西南财经大学
SOUTHWESTERN UNIVERSITY OF FINANCE AND ECONOMICS

Name: 兰文婕

Mission objectives: **iris**

Course name: **多元统计**

Date:2022 年 5 月 17 日

1 正态性检验

1.1 单变量的多元正态性检验

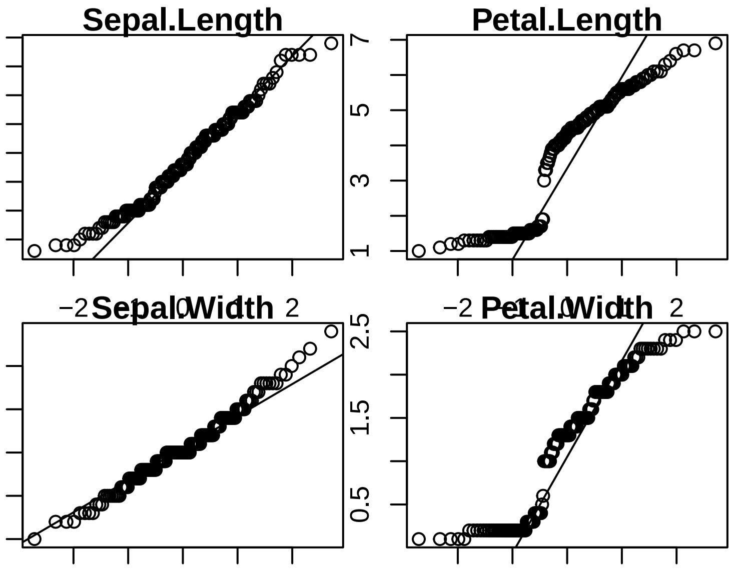
单因变量正态是多因变量多元正态的必要非充分条件。

```
iris_d=iris[,1:4]

#检验每一个变量的qqplot
layout(matrix(1:8,nc=4))
sapply(colnames(iris_d),function(x)
{qqnorm(iris_d[[x]],main=x)
qqline(iris_d[[x]])
})

```

code 1: 单变量多元性检验




```
## $Sepal.Length
## NULL
##
## $Sepal.Width
## NULL
##
## $Petal.Length
## NULL
##
## $Petal.Width
## NULL
```

Figure 1: 1.1

由上图可知,sepal.length 与 sepal.width 的概率图大致均匀分布在 y=x 线两侧,petal.length 与 petal.width 的图明显偏离 y=x 线，左边朝直线下方弯曲，右边朝直线上方弯曲，具有长尾分布的特征。

1.2 二维散点图

判定指标：通过变量间的两两散点图，如果存在不服从线性关系的两个变量，也就是散点图不成直线，则说明数据不服从多元正态分布。

```
pairs(iris_d)

```

code 2: 变量之间相关性检验

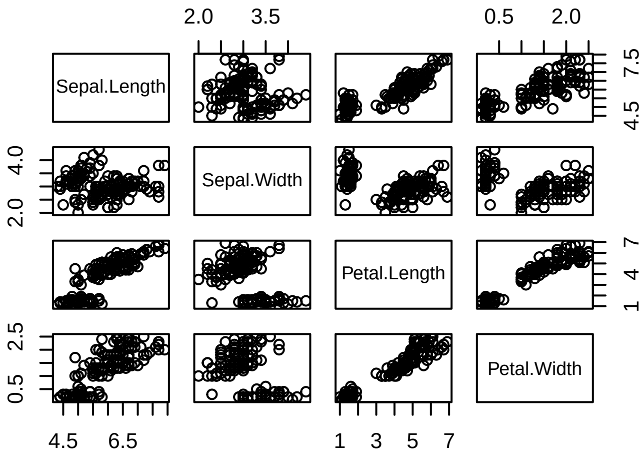


Figure 2: 1.2

1.3 马氏距离是否服从卡方分布

```
 y <- iris_d
cm <- colMeans(y)
S <- cov(y)
d <- apply(y, 1, function(y) t(y - cm) %*% solve(S) %*% (y - cm))
plot(qc <- qchisq((1:nrow(y) - 1/2) / nrow(y), df = 7), sd <- sort(d),
     xlab = expression(paste(chi[7]^2, " Quantile")),
     ylab = "Ordered distances", xlim = range(qc) * c(1, 1.1))
oups <- which(rank(abs(qc - sd), ties = "random") > nrow(y) - 3)
text(qc[oups], sd[oups] - 1.5, names(oups))
abline(a = 0, b = 1)

```

code 3: 马氏距离卡方分布检验

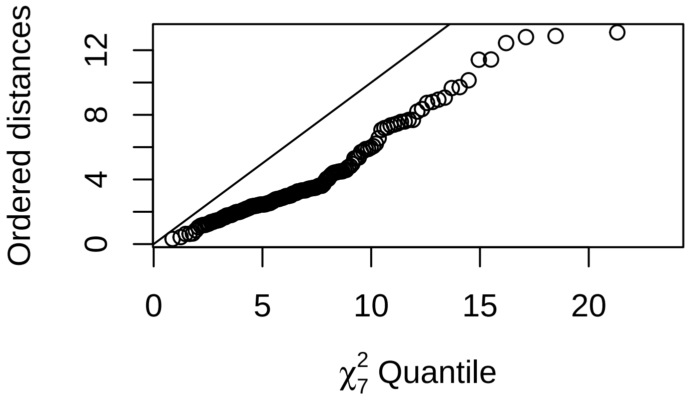


Figure 3: 1.3

从 qq 图可以看出，点距离标准线的偏离程度较大，说明可以拒绝数据服从正态分布的假设。

1.4 夏皮罗-威尔克检验 Shapiro–Wilk test

```
   library(mvnormtest)
iris_d=-t(iris[,1:4])
mshapiro.test(iris_d)

```

code 4: Shapiro–Wilk test

```
##
##  Shapiro-Wilk normality test
##
## data:  Z
## W = 0.97935, p-value = 0.02342
```

Figure 4: 1.4

p-value<0.05, 说明可以在 0.05 的水平上拒绝数据服从多元正态分布的假设。

1.5 总结

正态性检验在样本量不大时是有必要的，当样本量足够大时，由中心极限定理，不必太过纠结正态性的问题。

2 分类

使用 python。

2.1 LDA

2.1.1 降维至三维

```python
 #导入各种程序包
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
clf = LinearDiscriminantAnalysis()
from sklearn.model_selection import train_test_split
##划分训练集与测试集
from sklearn.datasets import load_iris
iris = load_iris()
X_train=iris.data
y_train=iris.target
X_train,X_test,y_train,y_test=train_test_split(X_train,y_train,
    test_size=0.4,random_state=0,stratify=y_train)
#三维（数据可视化）
def plot_LDA(converted_X,y):
    '''
    绘制经过 LDA 转换后的数据
    '''
    from mpl_toolkits.mplot3d import Axes3D
    fig=plt.figure()
    ax=Axes3D(fig)
    colors=['dodgerblue','lime','darkorange']
    target_names = iris.target_names
    markers='o*s'

    for target,color,marker,target_name in zip([0,1,2],colors,markers,target_names):
        pos=(y==target).ravel()
        X=converted_X[pos,:]

        ax.scatter(X[:,0], X[:,1], X[:,2],color=color,marker=marker,
           label=target_name)
    ax.legend(loc="best")
    fig.suptitle("After LDA")
def plot_LDA1(converted_X,y):
    '''
    绘制未经 LDA 转换后的数据

```

```
    '''
    from mpl_toolkits.mplot3d import Axes3D
    fig=plt.figure()
    ax=Axes3D(fig)
    colors=['dodgerblue','lime','darkorange']
    target_names = iris.target_names
    markers='o*s'

    for target,color,marker,target_name in zip([0,1,2],colors,markers,target_names):
        pos=(y==target).ravel()
        X=converted_X[pos,:]

        ax.scatter(X[:,0], X[:,1], X[:,2],color=color,marker=marker,
           label=target_name)
    ax.legend(loc="best")
    fig.suptitle("Before LDA")
    clf.fit(X_train,y_train)
# converted_X(150,3) 对数据降维了    权值lda.coef_(3, 4)  lda.intercept_(3,)
converted_X=np.dot(X_train,np.transpose(clf.coef_))+clf.intercept_  # X*权值+偏置b  就是输出值
plot_LDA(converted_X,y_train)
#最初未转化的三维的情况
plot_LDA1(X_train,y_train)
y_pred = clf.predict(X_test)
print(y_pred)
print(y_test)
#评估
# 分类报告：precision/recall/fi-score/均值/分类个数
from sklearn.metrics import classification_report
target_names = ['class 0', 'class 1', 'class 2']
print(classification_report(y_test, y_pred, target_names=target_names))
#准确率
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
# 混淆矩阵
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

code 5: 3—dimension-LDA

运行结果：

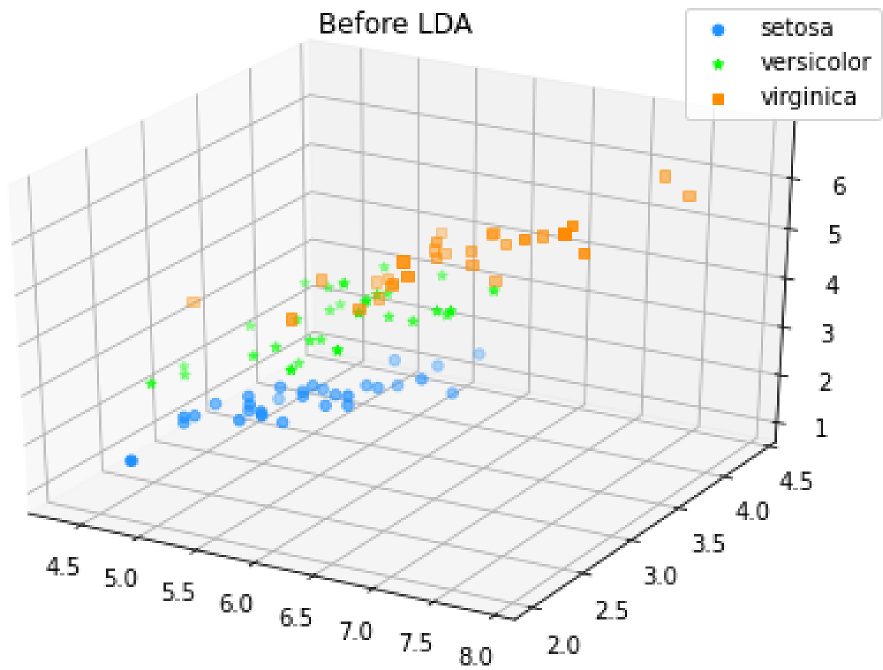


Figure 5: Before 3-dimension-LDA

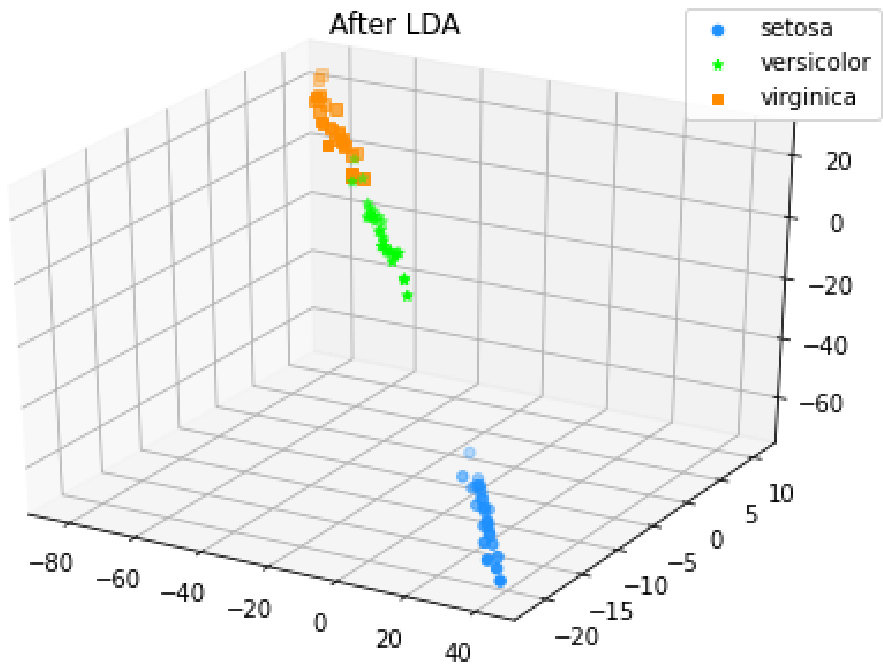


Figure 6: After 2-dimension-LDA

2.1.2 降维至二维

大致过程与上述相同，下面只给出了关键部分的代码。

```
#二维
```

```python
#数据训练
lda = LinearDiscriminantAnalysis(n_components=2)
X_transform = lda.fit(X_train, y_train).transform(X_test)
y_pred = lda.predict(X_test)
#绘图
colors=['dodgerblue','lime','darkorange']
plt.figure()
target_names = iris.target_names
lw=2
for color, i, target_name in zip(colors, [0, 1, 2], target_names):
    plt.scatter(X_transform[y_pred == i, 0], X_transform[y_pred == i, 1], alpha=.8, color=color,
                label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('After 2-dimension-LDA')
plt.show()

```

code 6: 2—dimension-LDA

2.1.3 运行结果

Tabel 1: 评估分类报告 LDA

	precision	recall	f1-score	support
class 0	1.00	1.00	1.00	20
class 1	1.00	1.00	1.00	20
class 2	1.00	1.00	1.00	20
accuracy	1.00	60		
macro avg	1.00	1.00	1.00	60
weighted avg	1.00	1.00	1.00	60

准确率：1.0

混淆矩阵：

$$\begin{matrix} 20 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 20 \end{matrix}$$

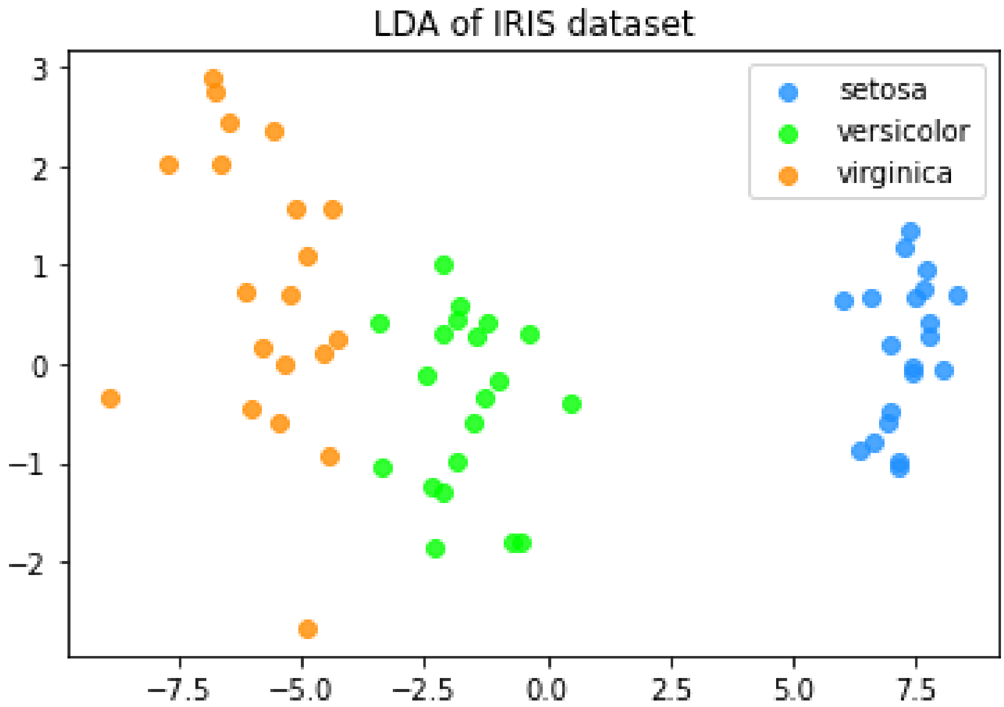


Figure 7: After 2-dimension-LDA

2.2 SVM

```
 #导入各种程序包
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.svm import SVC
#网格搜索调参
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                     'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]
#评分方法定义
scores = ['precision', 'recall']
for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
    print()

     # 调用 GridSearchCV, 将 SVC(), tuned_parameters, cv=5, 还有 scoring 传递进去,
    clf = GridSearchCV(SVC(), tuned_parameters, cv=5, scoring='%s_macro' % score)
    # 用训练集训练这个学习器 clf
```

```
    clf.fit(X_train, y_train)

    print("Best parameters set found on development set:")
    print()

    # 再调用 clf.best_params_ 就能直接得到最好的参数搭配结果
    print(clf.best_params_)

    print()
    print("Grid scores on development set:")
    print()
    means = clf.cv_results_['mean_test_score']
    stds = clf.cv_results_['std_test_score']

    # 看一下具体的参数间不同数值的组合后得到的分数是多少
    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r"
              % (mean, std * 2, params))
    y_pred=clf.predict(X_test)

```

code 7: SVM

2.2.1 运行结果

Tabel 2: 评估分类报告 SVM

	precision	recall	f1-score	support
class 0	1.00	1.00	1.00	20
class 1	1.00	0.90	0.95	20
class 2	0.91	1.00	0.95	20
accuracy	0.97	60		
macro avg	0.97	0.97	0.97	60
weighted avg	0.97	0.97	0.97	60

准确率：0.9666666666666667

混淆矩阵：

$$\begin{matrix} 20 & 0 & 0 \\ 0 & 18 & 2 \\ 0 & 0 & 20 \end{matrix}$$

2.3 总结

从上述评估结果对比可知，LDA 在 iris 数据集上的分类效果优于 SVM 的分类效果。