**实验四 特征选择与特征变换**

1. **实验目的**

掌握特征选择与特征变换的基本思想。通过特征选择与特征变换方法的学习，熟悉特征选择基本概念与方法，熟悉可分性判据，熟悉特征变换基本概念，掌握K-L变换方法和主要特点，掌握核函数变换方法和主要特点，掌握Fisher变换与判别方法。

1. **实验内容**

1. 数据集采用Iris鸢尾花数据，使用穷举法进行特征选择；

2. 对该数据进行特征变换，采用算法：PCA、KPCA、LDA。

1. **实验原理**

**基本概念**

1. **特征选取**

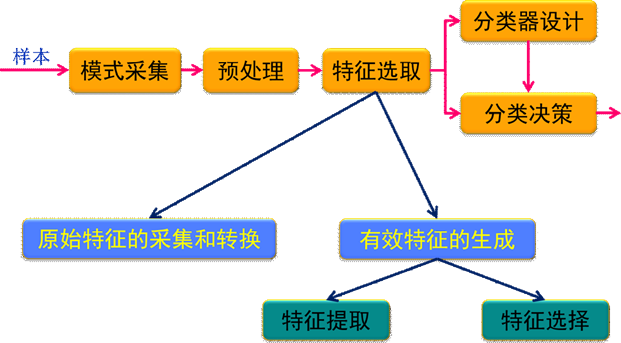


图 1 特征选取的内容

在模式识别系统中，确定分类和学习过程所使用的特征是非常重要的一个环节，获得对分类最有效的特征，同时尽最大可能减少特征维数，是特征选取的主要任务。特征选取可以分成原始特诊的采集和转换、有效特征的生成两个步骤。

（1）原始特征的采集和转换

对于一个模式识别任务，见过模式采集和预处理得到的模式信息不一定能直接用于模式分类，需要从中经过数据处理和转换得到对具体分类任务有效的特征。例如对于模式采集到的图像信息，其原始数据为像素点的颜色值矩阵，而对于不同的模式识别任务和模式识别算法，可以提取出不同类型的特征，如轮廓特征、颜色特征、颜色特征、数学特征。

（2）有效特征的生成

在获得了原始特征后，需要生成有效的特征，其主要目的是大幅度降低特征维度，减少模式识别算法的计算量。如果不经过这一降维过程，可能出现“维数灾难”，无法进行有效的模式识别分类。例如：在文本分类中，如果采用原始的词频统计数据作为分类特征，则有多少个不同的词就有多少维特征，一篇长文的特征维度会超过 1000 维，基本无法进行计算。

在降低特征维度的同时，还要提升所获得特征的有效性，因为尽管特征数量越多，用于分类的信息也越充足，但特征数量与分类有效性之间并不是线性关系。降维到同样数量时，不同的特征对分类的有效性是不同的。特征选取需要采用适当的算法，在降低特征维度的同时，最大可能地保留对分类有效的信息。

1. **特征提取**

特征提取是通过某种变换，将原始特征从高维空间映射到低维空间。

*A*：*X*→*Y*； *A* 称为特征提取器，通常是某种正交变换。

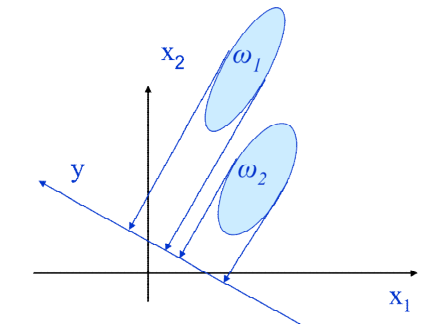
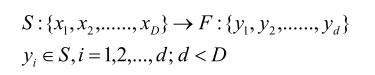


图 2 特征提取

对于各种可能的特征提取器，需要选择最优的一种，也就是降维后分类最有效的一种，通常设定一个准则函数 *J*(*A*)，使得取到最优特征提取时，准则函数值取到最大值，即 *J*(*A*\*)=max *J*(*A*)。

1. **特征选择**

特征选择是从高维特征中挑选出一些最有效的特征，以达到降低特征空间维数的目的。

****

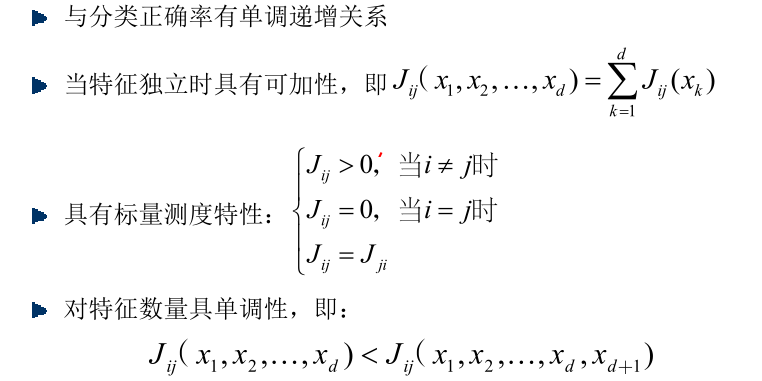
原始特征集合 *S* 中包含 *D* 个特征，目标特征集合 *F* 中包含 *d* 个特征。

同样，对于各种可能的特征选择方案，需要选择最优的一种，也就是降维后分类最有效的一种，通常设定一个准则函数 *J*(*F*)，使得取到最优特征选择时，准则函数值取到最大值，即 *J*(*F*\*)=max *J*(*F*)。

1. **准则函数的选取**

（1）准则函数的选取原则

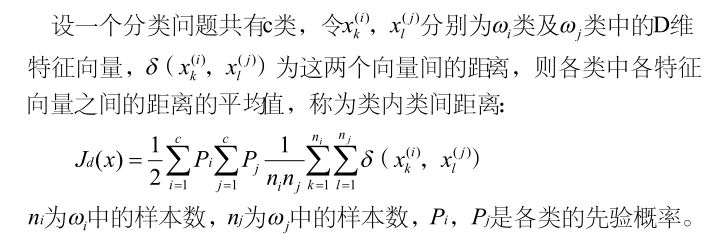
在设定了准则函数后，求取最优的特征提取或特征选择可以看作一个泛函求极值的问题，因此，准则函数的选取是特征提取或特征选择算法的关键。分类正确率是最佳的准则函数，如果经过某种方案的特征提取或特征选择后，得到的低维特征是所有可能方案中分类正确率最高的，就是最优的特征提取或特征选择。但是分类正确率难以直接计算，因此可以用特征选取方案对类别的可分性测度作为准则函数，通常两类之间的类别可分性测度要满足以下标准：



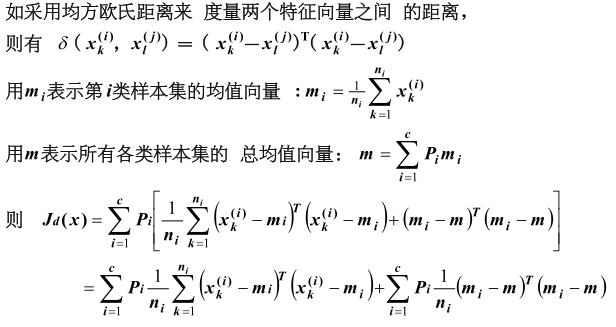
常用的类别可分析测度有基于类内类间距离和概率距离两种。

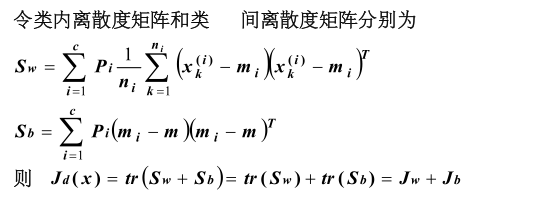
（2）类内类间距离

对于一个已知的样本集，类内类间距离的数学定义为：



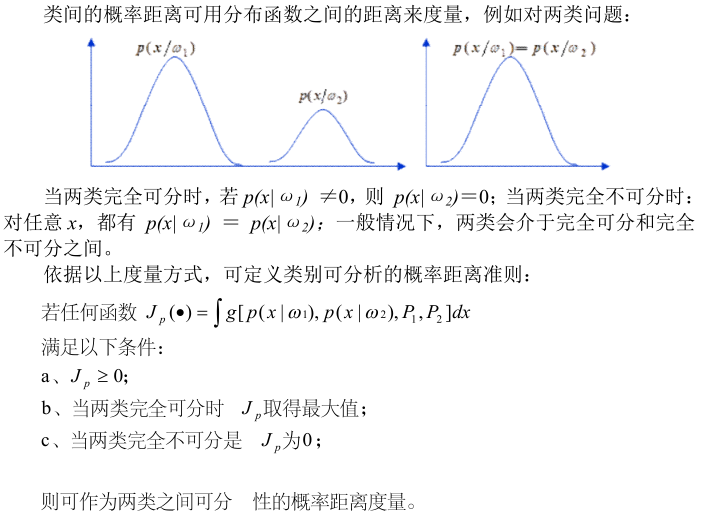
对于随机性的统计分类，如果样本集是给定的，则无论其中各类样本如何划分，类内类间距离都是相等的，也就是说，类内类间距离本身和分类错误率不相关，不能直接用于类别可分性测度。虽然类内类间距离本身不能用作类别可分性测度，但对其进行分解处理后，可以得到与类别可分性相关的测度指标。





*Jw* 称为类内平均距离，*Jb* 称为是类间平均距离。从类别可分性的要求来看，希望 *Jw* 尽可能小， *Jb* 尽可能大。

（3）概率距离



**使用类内类间距离进行特征提取**

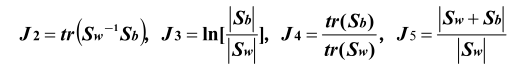
1. **准则函数的构造**

类内类间距离可表示为：*Jd=Jw+Jb*＝*tr*(*Sw*＋*Sb)*

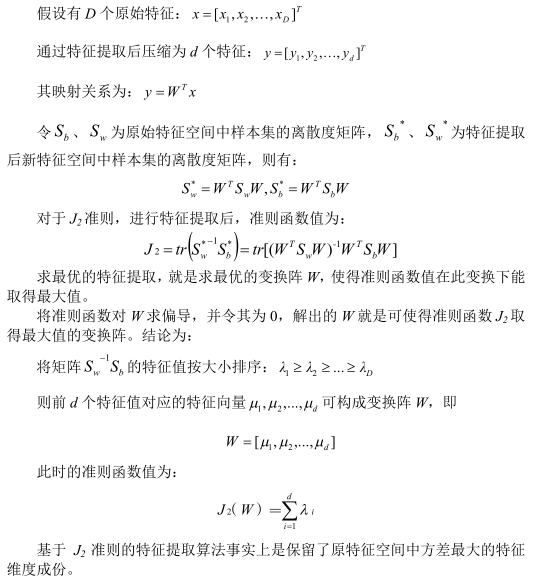
其中*Jw*是类内平均距离，*Jb* 是类间平均距离。

对于一个给定的样本集，*Jd* 是固定不变的。而通过特征提取后，新获得的特征使得样本集可以划分为不同的类，最佳的特征提取应当是使得各类之间的可分性最好，也就是 *Jb* 最大，*Jw* 最小。因此，可以直接采用 *Jb* 作为特征提取的准则函数，称为 *J1* 准则。

但直接使用准则难以得到可行的特征提取算法，考虑到类内离散度矩阵 *Sw*和类间离散度矩阵 *Sb* 是对称矩阵，迹和行列式值在正交变换下具有不变性，常构造以下几种特征提取准则函数：

**

1. **基于*J2*准则的特征提取算法**



**特征选择算法**

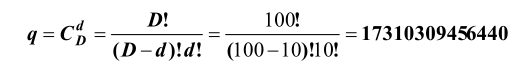
特征选择是从 *D* 个特征中挑选出最有效的 *d* 个特征的过程，常用的算法有以下几种。

1. **独立算法**

独立算法是指分别计算 *D* 个特征单独使用时的准则函数值，选取最优的前 *d*个特征。除非各特征相互独立，准则函数满足可加性，否则独立算法所得到的特征组合均不能保证是最优的特征组合。因此除特殊情况外，独立算法并不实用。

1. **穷举算法**

特征选择是一个组合过程，因此如从 *D* 个特征中考查所有可能的 *d* 个特征组合，计算其准则函数，找到最优的一个，从而得到最佳的特征选择结果，就是穷举法的算法。穷举法可以保证得到所有解中的全局最优解，但问题是计算量太大，例如当*D*＝100，*d*＝10，需要经过：

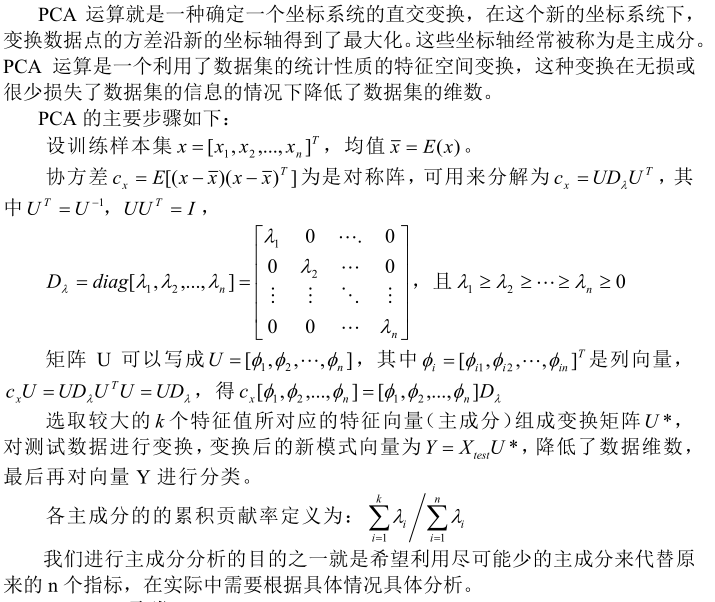
****

次计算才能得到特征选择结果，这在有限资源下基本是无解的。

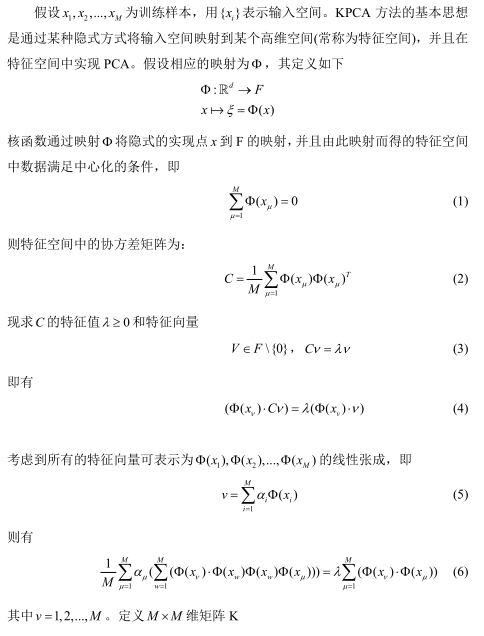
其他算法包括分支限界算法、次优算法、遗传算法等。

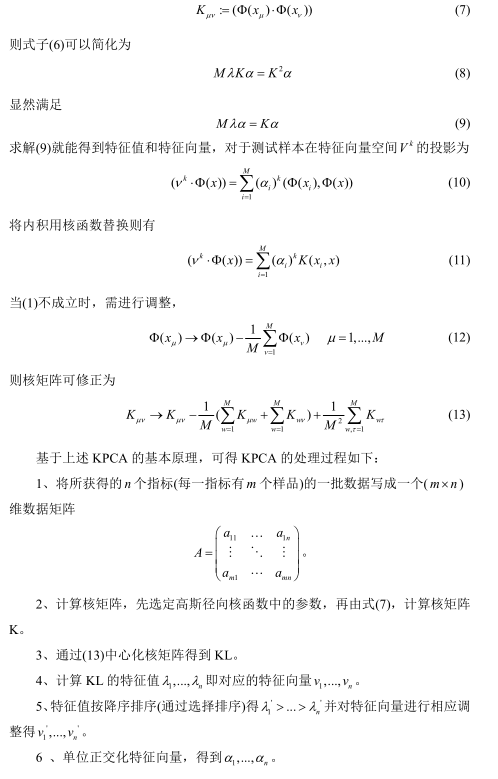
**特征变换算法**

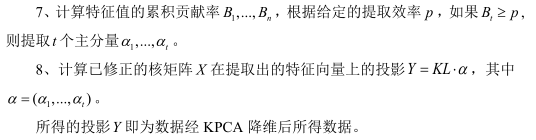
1. **主成分分析（PCA）**

****

1. **核主成分分析（KPCA）**



****

****

1. **线性判别分析**

假设有一组属于两个类的n个d维样本，．．．，，其中前个样本属于类 ，后面个样本属于类，均服从同协方差矩阵的高斯分布。各类样本均值向量（i=1，2）如式（1）：

= i=1，2 （1）

样本类内离散度矩阵和总的类内离散度矩阵如式(2)、式（3）：

 i=1，2 （2）

=+ （3）

样本类间离散度矩阵如式(4)：

 （4）

现寻找一个最佳超平面将两类分开，则只需将所有样本投影到此超平面的法线方向上，||w||=1:

 i=1,…,n （5）

得到n个标量,．．．，∈R，这n个标量相应的属于集合和，并且和能很好的分开。

为了能找到这样的能达到最好分类效果的投影方向w，Fisher规定了一个准则函数:要求选择的投影方向W能使降维后和两类具有最大的类间距离与类内距离比:

 （6）

其中类间距离用两类均值和之间的距离表示，类内距离用每类样本距其类均值距离的和表示，在式中为+。

其中(i=1，2)为降维后各类样本均值：

= i=1，2 （7）

（i=1，2)为降维后每类样本类内离散度，+为总的类内离散度：

， i=1，2 （8）

=+ （9）

类间离散度表示为。但式(6)Fisher准则函数并不是w的显示函数，无法根据此准则求解W，因此需要对Fisher准则函数形式进行修改：

因 i=1,…,n ，则

=== i=1,2 （10）

== （11）

=

同样（i=1，2）也可推出与w的关系：

=

== （12）

因此

+ = （13）

则最终可表示为：

 =  （14）

根据式(14)Fisher准则函数，要寻找一投影向量W，使最大化，则需对按变量W求导并使之为零：

 （15）

则需

（）()=0

= （16）

令，则

 （17）

这是一个广义特征值问题，若非奇异，则

 （18）

因此可以通过对进行特征值分解，将最大特征值对应的特征向量作为最佳投影方向W。

1. **实验过程**
2. **穷举法进行特征选择**

Iris数据集包含三类样本，分别是“setosa”、“versicolor“、”virginica“，每个样本有4个特征，分别是：'sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'，因此使用穷举法时分别有4选1、4选2、4选3、4选4，总共种选择。用穷举法选择特征后，使用sklearn中的LogisticRegression模型进行训练（60%作为训练集、40%作为测试集，运行500次求平均）得到分类结果。可以看到，随着所选特征数的增多，分类准确率Accuracy也基本呈现上升的趋势；选择1个特征时，'petal width (cm)'的分类效果最好；选择2个特征时，'sepal width (cm)'和‘petal width (cm)’的分类效果最好；选择3个特征时，'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'的分类效果最好；而选择全部4个特征时，分类效果在所有穷举法选出的特征组合中，效果是最好的。

|  |  |
| --- | --- |
| 选择特征 | Accuracy |
| 1 | 0.5980 |
| 2 | 0.4880 |
| 3 | 0.7497 |
| 4 | 0.8078 |
| 1，2 | 0.6912 |
| 1，3 | 0.8951 |
| 1，4 | 0.8553 |
| 2，3 | 0.8651 |
| 2，4 | 0.8978 |
| 3，4 | 0.8193 |
| 1，2，3 | 0.9076 |
| 1，2，4 | 0.8905 |
| 1，3，4 | 0.9264 |
| 2，3，4 | 0.9314 |
| 1，2，3，4 | 0.9400 |

1. **特征变换—PCA**

用PCA方法进行特征变换后，使用sklearn中的LogisticRegression模型进行训练（60%作为训练集、40%作为测试集，运行1000次求平均）得到分类结果。在一次试验中分解得到的特征值为[4.19667516 0.24062861 0.07800042 0.02352514]，特征向量矩阵如下：

[[ 0.36158968 -0.65653988 0.58099728 0.31725455]

[-0.08226889 -0.72971237 -0.59641809 -0.32409435]

[ 0.85657211 0.1757674 -0.07252408 -0.47971899]

[ 0.35884393 0.07470647 -0.54906091 0.75112056]]

可以看出，第一个特征值明显大于其他特征值，该特征值所对应的主成分对数据分类起了最主要的辨识作用。

当选择全部 4 个主成分时（评价指标分别为Accuracy、Precision、Recall、F1），实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 累计贡献率 | Accuracy | Precision | Recall | F1 |
| 1 | 0.8999 | 0.9094 | 0.9032 | 0.8985 |

选择特征值大的前 3 个主成分时，实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 累计贡献率 | Accuracy | Precision | Recall | F1 |
| 0.9948 | 0.9009 | 0.9120 | 0.9044 | 0.8994 |

选择特征值大的前 2 个主成分时，实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 累计贡献率 | Accuracy | Precision | Recall | F1 |
| 0.9776 | 0.8686 | 0.8862 | 0.8735 | 0.8659 |

选择特征值大的前 1 个主成分时，实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 累计贡献率 | Accuracy | Precision | Recall | F1 |
| 0.9246 | 0.8581 | 0.8839 | 0.8645 | 0.8499 |

从上面的数据可以看出，数据进行降维之后的分类效果受到一定的影响，并且随着选取特征值的数目的增加，分类的效果逐渐上升。从之前的单次实验中可以看出，第一个特征值明显大于其他特征值，对应的主成分所含的数据信息量最大，对分类起到了最主要作用。因此，可以抛弃该主成分，使用其他三个主成分进行分类。

选择第2个主成分时，实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 贡献率 | Accuracy | Precision | Recall | F1 |
| 0.053‬ | 0.4213 | 0.3716 | 0.4416 | 0.3715 |

选择第3个主成分时，实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 贡献率 | Accuracy | Precision | Recall | F1 |
| 0.0172 | 0.4184 | 0.3848 | 0.4452 | 0.3728 |

选择第4个主成分时，实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 贡献率 | Accuracy | Precision | Recall | F1 |
| 0.0052‬ | 0.2943 | 0.2264 | 0.3387 | 0.209 |

可以看出，这3个主成分对对分类任务并未起到主要作用。

**程序实现：**

*class* PCA:  
 *def* \_\_init\_\_(self, *n\_components*=*None*):  
 self.n\_components = *n\_components  
  
 def* fit(self, *X*):  
 self.mean\_ = *X*.mean(axis=0)  
 self.n\_features = *X*.shape[1]  
 \_, sigmas, VT = svd(*X* - self.mean\_, full\_matrices=*False*)  
 V = VT.T  
 a = np.argsort(sigmas)  
 # a = np.array([0,3,2,1])  
 V = V[:, np.argsort(sigmas)[::-1]]  
 # V = V[:, a[::-1]]  
 # V = V[:, [3,0,1,2]]  
 b = np.linalg.norm(V, axis=0)  
 V /= np.linalg.norm(V, axis=0)  
 self.eigenvalue = sigmas\*\*2 / *X*.shape[0]  
 self.scalings\_ = V  
  
 *def* transform(self, *X*):  
 *if not* hasattr(self, 'scalings\_'):  
 *raise* Exception('Please run `fit` before transform')  
 *if not* hasattr(self, 'mean\_'):  
 *raise* Exception('Please run `fit` before transform')  
 *assert X*.shape[1] == self.n\_features, 'X.shape[1] != self.n\_features'  
 *if* self.n\_components *is None*:  
 self.n\_components = self.n\_features  
 *return* ((*X* - self.mean\_) @ self.scalings\_)[:, :self.n\_components]  
  
 *def* fit\_transform(self, *X*):  
 self.fit(*X*)  
 *return* self.transform(*X*)

1. **特征变换—KPCA**

用KPCA方法进行特征变换后，使用sklearn中的LogisticRegression模型进行训练（60%作为训练集、40%作为测试集，运行500次求平均）得到分类结果。本实验针对 IRIS 数据进行核主成分分析，采用线性核函数（linear）、多项式核函数（poly）、径向基核函数（rbf）。

* 线性核函数（linear）

centered kernel matrix 的特征值为：

[629.50127448 36.09429217 11.70006231 3.52877104]

当选择全部 4 个主成分时（评价指标分别为Accuracy、Precision、Recall、F1），实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 累计贡献率 | Accuracy | Precision | Recall | F1 |
| 1.0 | 0.8997 | 0.9094 | 0.9032 | 0.8982 |

选择特征值大的前 3 个主成分时，实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 累计贡献率 | Accuracy | Precision | Recall | F1 |
| 0.9948 | 0.8975 | 0.9097 | 0.9014 | 0.8962 |

选择特征值大的前 2 个主成分时，实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 累计贡献率 | Accuracy | Precision | Recall | F1 |
| 0.9776 | 0.8691 | 0.8865 | 0.8735 | 0.8663 |

选择特征值大的前 1 个主成分时，实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 累计贡献率 | Accuracy | Precision | Recall | F1 |
| 0.9246 | 0.8578 | 0.8832 | 0.8637 | 0.8496 |

可以将采用线性核函数（linear）的KPCA与PCA得到的特征贡献度进行对比，发现其值是一样的，验证了采用线性核函数（linear）的KPCA就等价于PCA。

* 多项式核函数（poly）

centered kernel matrix 的特征值为：

[251974.73068994 7339.55084925 3578.31449477 1071.06819495]

当选择全部 4 个主成分时（评价指标分别为Accuracy、Precision、Recall、F1），实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 累计贡献率 | Accuracy | Precision | Recall | F1 |
| 1.0 | 0.9260 | 0.9278 | 0.9265 | 0.9252 |

选择特征值大的前 3 个主成分时，实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 累计贡献率 | Accuracy | Precision | Recall | F1 |
| 0.9959 | 0.9240 | 0.9282 | 0.9259 | 0.9215 |

选择特征值大的前 2 个主成分时，实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 累计贡献率 | Accuracy | Precision | Recall | F1 |
| 0.9824 | 0.9168 | 0.9244 | 0.9197 | 0.9137 |

选择特征值大的前 1 个主成分时，实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 累计贡献率 | Accuracy | Precision | Recall | F1 |
| 0.9546 | 0.8216 | 0.8425 | 0.8277 | 0.8075 |

* 径向基核函数（rbf）

centered kernel matrix 的特征值为：

[48.08181865 19.09195919 6.62368557 4.31294935]

当选择全部 4 个主成分时（评价指标分别为Accuracy、Precision、Recall、F1），实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 累计贡献率 | Accuracy | Precision | Recall | F1 |
| 1.0 | 0.9124 | 0.9163 | 0.9154 | 0.9118 |

选择特征值大的前 3 个主成分时，实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 累计贡献率 | Accuracy | Precision | Recall | F1 |
| 0.9448 | 0.9060 | 0.9096 | 0.9083 | 0.9048 |

选择特征值大的前 2 个主成分时，实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 累计贡献率 | Accuracy | Precision | Recall | F1 |
| 0.8600 | 0.9046 | 0.9085 | 0.9073 | 0.9037 |

选择特征值大的前 1 个主成分时，实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 累计贡献率 | Accuracy | Precision | Recall | F1 |
| 0.6156 | 0.6378 | 0.5385 | 0.6667 | 0.5592 |

* PCA与KPCA

主成分分析属于代数特征分析方法，是模式识别领域中一种经典的特征抽取和降维方法。但是 PCA 的缺点是需要很大的存储空间和计算复杂度。如果原始空间的维数是*n* , PCA 需要分解一个*n*  *n* 的非稀疏矩阵。因为 PCA 是一种线性映射方法，降维后的表示是由线性映射生成的，它忽略了数据之间高于 2 阶的相互关系，所以抽取的特征并不是最优的，这在一定程度上影响了 PCA 方法的效果。核主成分分析是线性 PCA 的非线性扩展算法，它采用非线性的方法抽取主成分，即 KPCA是在通过映射函数 把原始向量映射到高维空间 F，在 F上进行 PCA分析。

KPCA与 PCA具有本质上的区别：PCA 是基于指标的，而KPCA是基于样本的。KPCA不仅适合于解决非线性特征提取问题，而且它还能比PCA提供更多的特征数目和更多的特征质量，因为前者可提供的特征数目与输入样本的数目是相等的，而后者的特征数目仅为输入样本的维数。KPCA的优势是可以最大限度地抽取指标的信息；但是KPCA抽取指标的实际意义不是很明确，计算也比PCA复杂。

**程序实现：**

*class* KernelPCA:  
 *def* \_\_init\_\_(self, *n\_components*=*None*, *kernel*='linear', *kernel\_para*=0.1):  
 self.\_\_kernels = {'linear': self.\_\_kernel\_linear,  
 'rbf': self.\_\_kernel\_rbf}  
 *assert kernel in* self.\_\_kernels, 'arg kernel =\'' + *kernel* + '\' is not available'  
 self.n\_components = *n\_components* self.kernel = *kernel* self.kernel\_para = *kernel\_para  
  
 def* \_\_kernel\_linear(self, *x*, *y*):  
 *return x*.T @ *y  
  
 def* \_\_kernel\_rbf(self, *x*, *y*):  
 result = np.zeros((*x*.shape[1], *y*.shape[1]))  
 *for* i *in* range(result.shape[0]):  
 *for* j *in* range(result.shape[1]):  
 result[i, j] = np.exp(-self.kernel\_para \* (*x*[:, i] - *y*[:, j]).T @ (*x*[:, i] - *y*[:, j]))  
 *return* result  
  
 *def* fit(self, *X*):  
 self.n\_samples, self.n\_features = *X*.shape  
 K = self.\_\_kernels[self.kernel](*X*.T, *X*.T)  
 one\_M = np.ones((self.n\_samples, self.n\_samples)) / self.n\_samples  
 K = K - one\_M @ K - K @ one\_M + one\_M @ K @ one\_M  
 e\_vals, e\_vecs = eig(K)  
 e\_vals, e\_vecs = np.real(e\_vals), np.real(e\_vecs)  
 e\_vecs /= np.linalg.norm(e\_vecs, axis=0)  
 e\_vecs = e\_vecs[:, np.argsort(e\_vals)[::-1]] / np.sqrt(np.sort(e\_vals)[::-1])  
 self.scalings\_ = e\_vecs  
 self.e\_vals\_ = e\_vals  
 self.\_\_X = *X  
  
 def* transform(self, *X*):  
 *if not* hasattr(self, 'scalings\_'):  
 *raise* Exception('Please run `fit` before transform')  
 *assert X*.shape[1] == self.n\_features, 'X.shape[1] != self.n\_features'  
 *if* self.n\_components *is None*:  
 self.n\_components = *X*.shape[1]  
 K = self.\_\_kernels[self.kernel](*X*.T, self.\_\_X.T)  
 one\_M = np.ones((self.n\_samples, self.n\_samples)) / self.n\_samples  
 K = K - one\_M @ K - K @ one\_M + one\_M @ K @ one\_M  
 *return* (K @ self.scalings\_)[:, :self.n\_components]  
  
 *def* fit\_transform(self, *X*):  
 self.fit(*X*)  
 *return* self.transform(*X*)

1. **特征变换—LDA**

**（1）对一二类进行分类：**

第一类样本样本均值：（5.006, 3.418, 1.464, 0.244）

第一类样本样本均值：（5.936, 2.77 , 4.26 , 1.326）

类间离散度矩阵:

[[ 0.8649 -0.60264 2.60028 1.00626 ],

[-0.60264 0.419904 -1.811808 -0.701136],

[ 2.60028 -1.811808 7.817616 3.025272],

[ 1.00626 -0.701136 3.025272 1.170724]]

类内散度矩阵：

[[19.1434 9.0886 9.7528 3.25 ],

[ 9.0886 11.9388 4.6224 2.5794],

[ 9.7528 4.6224 12.2952 3.8612],

[ 3.25 2.5794 3.8612 2.4794]]

第一类与第二类最优为：

(0.06600043 0.42695115 -0.49229259 -0.75564851)

**（2）对一三类进行分类：**

第一类样本样本均值：（5.006, 3.418, 1.464, 0.244）

第一类样本样本均值：（6.588, 2.974, 5.552, 2.026）

类间离散度矩阵:

[[ 2.502724 -0.702408 6.467216 2.819124],

[-0.702408 0.197136 -1.815072 -0.791208],

[ 6.467216 -1.815072 16.711744 7.284816],

[ 2.819124 -0.791208 7.284816 3.175524]]

类内散度矩阵：

[[25.901 9.509 15.652 2.9224],

[ 9.509 12.21 4.07 2.8942],

[15.652 4.07 16.4 2.6716],

[ 2.9224 2.8942 2.6716 4.2594]]

第一类与第三类最优为：

(0.28105905 0.22080764 -0.65189412 -0.66879283)

**（3）对二三类进行分类：**

第一类样本样本均值：（5.936, 2.77 , 4.26 , 1.326）

第一类样本样本均值：（6.588, 2.974, 5.552, 2.026）

类间离散度矩阵:

[[0.425104 0.133008 0.842384 0.4564 ],

[0.133008 0.041616 0.263568 0.1428 ],

[0.842384 0.263568 1.669264 0.9044 ],

[0.4564 0.1428 0.9044 0.49 ]]

类内散度矩阵：

[[32.868 8.7684 23.8232 5.1388],

[ 8.7684 9.9212 7.5476 4.3528],

[23.8232 7.5476 25.7448 5.9744],

[ 5.1388 4.3528 5.9744 5.6124]]

第二类与第三类最优为：

(-0.22684996 -0.35584988 0.44461153 0.79008262)

首先进行一二两类的分类，将第一二两类数据与做点积，得到第一类样本均值 -1.52472315，第二类样本均值0.88462257，两类样本中心为-0.32005029。比样本中心大的值判为第一类，比样本中心小的值判为第二类，第一二两类样本均分类正确。

同样对第一三类样本分类，将一三类数据与做点积，得到第一类样本均值-2.46599151，第三类样本均值1.04414367，两类样本中心为-0.71092392，同样所有样本分类正确。

对第二类和第三类样本分类，将第二三类数据与做点积，得到第二类样本均值-1.51640554，第三类样本均值0.60940916，两类样本中心为1.06290735，第二类的第21，34个样本分类错误，第三类的第34个样本分类错误。

**程序实现：**

*class* LinearDiscriminantAnalysis:  
 *def* \_\_init\_\_(self, *n\_components*=*None*):  
 self.n\_components = *n\_components  
  
 def* fit(self, *X*, *y*):  
 self.n\_features = *X*.shape[1]  
 u1 = *X*[*y* == 1, :].mean(axis=0).reshape((-1, 1))  
 u0 = *X*[*y* == 0, :].mean(axis=0).reshape((-1, 1))  
 Sb = (u1 - u0) @ (u1 - u0).T  
 Sw = (*X*[*y* == 1, :].T - u1) @ (*X*[*y* == 1, :].T - u1).T + (*X*[*y* == 0, :].T - u0) @ (*X*[*y* == 0, :].T - u0).T  
 e\_vals, e\_vecs = eig(np.linalg.pinv(Sw) @ Sb)  
 e\_vals, e\_vecs = np.real(e\_vals), np.real(e\_vecs)  
 e\_vecs = e\_vecs[:, np.argsort(e\_vals)[::-1]]  
 e\_vecs /= np.linalg.norm(e\_vecs, axis=0)  
 self.scalings\_ = e\_vecs  
  
 *def* transform(self, *X*):  
 *if not* hasattr(self, 'scalings\_'):  
 *raise* Exception('Please run `fit` before transform')  
 *assert X*.shape[1] == self.n\_features, 'X.shape[1] != self.n\_features'  
 *if* self.n\_components *is None*:  
 self.n\_components = self.n\_features  
 *return* (*X* @ self.scalings\_)[:, :self.n\_components]  
  
 *def* fit\_transform(self, *X*, *y*):  
 self.fit(*X*, *y*)  
 *return* self.transform(*X*)

1. **实验结论**

通过本次实验，基本掌握了特征选择与特征变换的基本思想。通过特征选择与特征变换方法的学习，熟悉了特征选择基本概念与方法、可分性判据以及特征变换基本概念，掌握了核函数变换方法和主要特点及Fisher变换与判别方法。