, (x_idx, y_idx) in enumerate(feature_pairs):
        ax = axes[i]
        for label, marker, color in zip(range(1, 4), ('^', 's', 'o'), ('blue',
'red', 'green')):
            ax.scatter(x=X[y == label, x_idx],
                       y=X[y == label, y_idx],
                       marker=marker,
                       color=color,
                       alpha=0.5,
                       label=label_dict[label])

        ax.set_xlabel(f'X[{x_idx}]')  # 设置 x 轴标签
        ax.set_ylabel(f'X[{y_idx}]')  # 设置 y 轴标签
        ax.set_title(f'Feature Pair (X[{x_idx}], X[{y_idx}])')

        # 隐藏图的边框线
        ax.spines["top"].set_visible(False)
        ax.spines["right"].set_visible(False)
        ax.spines["bottom"].set_visible(False)
        ax.spines["left"].set_visible(False)

        ax.grid()

    # 添加图例到最后一个子图中
    handles, labels = ax.get_legend_handles_labels()
    fig.legend(handles, labels, loc='upper right', fancybox=True, framealpha=0.
5)

    plt.tight_layout()
    plt.show()

# 调用函数来绘制图像
# 假设 X 是你的特征矩阵，y 是类别标签，label_dict 是类别标签的字典
plot_step_lda(X, y, label_dict)
```



从上述输出结果可以发现，如果对原始数据集随机取两维数据，数据集并不能按类别划分开，很多 数据都堆叠在一起(尤其是图中方块和圆形数据点)。再来看看降维后的数据点分布，绘图代码保持不 变，只需要传入降维后的两维数据即可，可以生成图15-5的输出。

In [12]:

```
from matplotlib import pyplot as plt
# 可视化展示
def plot_step_lda():

    ax = plt.subplot(111)
    for label,marker,color in zip(
        range(1,4),('^', 's', 'o'),('blue', 'red', 'green')):

        plt.scatter(x=X_lda[:,0].real[y == label],
                y=X_lda[:,1].real[y == label],
                marker=marker,
                color=color,
                alpha=0.5,
                label=label_dict[label]
                )

    plt.xlabel('LD1')
    plt.ylabel('LD2')

    leg = plt.legend(loc='upper right', fancybox=True)
    leg.get_frame().set_alpha(0.5)
    plt.title('LDA on iris')

    # 把边边角角隐藏起来
    plt.tick_params(axis="both", which="both", bottom="off", top="off",
            labelbottom="on", left="off", right="off", labelleft="on")

    # 为了看的清晰些，尽量简洁
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.spines["bottom"].set_visible(False)
    ax.spines["left"].set_visible(False)

    plt.grid()
    plt.tight_layout
    plt.show()

plot_step_lda()
```



可以明显看到，坐标轴变成LD1与LD2，这就是降维后的结果，从数据点的分布来看，混杂在一起的数据不多，划分起来就更容易。这就是经过一步步计算得到的最终降维结果。

迪哥说:当拿到一份规模较大的数据集时，如何选定降维的维数呢?一方面可以通过观察特征值排 序结果来决定，另一方面还是要通过实验来进行交叉验证。

我们肯定希望用更高效、稳定的方法来完成一个实际任务，再来看看sklearn工具包中如何调用线性判别分析进行降维:

In [12]:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

# 调用包实现LDA
sklearn_lda = LDA(n_components=2)
X_lda_sklearn = sklearn_lda.fit_transform(X, y)
```

In [13]:

```
X_lda_sklearn.shape
```

Out[13]:

```
(150, 2)
```

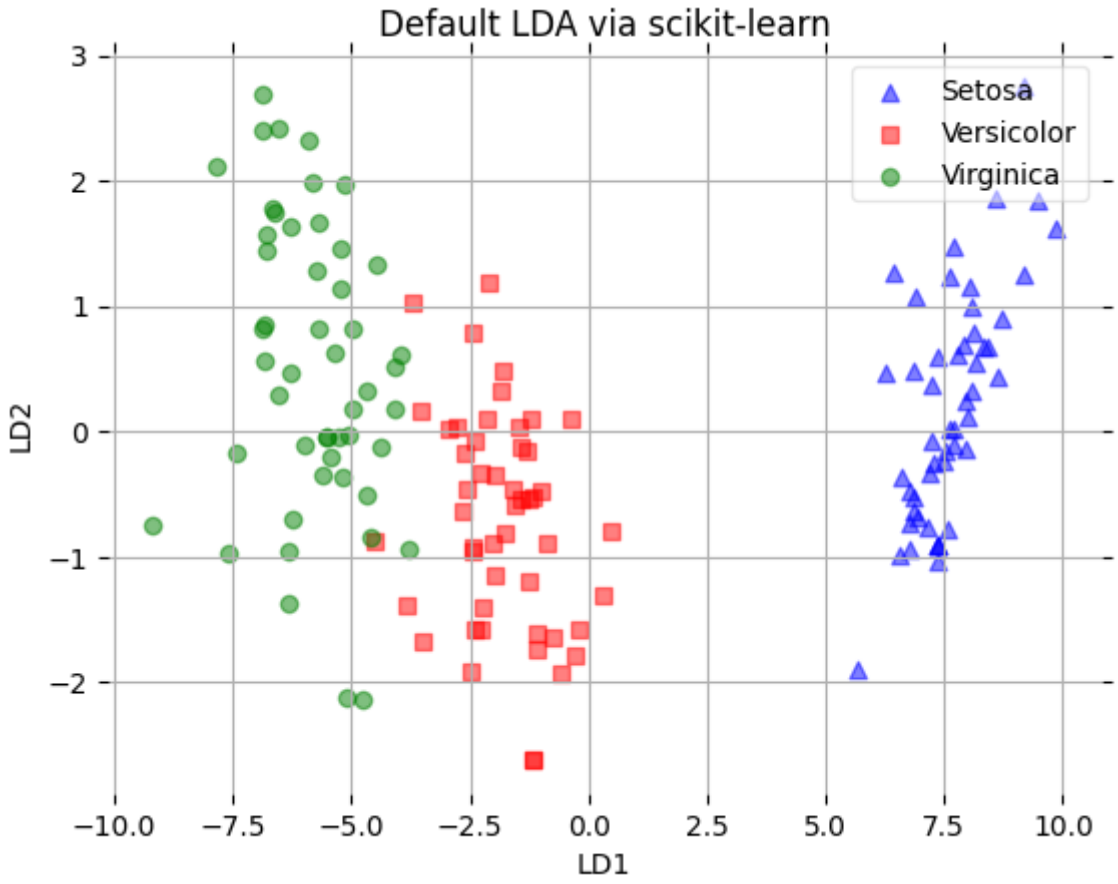
In [14]:

```
# 查看sklearn算出来的特征向量和特征值

# 查看特征值
print("特征值（explained_variance_ratio_）：", sklearn_lda.explained_variance_rati
o_)

# 查看特征向量
print("特征向量（scalings_）：", sklearn_lda.scalings_)
```

```
特征值（explained_variance_ratio_）： [0.99147248 0.00852752]
特征向量（scalings_）： [[ 0.81926852 -0.03285975]
 [ 1.5478732  -2.15471106]
 [-2.18494056  0.93024679]
 [-2.85385002 -2.8060046 ]]
```

In [16]:

```
# 绘图
def plot_scikit_lda(X, title):

    ax = plt.subplot(111)
    for label,marker,color in zip(
        range(1,4),('^', 's', 'o'),('blue', 'red', 'green')):

        plt.scatter(x=X[:,0][y == label],
                    y=X[:,1][y == label] * -1, # flip the figure
                    marker=marker,
                    color=color,
                    alpha=0.5,
                    label=label_dict[label])

    plt.xlabel('LD1')
    plt.ylabel('LD2')

    leg = plt.legend(loc='upper right', fancybox=True)
    leg.get_frame().set_alpha(0.5)
    plt.title(title)

    # hide axis ticks
    plt.tick_params(axis="both", which="both", bottom="off", top="off",
            labelbottom="on", left="off", right="off", labelleft="on")

    # remove axis spines
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.spines["bottom"].set_visible(False)
    ax.spines["left"].set_visible(False)

    plt.grid()
    plt.tight_layout
    plt.show()

plot_scikit_lda(X_lda_sklearn, title='Default LDA via scikit-learn')
```



可以发现，使用sklearn工具包降维后的结果与自己一步步计算的结果完全一致。

测试降到1维

In [44]:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

# 调用包实现LDA并降到1维
sklearn_lda1 = LDA(n_components=1)
X_lda_sklearn1 = sklearn_lda1.fit_transform(X, y)  # 自动降到1维

# 查看降维后的数据形状
print("降维后的数据形状: ", X_lda_sklearn1.shape)  # 应该是 (150, 1)

# 查看特征值（解释方差的比例）
print("特征值 (explained_variance_ratio_) : ", sklearn_lda1.explained_variance_rat
io_)

# 查看特征向量（scalings_）
print("特征向量 (scalings_) : ", sklearn_lda1.scalings_)
```

```
降维后的数据形状:  (150, 1)
特征值 (explained_variance_ratio_) :  [0.9915]
特征向量 (scalings_) :  [[ 0.8193 -0.0329]
 [ 1.5479 -2.1547]
 [-2.1849  0.9302]
 [-2.8539 -2.806 ]]
```

In [50]:

```
# 获取第一个线性判别方向的投影向量（第1列）
w_first_ld = sklearn_lda1.scalings_[:, 0]

# 将数据投影到第一个线性判别方向
X_lda1 = X.dot(w_first_ld)

# 查看手动降维后的数据形状
print("手动降维后的数据形状：", X_lda_manual.shape)  # 应该是 (150,)
```

手动降维后的数据形状： (150,)

In [51]:

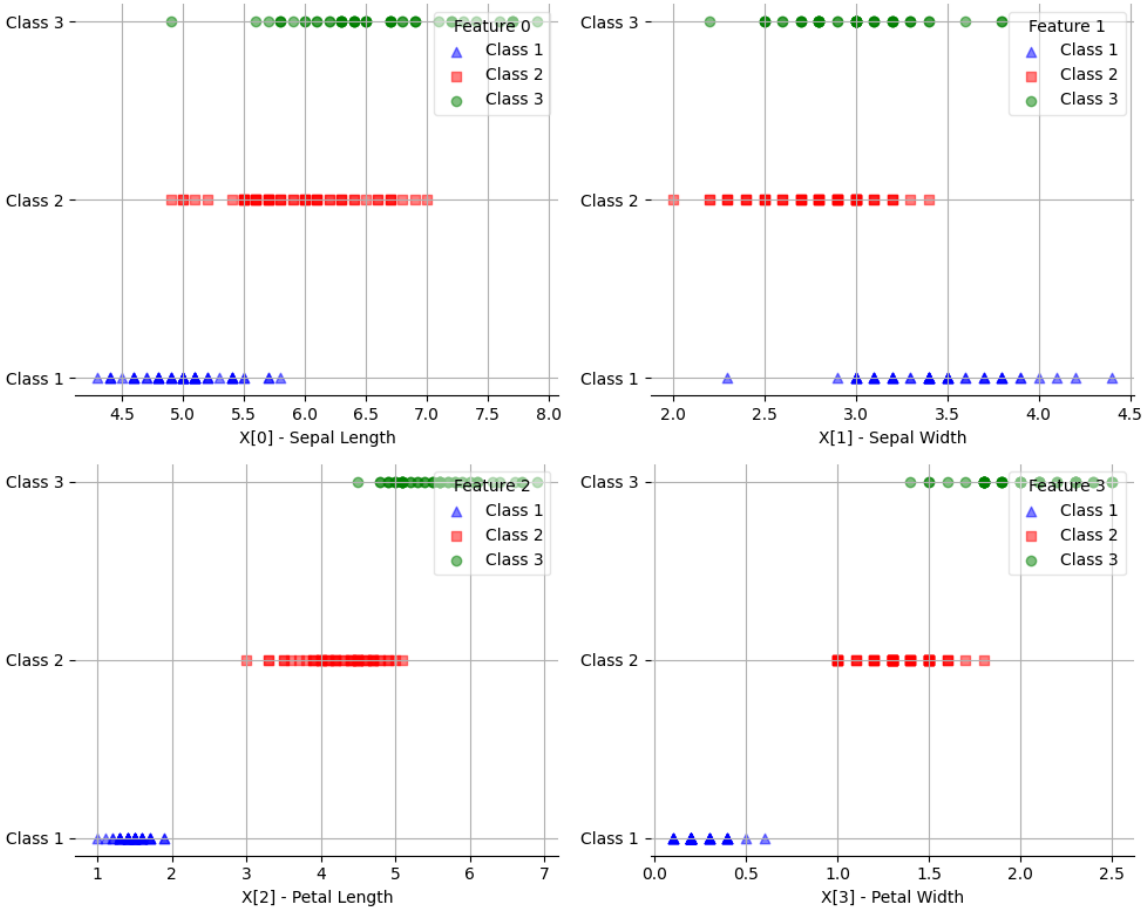
```
X_lda1
```

Out[51]:

```
array([  5.9661,   5.0283,   5.3926,   4.7189,   6.039 ,   5.6048,
         5.1163,   5.5109,   4.4639,   5.25  ,   6.303 ,   5.1286,
         5.2318,   5.4777,   7.7506,   7.0615,   6.4788,   5.6808,
         5.9812,   5.9266,   5.4016,   5.4864,   6.5853,   4.1449,
         4.4731,   4.6733,   4.7217,   5.8296,   5.8933,   4.7371,
         4.6642,   5.2679,   7.0437,   7.3774,   5.25  ,   5.8568,
         6.5123,   5.25  ,   4.8372,   5.5929,   5.8173,   3.5502,
         5.1468,   4.3057,   4.7673,   4.661 ,   5.9935,   5.0921,
         6.2211,   5.5746,  -3.5765,  -3.9165,  -4.5356,  -4.3837,
        -4.6722,  -4.5384,  -4.566 ,  -2.3348,  -3.8647,  -4.0772,
        -3.3091,  -3.9802,  -3.2727,  -4.7782,  -2.4991,  -3.3216,
        -4.8815,  -2.8811,  -5.6282,  -3.2029,  -5.8378,  -3.1182,
        -5.9559,  -4.3623,  -3.3731,  -3.5583,  -4.578 ,  -5.6435,
        -4.7086,  -1.8068,  -3.2211,  -2.7173,  -3.0149,  -6.6145,
        -5.0453,  -4.22  ,  -4.2625,  -4.6022,  -3.4367,  -4.0741,
        -4.5079,  -4.405 ,  -3.3882,  -2.4077,  -4.1196,  -3.2879,
        -3.7281,  -3.537 ,  -1.6461,  -3.6644,  -9.9749,  -7.6345,
        -8.4238,  -7.7224,  -8.9823,  -9.5436,  -6.7997,  -8.4326,
        -8.4508,  -8.9917,  -6.5725,  -7.5799,  -7.7956,  -8.0929,
        -8.9066,  -7.9475,  -7.1852,  -8.7273, -11.3071,  -6.8845,
        -8.4119,  -7.492 ,  -9.7044,  -6.5025,  -7.8502,  -7.3946,
        -6.2111,  -6.202 ,  -8.6514,  -6.6965,  -8.3538,  -7.3372,
        -8.9368,  -5.9285,  -7.209 ,  -8.94  ,  -8.6607,  -7.1124,
        -6.0654,  -7.3404,  -8.7974,  -7.2557,  -7.6345,  -8.9308,
        -8.9917,  -7.7928,  -7.3159,  -7.1005,  -8.0203,  -6.8028])
```

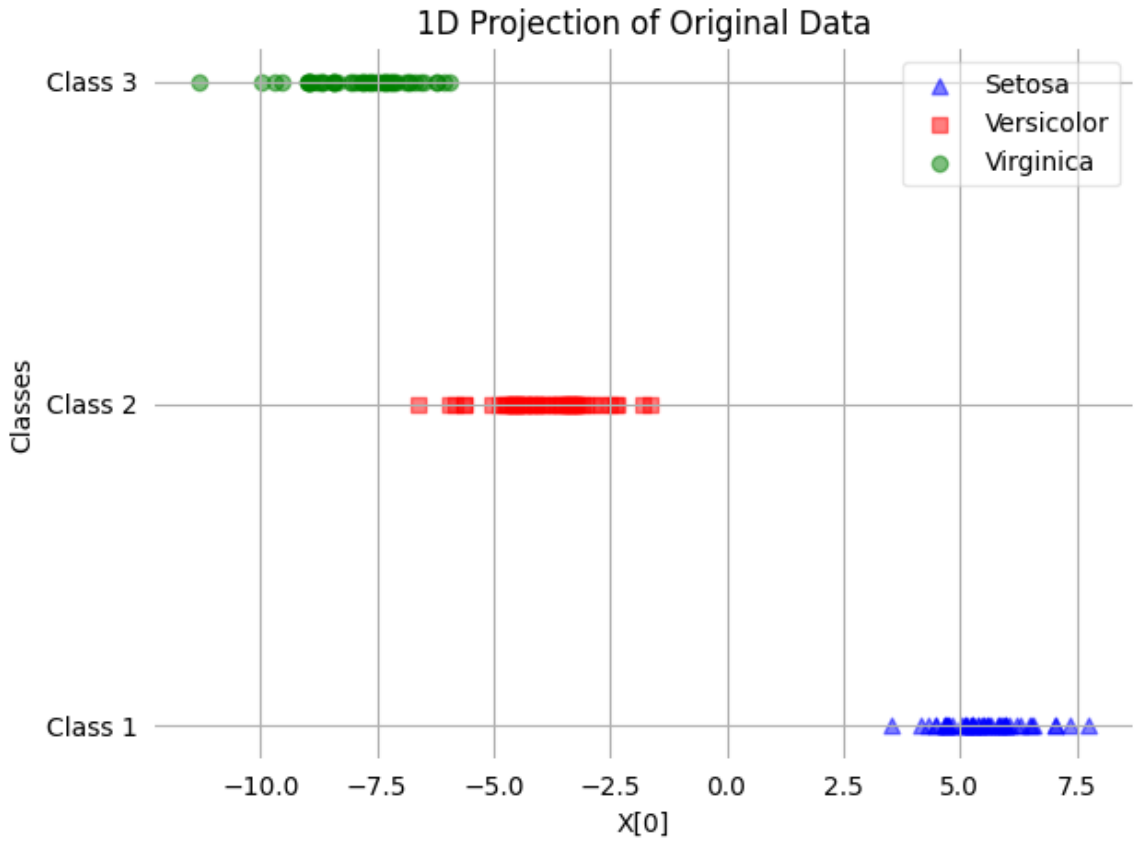
In [63]:

```
from matplotlib import pyplot as plt

# 可视化展示4个特征的一维数据（2x2布局）
def plot_step_lda():
    fig, axes = plt.subplots(2, 2, figsize=(10, 8))  # 2行2列子图布局
    feature_names = ['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Widt
h']  # 特征名称

    # 使用 enumerate 迭代4个特征
    for i, ax in enumerate(axes.ravel()):  # ravel() 将 2x2 的 axes 摊平成1维
        for idx, (label, marker, color) in enumerate(zip(
            range(1, 4), ('^', 's', 'o'), ('blue', 'red', 'green'))):

            # 绘制每个特征的一维散点图
            ax.scatter(x=X[:, i].real[y == label],  # 当前特征X[:, i]
                       y=[idx+1] * sum(y == label),  # 用类别的索引作为 y 轴
                       marker=marker,
                       color=color,
                       alpha=0.5,
                       label=f'Class {label}'  # 显示类别标签
                       )

        # 设置x轴为特征名称，y轴为类别
        ax.set_xlabel(f'X[{i}] - {feature_names[i]}')  # 显示特征序号 i 和名称
        ax.set_yticks([1, 2, 3])  # 设置类别索引为 y 轴刻度
        ax.set_yticklabels(['Class 1', 'Class 2', 'Class 3'])  # 设置y轴标签为类别
名称

        # 简洁化边框，隐藏多余的轴线
        ax.spines["top"].set_visible(False)
        ax.spines["right"].set_visible(False)
        ax.spines["bottom"].set_visible(True)
        ax.spines["left"].set_visible(False)
        ax.grid(True)

        # 在每个子图中显示特征序号 i 的图例
        ax.legend(title=f'Feature {i}', loc='upper right', fancybox=True, framea
lpha=0.5)

    plt.tight_layout()
    plt.show()

plot_step_lda()
```


Class 3
Class 2
Class 1
Feature 0
Class 1
Class 2
Class 3
4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
X[0] - Sepal Length
Feature 1
2.0 2.5 3.0 3.5 4.0 4.5
X[1] - Sepal Width
Feature 2
1 2 3 4 5 6 7
X[2] - Petal Length
Feature 3
0.0 0.5 1.0 1.5 2.0 2.5
X[3] - Petal Width

In [62]:

```
from matplotlib import pyplot as plt

# 可视化展示一维数据
def plot_step_lda():
    ax = plt.subplot(111)

    # 遍历每个类别并绘制一维散点图
    for idx, (label, marker, color) in enumerate(zip(
        range(1, 4), ('^', 's', 'o'), ('blue', 'red', 'green'))):

        plt.scatter(x=X_lda1[:].real[y == label],  # 只绘制 X[:, 0]
                    y=[idx+1] * sum(y == label),  # 用类别的索引作为 y 轴
                    marker=marker,
                    color=color,
                    alpha=0.5,
                    label=label_dict[label]
                    )

    # 设置 x 轴和 y 轴的标签
    plt.xlabel('X[0]')
    plt.yticks([1, 2, 3], ['Class 1', 'Class 2', 'Class 3'])  # 设置 y 轴标签为类别
    plt.ylabel('Classes')

    # 添加图例
    leg = plt.legend(loc='upper right', fancybox=True)
    leg.get_frame().set_alpha(0.5)
    plt.title('1D Projection of Original Data')

    # 隐藏图的边框
    plt.tick_params(axis="both", which="both", bottom=False, top=False,
                    labelbottom=True, left=False, right=False, labelleft=True)

    # 隐藏顶部和右侧坐标轴
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.spines["bottom"].set_visible(False)
    ax.spines["left"].set_visible(False)

    # 显示网格
    plt.grid()
    plt.tight_layout()
    plt.show()

plot_step_lda()
```


1D Projection of Original Data
Class 3
Class 2
Class 1
Classes
Setosa
Versicolor
Virginica
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
X[0]

In [ ]: