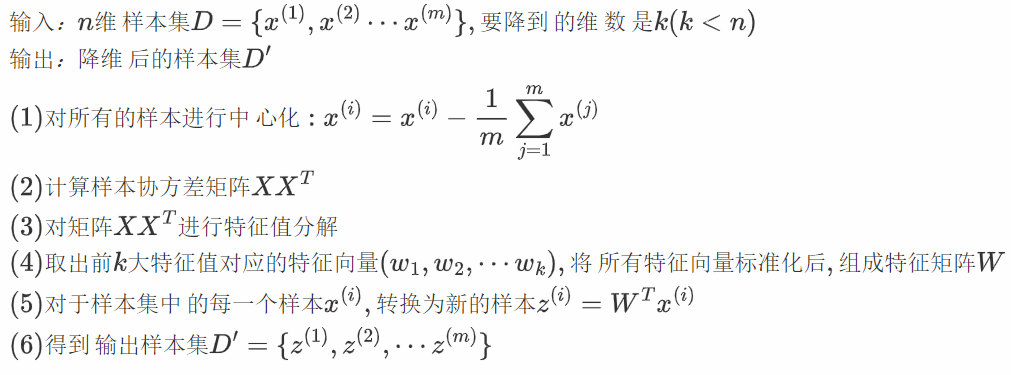
实验7 PCA和LDA降维算法

1. 实验思路：

算法基本思想：PCA的主要思想就是找出数据里最主要的方面，用数据里最主要的方面来代替原始数据。假设要将原来的n维数据降到m维(m<n)，那我们肯定希望这m维能尽可能的代替原来的n维数据；我们知道降维肯定会带来损失，PCA的目的就是尽可能的降低这种损失，最大可能的保留原来的数据的信息。

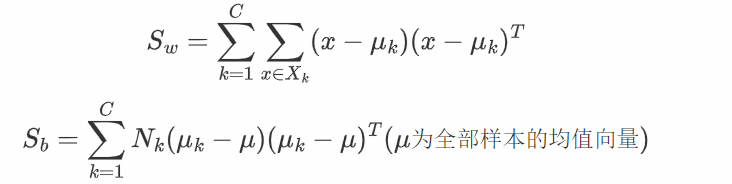
算法流程：



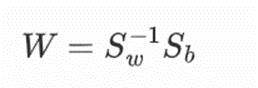
LDA降维思路：LDA降维思路与Fisher线性判别算法思路相同，即使得“投影后类内方差最小，类间方差最大”。我们要将数据在低维度上进行投影，投影后希望每一种类别数据的投影点尽可能的接近，而不同类别的数据的类别中心之间的距离尽可能的大。

算法流程：

1)计算类内和类间散度矩阵



2)计算投影矩阵W(通过选取前S\_w前k个特征向量组成投影到k维矩阵):



PCA与LDA降维区别：

1)LDA是有监督的降维方法，而PCA是无监督的降维方法;

2)LDA降维最多降到类别数k-1的维数，而PCA没有这个限制;

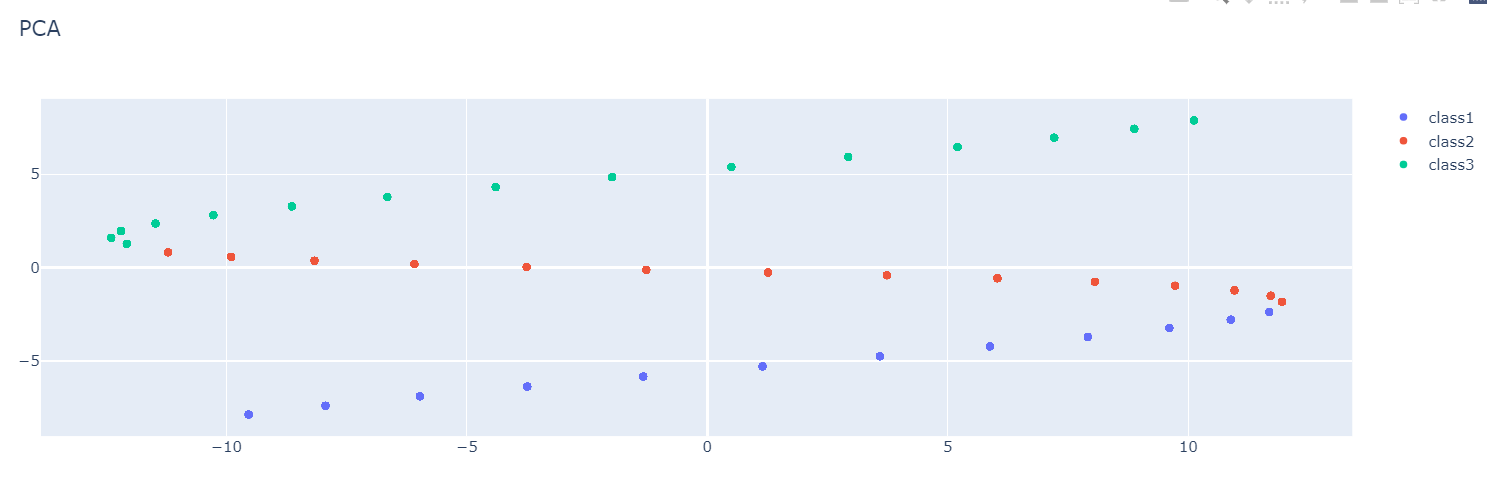
3)LDA除了可以用于降维，还可以用于分类;

4)LDA选择分类性能最好的投影方向，而PCA选择样本点投影具有最大方差的方向。

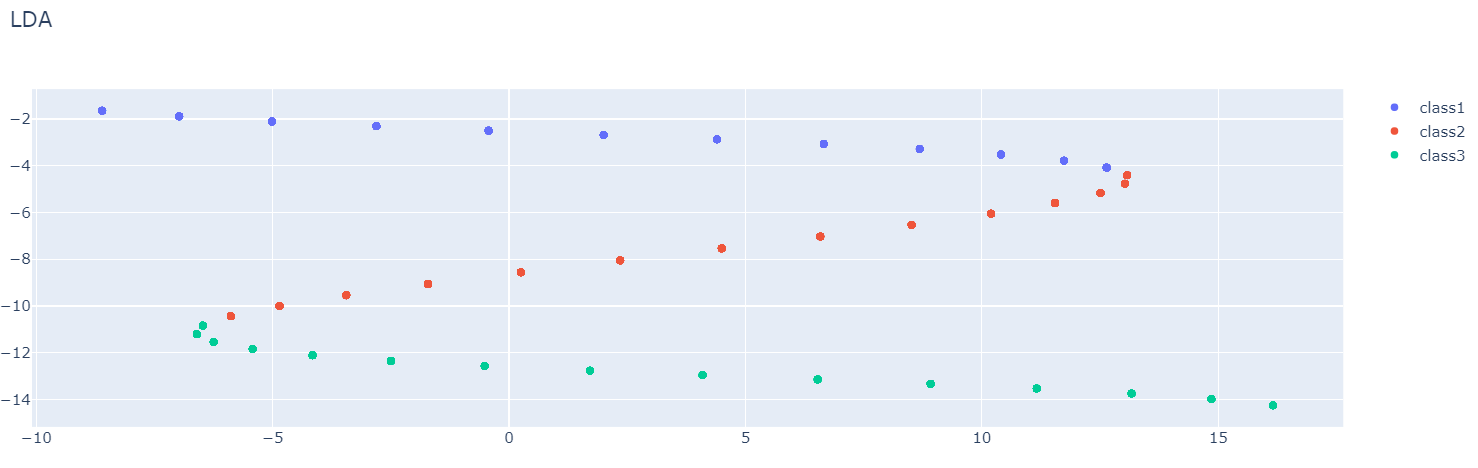
实验过程：

1. 使用pandas导入数据集
2. 编写PCA与LDA
3. 实例化PCA与LDA并将数据降到2维
4. 使用plotly对降维后的数据进行可视化
5. 实验结果

使用PCA将数据由三维降维到二维后数据分布图：



使用LDA将数据由三维降到二维后数据分布图:



1. 源代码(Python)

import numpy as np

import pandas as pd

import plotly.graph\_objects as go

# 读取数据集

df = pd.read\_csv("swiss-data.txt", header=None, sep='\s+')

df[0] = df[0].values.astype(np.int32)

X, y = df[[1, 2, 3]].values, df[0].values

# 定义PCA模型

class PCA:

def \_\_init\_\_(self, n\_components):

self.n\_compoents =n\_components

def fit\_transform(self, X):

means = np.mean(X, axis=0)

X = X - means

# Covariance Matrix

covM = np.cov(X, rowvar=0)

eigval, eigvec = np.linalg.eig(covM)

indexes = np.argsort(eigval)[-self.n\_compoents:]

self.W = eigvec[:, indexes]

return np.dot(X, self.W)

# LDA

class LDA:

def \_\_init\_\_(self, priors=None, n\_components=None):

self.S\_w = None

self.S\_b = None

self.label = None

self.priors = priors

self.mu\_i = None # 每一类的均值向量

self.mu = None

self.w = None

self.n\_components = n\_components

def fit\_transform(self, X, y):

X, y = np.asarray(X), np.asarray(y)

n\_samples, n\_features = X.shape

assert n\_samples >= 2

# 计算先验概率

if self.priors is None:

self.priors = np.bincount(y) / n\_samples

# 获取类别个数

self.labels, yidx = np.unique(y, return\_inverse=True)

# 计算每个均值向量

means = np.zeros((len(self.labels), n\_features))

np.add.at(means, yidx, X)

self.mu\_i = means / np.expand\_dims(np.bincount(y), 1)

# 计算总体均值向量

self.mu = np.dot(np.expand\_dims(self.priors, axis=0), self.mu\_i)

# 计算类内散度矩阵

covMatrix = [np.cov(X[y == group].T) for group in self.labels]

self.S\_w = sum(covMatrix) / len(covMatrix)

# 计算类间散度矩阵

self.S\_b = sum([sum(y == group)\*np.dot((self.mu\_i[idx, None] - self.mu).T, (self.mu\_i[idx, None] - self.mu))

for idx, group in enumerate(self.labels)]) / (n\_samples - 1)

# 计算投影矩阵

# SVD求Sw的逆矩阵

U, Sigma, V = np.linalg.svd(self.S\_w)

Sigma\_inv = np.linalg.inv(np.diag(Sigma))

Sw\_inv = np.dot(np.dot(V.T, Sigma\_inv), U.T)

Sw\_inv\_Sb = np.dot(Sw\_inv, self.S\_b)

# 求特征值和特征向量，并取实数部分

la, vectors = np.linalg.eig(Sw\_inv\_Sb)

la = np.real(la)

vectors = np.real(vectors)

# 特征值的下标从大到小排列

laIdx = np.argsort(-la)

# 默认选取(N-1)个特征值的下标

if self.n\_components == None:

self.n\_components = len(self.labels)-1

# 选取特征值和向量

lambda\_index = laIdx[:self.n\_components]

w = vectors[:, lambda\_index]

self.w = w

return np.dot(X, self.w)

# 进行Pca降维

pca = PCA(2)

pca\_data = pca.fit\_transform(df.values)

# 结果绘制

graph = []

for i in range(3):

graph.append(go.Scatter(x=pca\_data[y == (i + 1), 1],

y=pca\_data[y == (i + 1), 0],

mode='markers', name=f'class{i+1}'))

fig = go.Figure(graph)

fig.update\_layout(title='PCA')

fig.show()

# 进行LDA降维

lda = LDA(n\_components=2)

lda\_X = lda.fit\_transform(X, y - 1)

# 结果绘制

graph = []

for i in range(3):

graph.append(go.Scatter(x=lda\_X[y == (i + 1), 1],

y=lda\_X[y == (i + 1), 0],

mode='markers', name=f'class{i+1}'))

fig = go.Figure(graph)

fig.update\_layout(title='LDA')

fig.show()