# 《模式识别》课程

# 实 验 报 告



**姓 名： 金家耀**

**专 业：**  人工智能

**学 号： 1193210320**

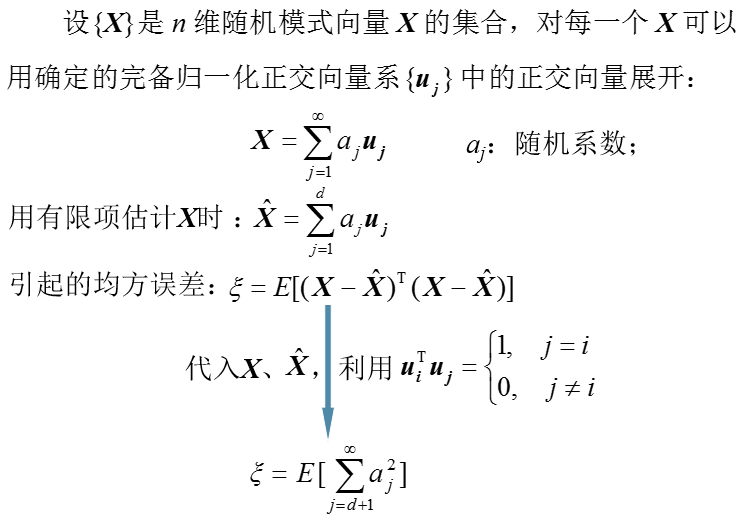
**江南大学人工智能与计算机学院**

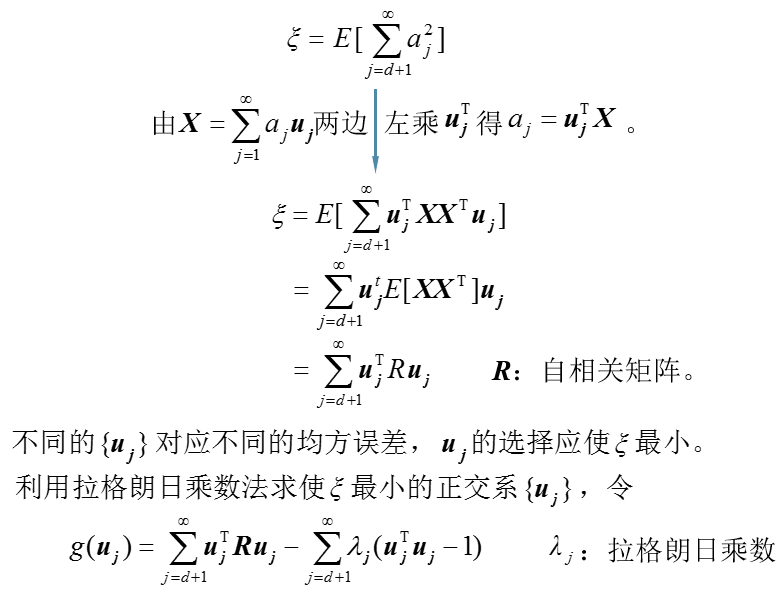
# K-L变换的多类模式特征提取

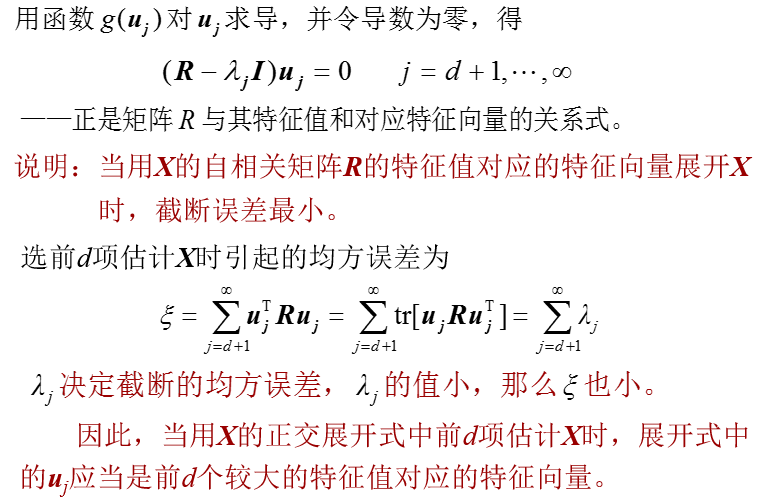
**1实验目的**

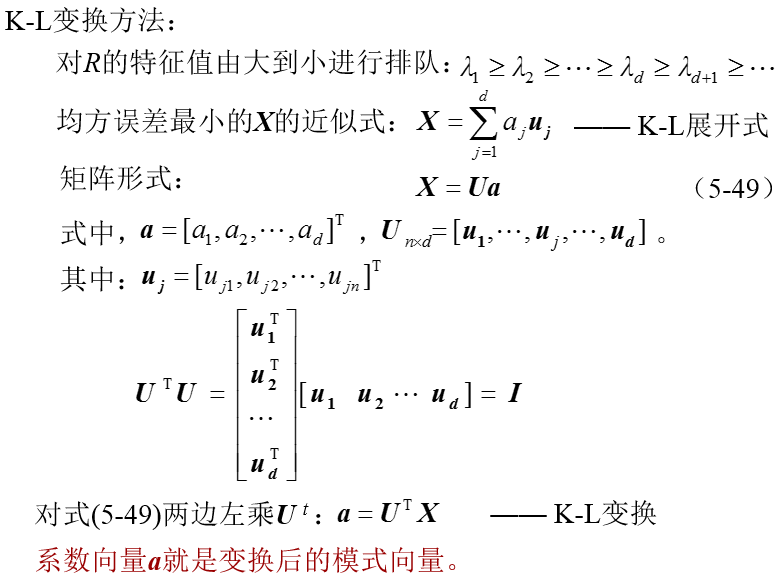
K-L变换以最小均方误差为准则进行数据压缩，是最小均方误差意义下的最优正交变换，是一种常用的特征提取方法，适用于任意的概率密度函数，在消除模式特征之间的相关性、突出差异性方面有最优的效果。本实验目的在于加深学生对K-L变换基本原理的理解，通过实际的数据处理掌握K-L变换的处理流程、验证K-L变换的性能，体会其在模式识别中的作用。

**2实验原理**

****

****

****

****

**3实验内容**

利用Iris数据集进行实验。该数据集以鸢尾花的特征作为数据来源，由3种不同类型的鸢尾花的50个样本数据构成。

样本特征（属性）：

Sepal.Length（花萼长度），单位是cm;

Sepal.Width（花萼宽度），单位是cm;

Petal.Length（花瓣长度），单位是cm;

Petal.Width（花瓣宽度），单位是cm;

种类：Iris Setosa（山鸢尾）、Iris Versicolour（杂色鸢尾）、Iris Virginica（维吉尼亚鸢尾）。

鸢尾花数据的特征维数是4，利用K-L变换进行特征提取并降维，结合最近邻法完成分类。

**4实验要求**

1. 利用最近邻法针对Iris原始特征进行分类（可随机选取80%的样本作为训练集，其余的作为测试集），要求显示分类结果和分类精度；
2. 先利用自相关矩阵作K-L变换进行特征提取并降维，再用最近邻法分类，显示分类结果和分类精度；
3. 分别采用类间散布矩阵和总体散布矩阵作K-L变换进行特征提取并降维，再用最近邻法分类，比较分类结果和分类精度；

**5实验代码和结果**

代码已上传至

<https://github.com/shinejjy/PatternRecognition/blob/master/sy5/K-L.ipynb>

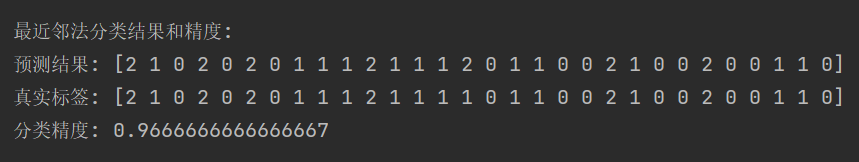
**5.1 数据准备**

首先，导入Iris数据集并分割成训练集和测试集。

from sklearn import datasets  
from sklearn.model\_selection import train\_test\_split  
  
iris = datasets.load\_iris()  
X = iris.data  
y = iris.target  
  
# 分割数据集  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

**5.2 利用最近邻法进行分类**

import numpy as np  
  
class KNNClassifier:  
 def \_\_init\_\_(self, k=3):  
 self.k = k  
 self.X\_train = None  
 self.y\_train = None  
  
 def fit(self, X\_train, y\_train):  
 self.X\_train = X\_train  
 self.y\_train = y\_train  
  
 def predict(self, X\_test):  
 y\_pred = []  
 for x in X\_test:  
 distances = np.sqrt(np.sum((self.X\_train - x) \*\* 2, axis=1))  
 nearest\_neighbors = np.argsort(distances)[:self.k]  
 nearest\_labels = self.y\_train[nearest\_neighbors]  
 most\_common\_label = np.argmax(np.bincount(nearest\_labels))  
 y\_pred.append(most\_common\_label)  
 return np.array(y\_pred)  
  
# 创建最近邻分类器  
knn\_classifier = KNNClassifier(k=3)  
  
# 使用训练集进行训练  
knn\_classifier.fit(X\_train, y\_train)  
  
# 预测测试集  
y\_pred = knn\_classifier.predict(X\_test)  
  
# 显示分类结果和分类精度  
accuracy = np.sum(y\_test == y\_pred) / len(y\_test)  
print("最近邻法分类结果和精度:")  
print("预测结果:", y\_pred)  
print("真实标签:", y\_test)  
print("分类精度:", accuracy)



直接利用样本原始特征进行KNN分类，得到最后精度为96.67%。

**5.3 K-L变换**

import numpy as np  
  
class KLDimensionalityReduction:  
 def \_\_init\_\_(self, n\_components):  
 self.n\_components = n\_components  
 self.mean\_vector = None  
 self.eigenvectors = None  
  
 def fit\_transform(self, X, y, scatter\_matrix\_type='class'):  
 # 计算均值向量  
 self.mean\_vector = np.mean(X, axis=0)  
  
 if scatter\_matrix\_type == 'class':  
 scatter\_matrix = self.\_compute\_class\_scatter\_matrix(X, y)  
 elif scatter\_matrix\_type == 'total':  
 scatter\_matrix = self.\_compute\_total\_scatter\_matrix(X)  
 elif scatter\_matrix\_type == 'auto':  
 scatter\_matrix = self.\_compute\_con\_matrix(X)  
 else:  
 raise ValueError("Invalid scatter\_matrix\_type. Use 'class', 'total', or 'auto'.")  
  
 # 计算特征值和特征向量  
 eigenvalues, eigenvectors = np.linalg.eig(scatter\_matrix)  
  
 # 排序特征值和对应的特征向量  
 sorted\_indices = np.argsort(eigenvalues)[::-1]  
 self.eigenvectors = eigenvectors[:, sorted\_indices[:self.n\_components]]  
  
 # 映射到新特征空间  
 X\_transformed = np.dot(X - self.mean\_vector, self.eigenvectors)  
  
 return X\_transformed  
  
 def \_compute\_class\_scatter\_matrix(self, X, y):  
 unique\_classes = np.unique(y)  
 class\_scatter\_matrix = np.zeros((X.shape[1], X.shape[1]))  
  
 for cls in unique\_classes:  
 X\_cls = X[y == cls]  
 mean\_vector\_cls = np.mean(X\_cls, axis=0)  
 class\_scatter\_matrix += len(X\_cls) \* np.outer((mean\_vector\_cls - self.mean\_vector), (mean\_vector\_cls - self.mean\_vector))  
  
 return class\_scatter\_matrix  
  
 def \_compute\_total\_scatter\_matrix(self, X):  
 total\_scatter\_matrix = np.cov((X - self.mean\_vector).T)  
 return total\_scatter\_matrix  
  
 def \_compute\_con\_matrix(self, X):  
 return np.dot(X.T, X)

这段代码实现了一个基于K-L变换的特征提取和降维的类 KLDimensionalityReduction。在该类中，通过计算不同散布矩阵类型（类内散布矩阵、总体散布矩阵、自相关矩阵）的特征值和特征向量，实现了对输入数据的降维操作。

首先，在初始化方法中，指定了目标降维的维度数 n\_components，并初始化了均值向量 mean\_vector 和特征向量 eigenvectors。

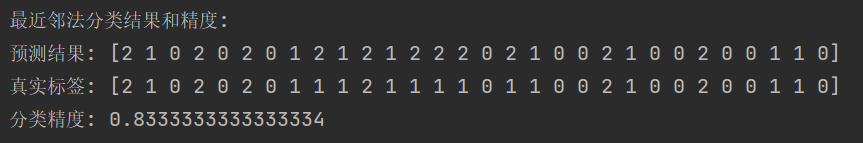
fit\_transform 方法是该类的核心功能，接受输入数据 X 和对应的标签 y，以及指定的散布矩阵类型。根据选择的散布矩阵类型，计算相应的散布矩阵，进行特征值分解以获取特征值和特征向量。随后，根据特征值的降序排列，选择前 n\_components 个特征向量。最终，利用选取的特征向量对原始数据进行线性变换，实现了降维操作。

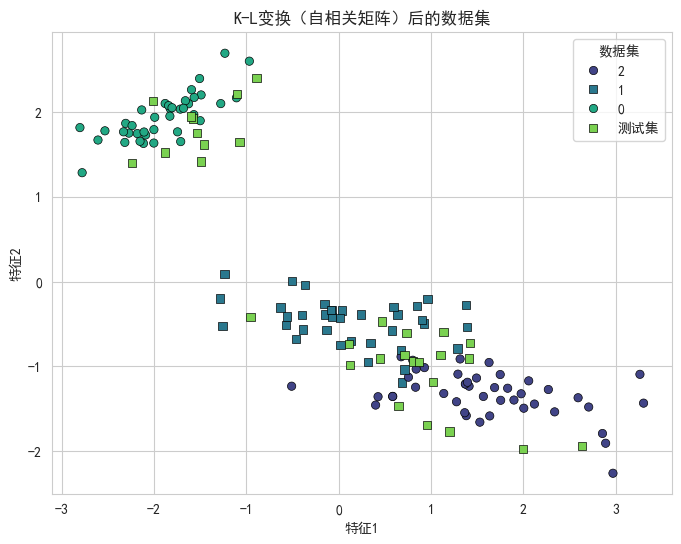
具体的散布矩阵计算方法分别包括：

* \_compute\_class\_scatter\_matrix 方法用于计算类内散布矩阵，根据不同类别内的样本均值向量计算加权和。
* \_compute\_total\_scatter\_matrix 方法计算总体散布矩阵，直接利用 numpy 的协方差矩阵函数。
* \_compute\_con\_matrix 方法计算自相关矩阵，使用矩阵乘法 np.dot 计算原始数据的转置与自身的乘积。

5.3.1 自相关矩阵

kld = KLDimensionalityReduction(n\_components=2)  
X\_train\_kld1 = kld.fit\_transform(X\_train, y\_train, scatter\_matrix\_type='auto')  
X\_test\_kld1 = kld.fit\_transform(X\_test, y\_test, scatter\_matrix\_type='auto')  
  
knn\_classifier = KNNClassifier(k=3)  
knn\_classifier.fit(X\_train\_kld1, y\_train)  
y\_pred = knn\_classifier.predict(X\_test\_kld1)  
  
# 显示分类结果和分类精度  
accuracy = np.sum(y\_test == y\_pred) / len(y\_test)  
print("最近邻法分类结果和精度:")  
print("预测结果:", y\_pred)  
print("真实标签:", y\_test)  
print("分类精度:", accuracy)  
  
accuracy\_dict['K-L变换（自相关矩阵）'] = accuracy

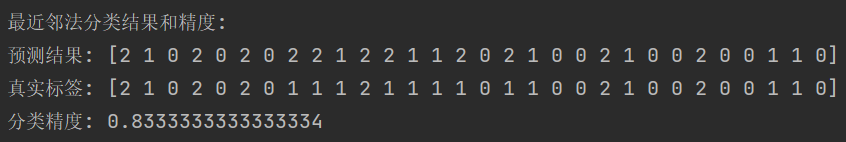


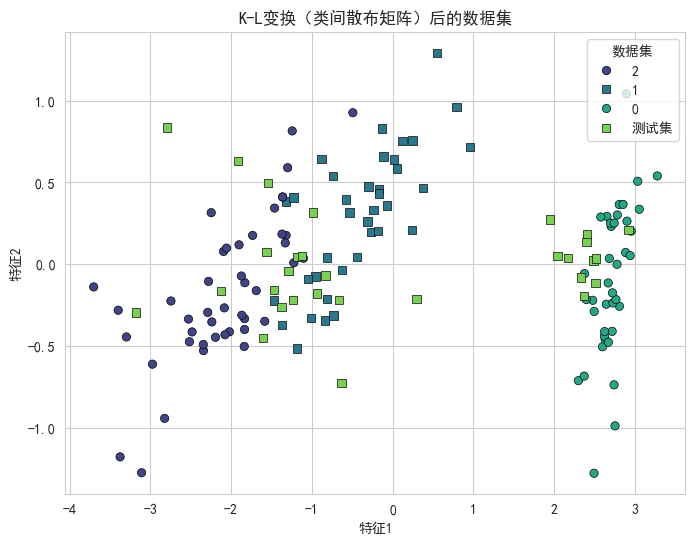
****

利用自相关矩阵得到最后精度为83.34%。

**5.3.2 类间散布矩阵**

kld = KLDimensionalityReduction(n\_components=2)  
X\_train\_kld2 = kld.fit\_transform(X\_train, y\_train, scatter\_matrix\_type='class')  
X\_test\_kld2 = kld.fit\_transform(X\_test, y\_test, scatter\_matrix\_type='class')  
  
knn\_classifier = KNNClassifier(k=3)  
knn\_classifier.fit(X\_train\_kld2, y\_train)  
y\_pred = knn\_classifier.predict(X\_test\_kld2)  
  
# 显示分类结果和分类精度  
accuracy = np.sum(y\_test == y\_pred) / len(y\_test)  
print("最近邻法分类结果和精度:")  
print("预测结果:", y\_pred)  
print("真实标签:", y\_test)  
print("分类精度:", accuracy)  
  
accuracy\_dict['K-L变换（类间散布矩阵）'] = accuracy

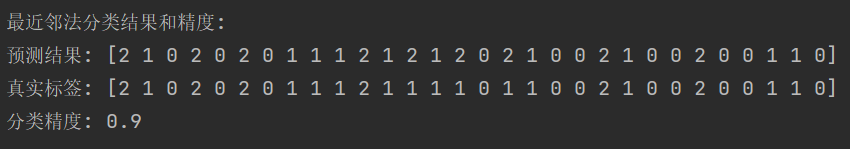


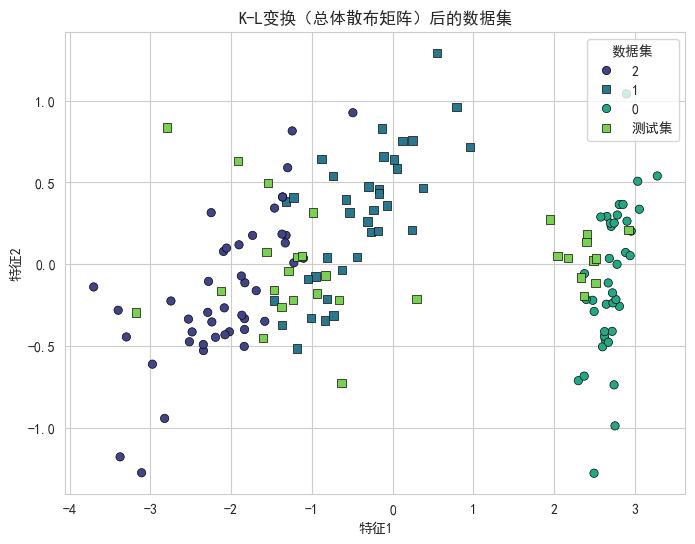
****

利用类间散布矩阵得到最后精度为83.34%。

**5.3.3 总体散布矩阵**

kld = KLDimensionalityReduction(n\_components=2)  
X\_train\_kld3 = kld.fit\_transform(X\_train, y\_train, scatter\_matrix\_type='total')  
X\_test\_kld3 = kld.fit\_transform(X\_test, y\_test, scatter\_matrix\_type='total')  
  
knn\_classifier = KNNClassifier(k=3)  
knn\_classifier.fit(X\_train\_kld3, y\_train)  
y\_pred = knn\_classifier.predict(X\_test\_kld3)  
  
# 显示分类结果和分类精度  
accuracy = np.sum(y\_test == y\_pred) / len(y\_test)  
print("最近邻法分类结果和精度:")  
print("预测结果:", y\_pred)  
print("真实标签:", y\_test)  
print("分类精度:", accuracy)  
  
accuracy\_dict['K-L变换（总体散布矩阵）'] = accuracy

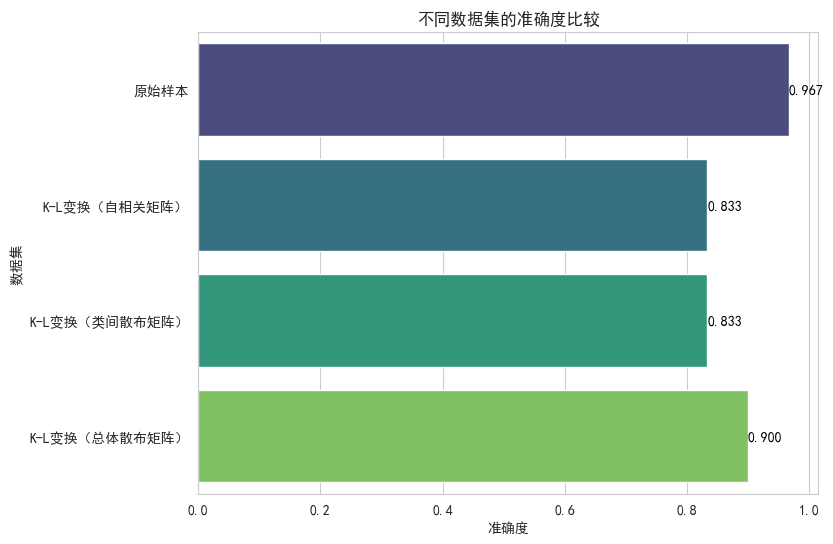


****

利用类间散布矩阵，其实就是PCA（主成分分析），得到最后精度为90.00%。

**5.4 总体对比图**

import seaborn as sns  
import matplotlib.pyplot as plt  
  
# 将字典转换为 DataFrame  
df\_accuracy = pd.DataFrame(list(accuracy\_dict.items()), columns=['数据集', '准确度'])  
  
# 使用Seaborn绘制条形图  
plt.figure(figsize=(8, 6))  
ax = sns.barplot(x='准确度', y='数据集', data=df\_accuracy, palette='viridis')  
  
# 在每个条形上标出数值  
for index, value in enumerate(df\_accuracy['准确度']):  
 ax.text(value, index, f'{value:.3f}', ha='left', va='center', fontsize=10, color='black')  
  
plt.title('不同数据集的准确度比较')  
plt.xlabel('准确度')  
plt.ylabel('数据集')  
plt.show()

****

**6实验心得**

在这次实验中，我学到了如何进行数据准备和理解，以及如何实现简单的K-Nearest Neighbors（KNN）分类器。通过加载和分割Iris数据集，我熟悉了数据的基本处理流程。

在K-L变换部分，我学到了如何使用特征提取和降维技术。通过实现K-L变换的类，我了解了自相关矩阵、类间散布矩阵和总体散布矩阵在特征提取中的作用。这种降维方法可以帮助我们更好地理解数据并减少模型复杂度。

整体而言，这次实验拓展了我的机器学习知识，让我对数据处理、KNN算法和特征提取有了更深刻的认识。