

数据科学作业3--鸢尾花分类

1.用到的分类算法及其实现

1.1支持向量机算法（SVM）

在线性可分的数据集中，可能存在无数个决策边界，它们的训练误差都等于零。但因为使用不同的超平面，分类器的泛化误差是不一样的。具有较大边缘的决策边界比那些具有较小边缘的决策边界具有更好的泛化误差。而线性SVM分类器正是寻找具有最大边缘的超平面作为决策边界，因此它也经常被称为最大边缘分类器。模型训练过程和结果如下：

```
from sklearn.svm import LinearSVC
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import numpy as np
if __name__ == "__main__":
    dataset = load_iris()
    X = dataset.data
    y = dataset.target
    Xd_train, Xd_test, y_train, y_test = train_test_split(X, y, random_state=14)
    clf = LinearSVC(random_state=0)
    clf = clf.fit(Xd_train, y_train)
    y_predicted = clf.predict(Xd_test)
    accuracy = np.mean(y_predicted == y_test) * 100
    print("y_test\n", y_test)
    print("y_predicted\n", y_predicted)
    print("accuracy:", accuracy)
```

```
D:\python_project\venv\Scripts\python.exe D:/Python_project/数据科学课程/test1.py
D:\python_project\venv\lib\site-packages\sklearn\svm\_base.py:1225: ConvergenceWarning:
  iterations.
  warnings.warn(
y_test
[0 0 0 1 2 1 0 1 0 1 2 0 2 2 0 1 0 2 2 1 0 0 0 1 0 2 0 1 1 0 0 1 1 0 1 0 2
 1]
y_predicted
[0 0 0 1 2 1 0 1 0 1 2 0 2 2 0 1 0 2 2 1 0 0 0 1 0 2 0 1 1 0 0 1 1 0 1 0 2
 1]
accuracy: 100.0
```

1.2决策树

决策树是一种树形结构，其中每个节点表示一个属性的测试；每个分支代表一个测试输出，每个叶子节点代表一种类别。分裂属性就是在某个节点处按照某一特征属性的不同划分构造不同的分支，其目标是让各个分裂子集尽可能地“纯”。尽可能“纯”就是尽量让一个分裂子集中待分类项属于同一类别。分裂属性分为三种不同的情况：

1.属性是离散值且不要求生成二叉决策树。此时用属性的每一个划分作为一个分支。 2.属性是离散值且要求生成二叉决策树。此时使用属性划分的一个子集进行测试，按照“属于此子集”和“不属于此子集”分成两个分支。 3.属性是连续值。此时确定一个值作为分裂点split point，按照 $> \text{split point}$ 和 $\leq \text{split point}$ 生成两个分支。

模型训练过程和结果如下：

```
from sklearn import tree
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import numpy as np

if __name__ == "__main__":
    # 加载数据
    dataset = load_iris()
    # 提取属性数据
    X = dataset.data
    # 提取标签数据
    y = dataset.target
    # train_test_split函数用于划分数据集为训练集和测试集，其中参数test_size默认为0.25,表示
    # 将25%的数据划分为测试集
    Xd_train, Xd_test, y_train, y_test = train_test_split(X, y, random_state=14)
    # 创建决策树分类器
    clf = tree.DecisionTreeClassifier()
    # 训练分类器模型
    clf = clf.fit(Xd_train, y_train)
    y_predicted = clf.predict(Xd_test)

    # 计算预测准确率
    accuracy = np.mean(y_predicted == y_test) * 100

    print("y_test\n", y_test)
    print("y_predicted\n", y_predicted)
    print("accuracy:", accuracy)
```

```
test1 x
D:\python_project\venv\Scripts\python.exe D:/Python_project/数据科学课程/test1.py
y_test
[0 0 0 1 2 1 0 1 0 1 2 0 2 2 0 1 0 2 2 1 0 0 0 1 0 2 0 1 1 0 0 1 1 0 1 0 2
1]
y_predicted
[0 0 0 1 2 1 0 1 0 1 1 0 2 2 0 1 0 2 2 1 0 0 0 1 0 2 0 1 1 0 0 1 1 0 1 0 2
1]
accuracy: 97.36842105263158

进程已结束,退出代码0
```

1.3K最近邻算法

K 近邻算法使用的模型实际上对应于对特征空间的划分。K 值的选择，距离度量和分类决策规则是该算法的三个基本要素。K 值的选择会对算法的结果产生重大影响：

1.K值较小意味着只有与输入实例较近的训练实例才会对预测结果起作用，但容易发生过拟合；2.如果 K 值较大，优点是可以减少学习的估计误差，但缺点是学习的近似误差增大，这时与输入实例较远的训练实例也会对预测起作用，使预测发生错误。在实际应用中，K 值一般选择一个较小的数值，通常采用交叉验证的方法来选择最优的 K 值。3.随着训练实例数目趋向于无穷和 K=1 时，误差率不会超过贝叶斯误差率的2倍，如果K也趋向于无穷，则误差率趋向于贝叶斯误差率。4.该算法中的分类决策规则往往是多数表决，即由输入实例的 K 个最临近的训练实例中的多数类决定输入实例的类别距离度量一般采用 Lp 距离，当p=2时，即为欧氏距离，在度量之前，应该将每个属性的值规范化，这样有助于防止具有较大初始值域的属性比具有较小初始值域的属性的权重过大。

最常见以欧氏距离作为衡量标准：

$$D(x_i, x_j) = \left(\sum_{k=1}^n (x_i^k - x_j^k)^2 \right)^{\frac{1}{2}}$$

模型训练过程和结果如下：

```

from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
import numpy as np
from sklearn.model_selection import train_test_split
if __name__ == '__main__':
    dataset = load_iris()
    X = dataset.data
    y = dataset.target
    Xd_train, Xd_test, y_train, y_test = train_test_split(X, y, random_state=14)
    clf = KNeighborsClassifier(n_neighbors=3).fit(Xd_train, y_train)
    y_predicted = clf.predict(Xd_test)
    # 准确率
    accuracy = np.mean(y_predicted == y_test) * 100
    print ("y_test\n", y_test)
    print ("y_predicted\n", y_predicted)
    print ("accuracy:", accuracy)

```

```

test1 x
D:\python_project\venv\Scripts\python.exe D:/Python_project/数据科学课程/test1.py
y_test
[0 0 0 1 2 1 0 1 0 1 2 0 2 2 0 1 0 2 2 1 0 0 0 1 0 2 0 1 1 0 0 1 1 0 1 0 2
 1]
y_predicted
[0 0 0 1 2 1 0 1 0 1 1 0 2 2 0 1 0 2 2 1 0 0 0 1 0 2 0 1 1 0 0 1 1 0 1 0 2
 1]
accuracy: 97.36842105263158

```

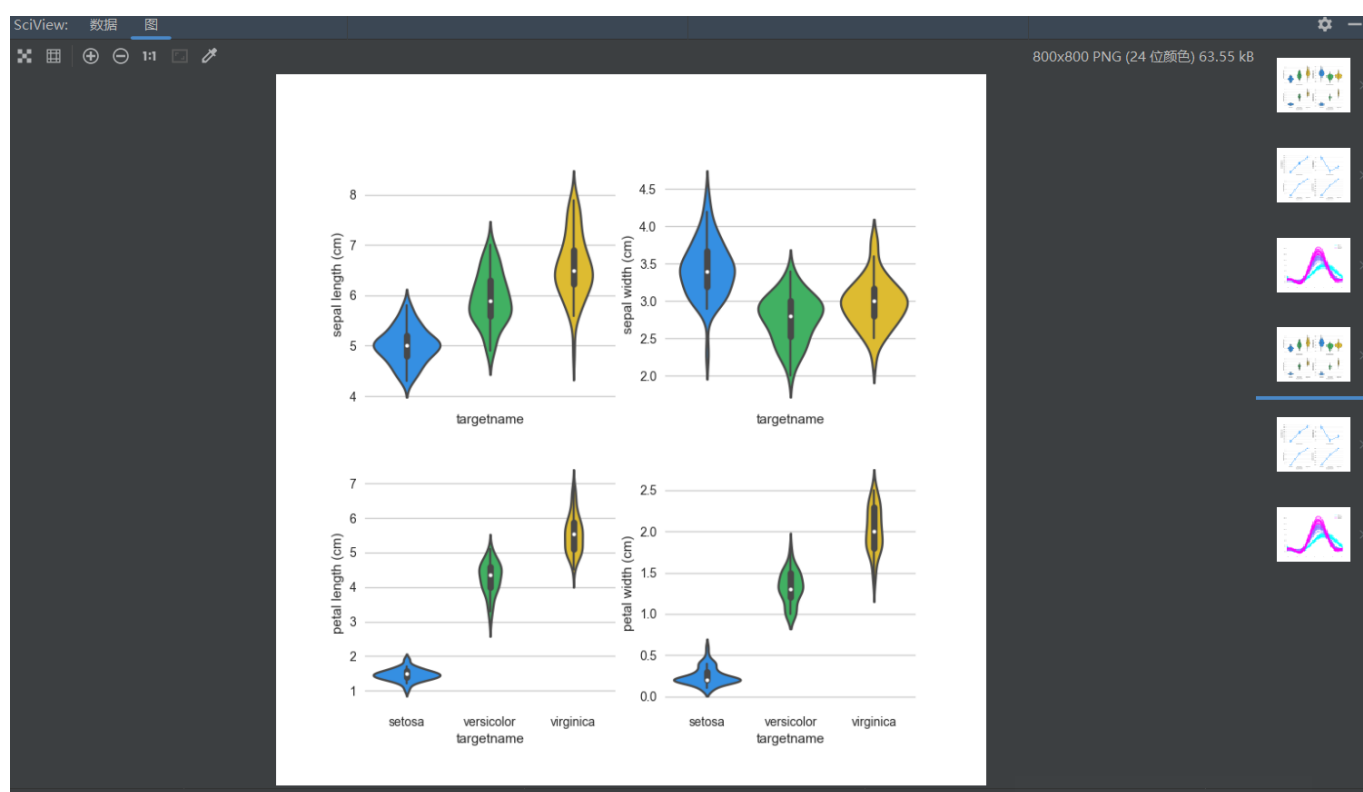
2.可视化处理

2.1绘制 Violinplot

```
f, axes = plt.subplots(2, 2, figsize=(8, 8), sharex=True)
sns.despine(left=True)

sns.violinplot(x='targetname', y='sepal length (cm)', data=iris, palette=antV,
ax=axes[0, 0])
sns.violinplot(x='targetname', y='sepal width (cm)', data=iris, palette=antV,
ax=axes[0, 1])
sns.violinplot(x='targetname', y='petal length (cm)', data=iris, palette=antV,
ax=axes[1, 0])
sns.violinplot(x='targetname', y='petal width (cm)', data=iris, palette=antV,
ax=axes[1, 1])

plt.show()
```

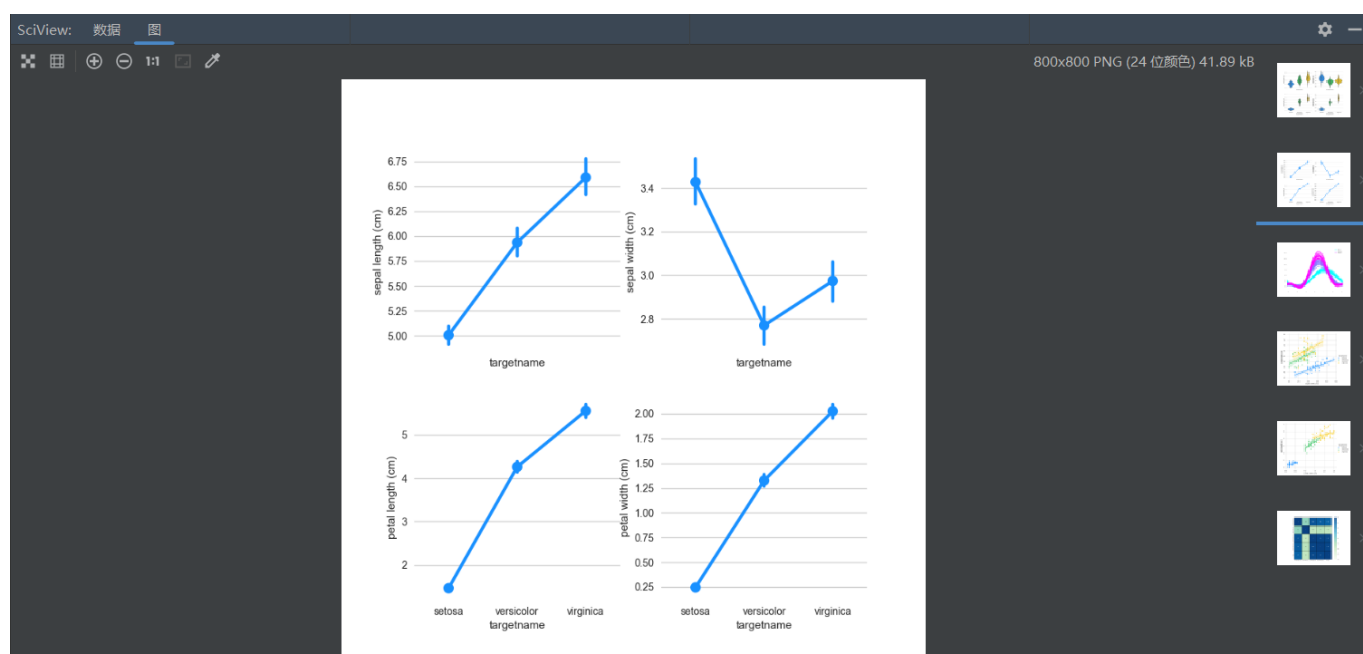


2.2绘制pointplot

```
f, axes = plt.subplots(2, 2, figsize=(8, 8), sharex=True)
sns.despine(left=True)

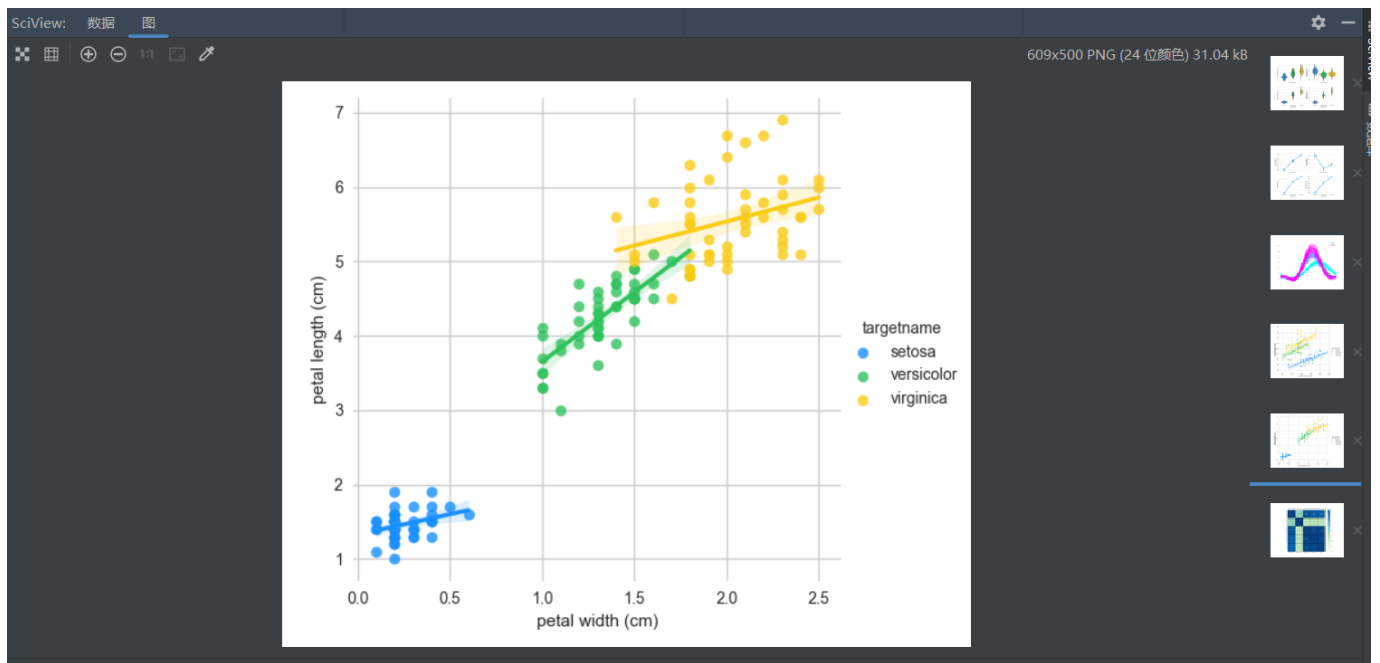
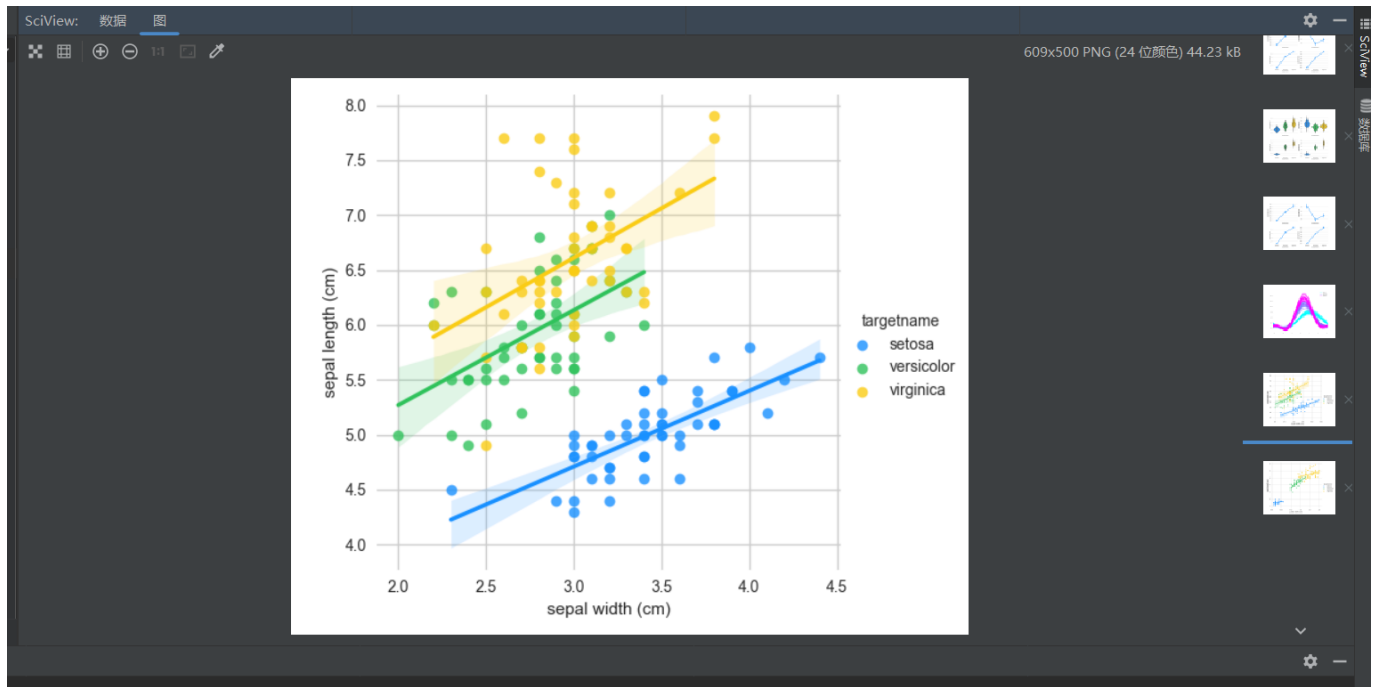
sns.pointplot(x='targetname', y='sepal length (cm)', data=iris, color=antV[0],
ax=axes[0, 0])
sns.pointplot(x='targetname', y='sepal width (cm)', data=iris, color=antV[0],
ax=axes[0, 1])
sns.pointplot(x='targetname', y='petal length (cm)', data=iris, color=antV[0],
ax=axes[1, 0])
sns.pointplot(x='targetname', y='petal width (cm)', data=iris, color=antV[0],
ax=axes[1, 1])

plt.show()
```



2.3线性回归可视化

```
g = sns.lmplot(data=iris, x='sepal width (cm)',
y='sepal length (cm)', palette=antV, hue='targetname')
g = sns.lmplot(data=iris, x='petal width (cm)',
y='petal length (cm)', palette=antV, hue='targetname')
```

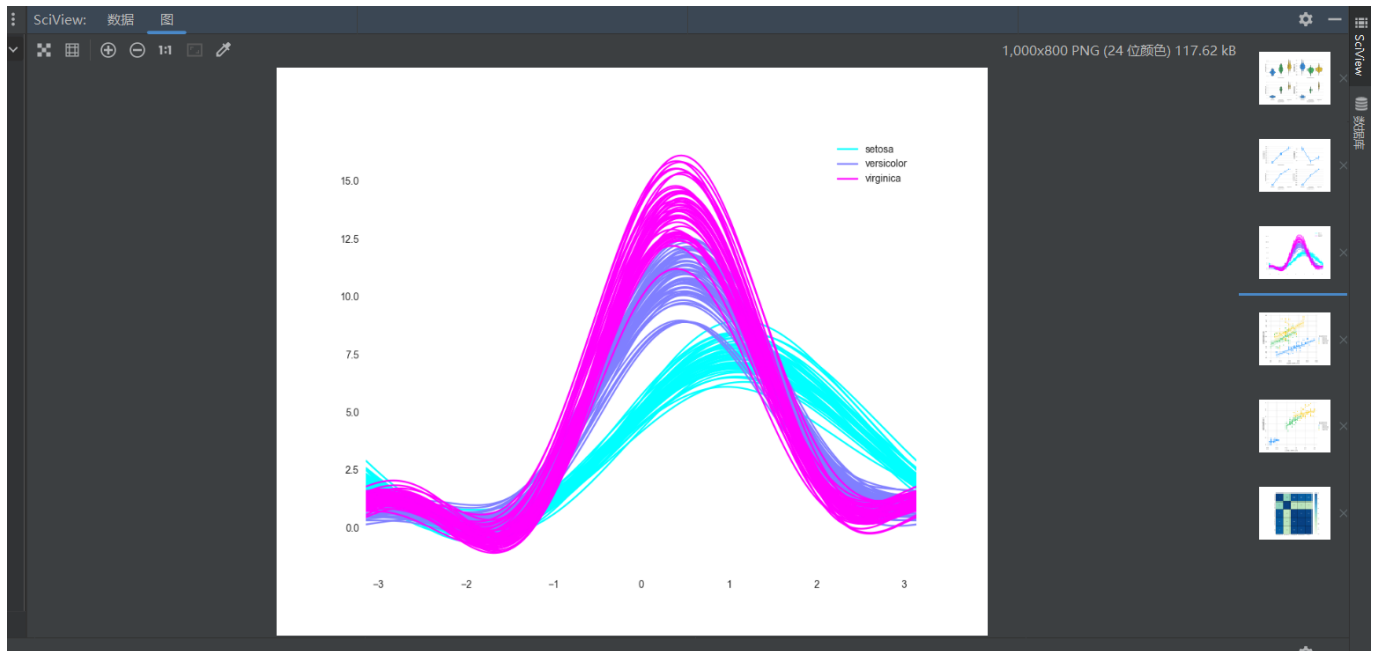


2.4使用 Andrews Curves 将每个多变量观测值转换为曲线并表示傅立叶级数的系数

这对于检测时间序列数据中的异常值很有用。

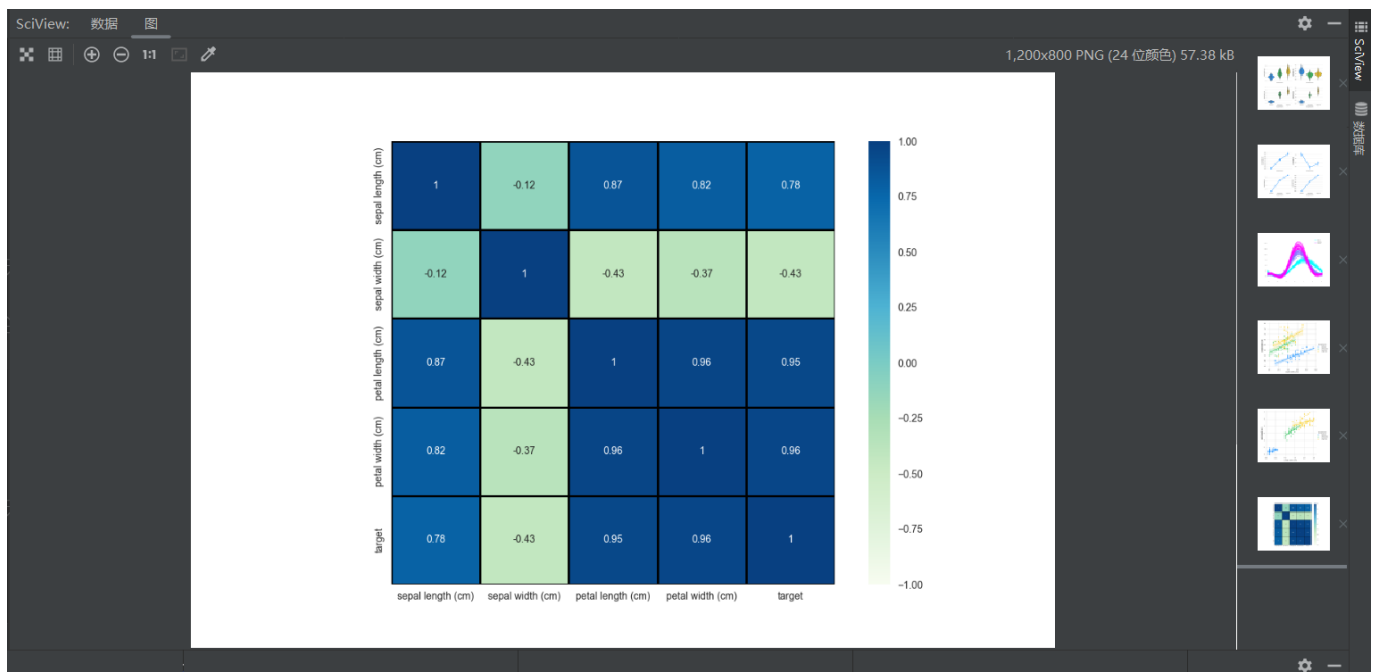
```
plt.subplots(figsize = (10,8))
plotting.andrews_curves(iris, 'targetname', colormap='cool')

plt.show()
```



2.5通过热图找出数据集中不同特征之间的相关性

```
fig=sns.heatmap(iris.corr(), annot=True,cmap='GnBu', linewidths=1,  
linecolor='k',square=True, mask=False, vmin=-1, vmax=1, cbar_kws  
={"orientation": "vertical"}, cbar=True)
```



3.集成各种分类方法


```

X = iris[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']]
y = iris['targetname']

encoder = LabelEncoder()
y = encoder.fit_transform(y)
#print(y)

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size = 0.3,
random_state = 101)
#print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

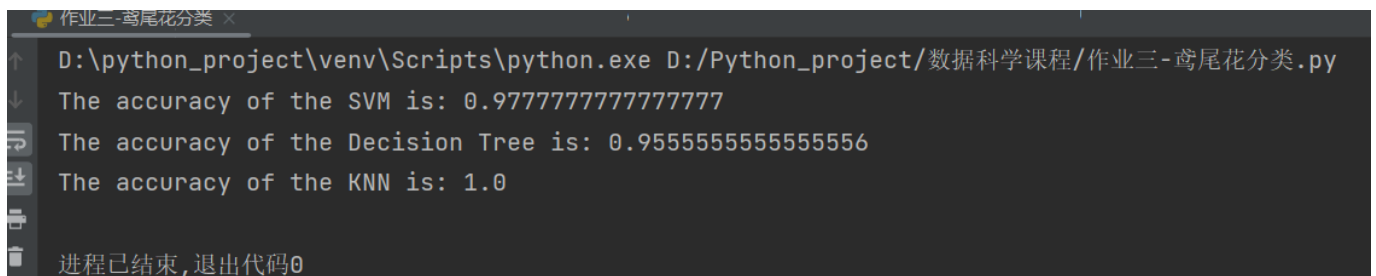
# Support Vector Machine
model = svm.SVC()
model.fit(train_X, train_y)
prediction = model.predict(test_X)
print('The accuracy of the SVM is:
{0}'.format(metrics.accuracy_score(prediction, test_y)))

# Decision Tree
model=DecisionTreeClassifier()
model.fit(train_X, train_y)
prediction = model.predict(test_X)
print('The accuracy of the Decision Tree is:
{0}'.format(metrics.accuracy_score(prediction, test_y)))

# K-Nearest Neighbours
model=KNeighborsClassifier(n_neighbors=3)
model.fit(train_X, train_y)
prediction = model.predict(test_X)
print('The accuracy of the KNN is:
{0}'.format(metrics.accuracy_score(prediction, test_y)))

```

三种方法的准确度运行结果,根据预测准确度来评价分类器性能,准确度高即性能好。



```

D:\python_project\venv\Scripts\python.exe D:/Python_project/数据科学课程/作业三-鸢尾花分类.py
The accuracy of the SVM is: 0.9777777777777777
The accuracy of the Decision Tree is: 0.9555555555555556
The accuracy of the KNN is: 1.0
进程已结束,退出代码0

```

4.可视化结果

