使用Python语言进行机器学习工作流的实例分析

王晓东

2018年11月2日，北京

中国科学院
计算机网络信息中心
Computer Network Information Center,
Chinese Academy of Sciences

目录

1-介绍

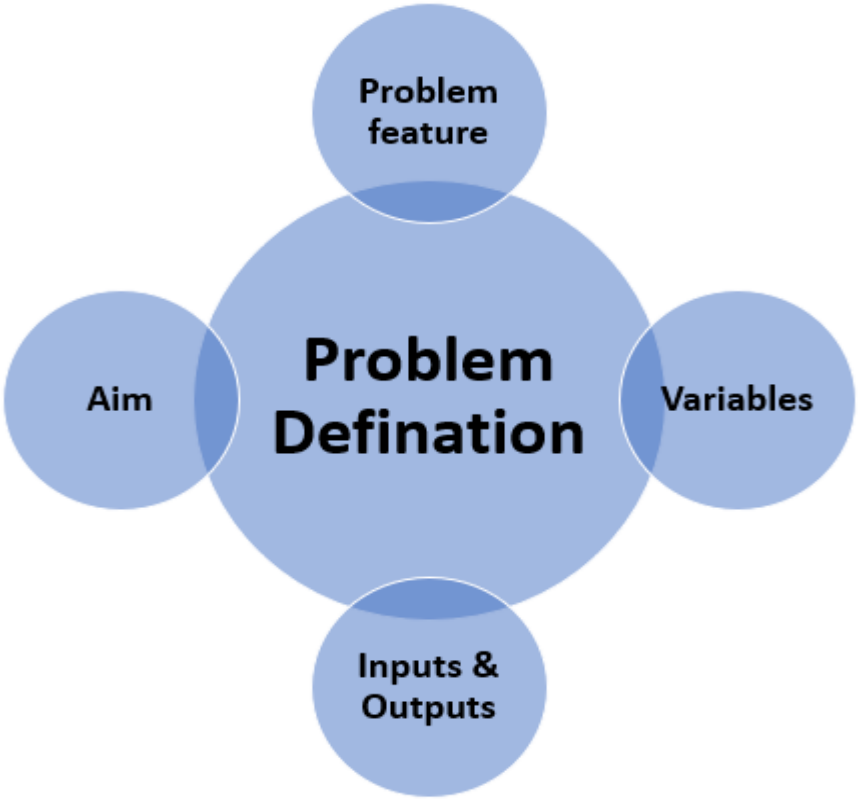
- IRIS(/’aɪrɪs/ 鸢yuān尾属植物)的全面ML技术
  - 关于IRIS数据集[1]的机器学习技术
  - 帮助指导机器学习的问题的处理过程
  - 用python包作为一个全面的工作流来解决一个简单的机器学习问题
  - 学习处理一般数据科学问题的工作流

2-机器学习工作流程

- 机器学习的工作流程
  - 定义问题
  - 指定输入和输出
  - 探索性数据分析
  - 数据采集
  - 数据预处理
  - 数据清理
  - 可视化
  - 模型设计
  - 模型部署
  - 模型维护，诊断

3-问题定义

- 当开始一个新的机器学习项目时，重要的事情之一是定义问题
- 问题定义有四个步骤，如下图所示：



3-1 问题特征

- 我们将使用经典的Iris数据集。该数据集包含有关三种不同类型鸢尾花的信息：
  - 变色鸢尾
  - Iris Virginica
  - Iris Setosa
- 数据集包含四个变量的度量：
  - 萼片长度
  - 萼片宽度
  - 花瓣长度
  - 花瓣宽度
- Iris数据集有许多有趣的特点：
  - 其中一个类（Iris Setosa）与其他两个类是线性可分的。但是，其他两个类不是线性可分的
  - Versicolor和Virginica类之间存在一些重叠，因此不太可能实现完美的分类
  - 四个输入变量中存在一些冗余，因此可以只用其中三个来实现一个好的解决方案，或者甚至（有困难）用两个，但是最佳变量的精确选择并不明显

3-2 Aim

- 目标是从萼片和花瓣的长度和宽度的测量中将虹膜花分为三个物种（setosa，versicolor或virginica）

3-3 Variables

- 变量是：
  - sepal_length：萼片长度，以厘米为单位，用作输入
  - sepal_width：用作输入的萼片宽度，以厘米为单位
  - petal_length：用作输入的花瓣长度，以厘米为单位
  - petal_width：用作输入的花瓣宽度，以厘米为单位
  - setosa：Iris setosa，真或假，用作目标
  - Iris versicolour，真或假，用作目标
  - virginica：Iris virginica，真或假，用作目标

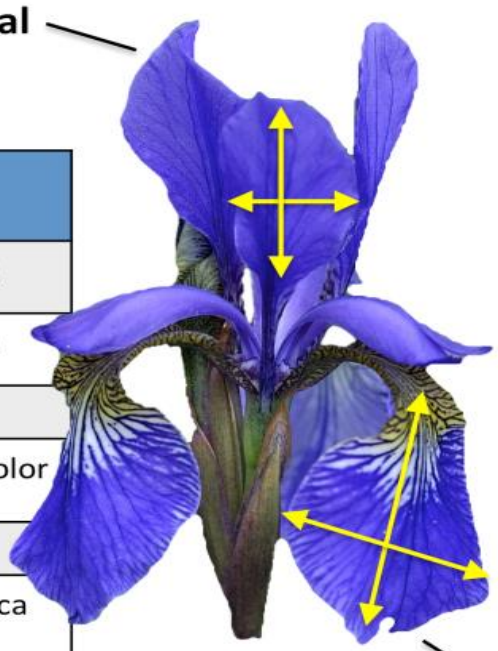
4-输入和输出

- 鸢尾花数据集简单介绍
  - Iris是机器学习中非常流行的分类问题，就像“Hello world”程序
  - 虹膜花数据集是多变量数据集，由英国统计学家和生物学家罗纳德·费希尔（Ronald Fisher）在其1936年的论文中介绍了在分类学问题中使用多种测量方法作为线性判别分析的一个例子
  - 它有时被称为安德森的数据集，因为埃德加安德森收集了数据来量化三个相关物种中鸢尾花的形态变异。这三个物种中的两个是在加斯佩半岛收集的，“所有这些都来自同一个牧场，并在同一天采摘并由同一个人用同一个设备同时测量”
  - 该数据集由来自三种鸢尾（Iris setosa，Iris virginica和Iris versicolor）中的每一种的50个样品组成
  - 从每个样品测量四个特征：萼片和花瓣的长度和宽度，以厘米为单位

4-输入和输出

Samples
(instances, observations)

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica



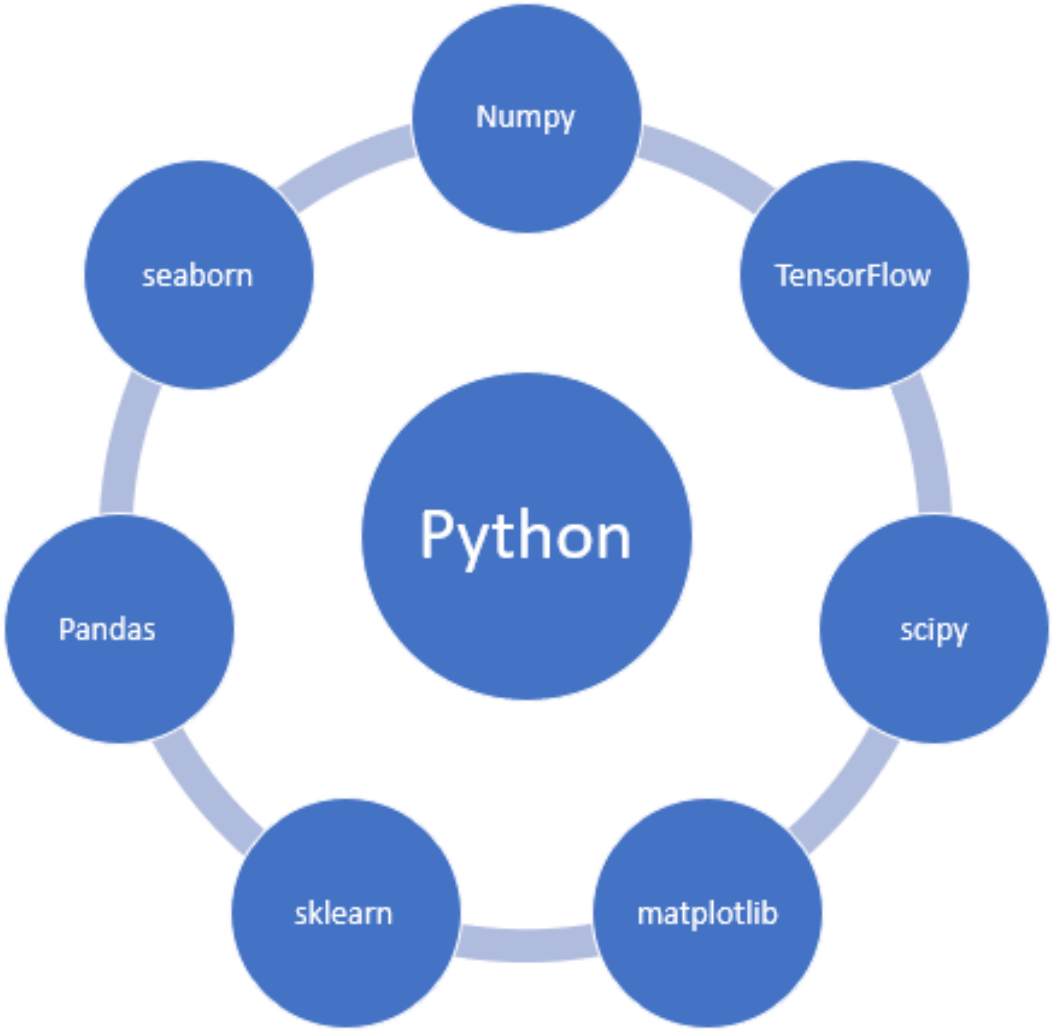
Features
(attributes, measurements, dimensions)

Class labels
(targets)

5 安装工具和依赖包

- Windows：
  - Anaconda（来自https://www.continuum.io）是SciPy的免费Python发行版。它也适用于Linux和Mac
  - Canopy（https://www.enthought.com/products/canopy/）免费提供商业发行版，并提供适用于Windows，Linux和Mac的完整SciPy
  - Python（x，y）是一个免费的Python发行版，包含SciPy和适用于Windows操作系统的Spyder IDE。（可从http://python-xy.github.io/下载）

5 安装工具和依赖包


Numpy
TensorFlow
scipy
matplotlib
sklearn
Pandas
seaborn
Python

5 安装工具和依赖包

In [1]:
```
# packages to load
# Check the versions of libraries
# Python version
import warnings
warnings.filterwarnings('ignore')
import sys
print('Python: {}'.format(sys.version))
# scipy
import scipy
print('scipy: {}'.format(scipy.__version__))
import numpy
# matplotlib
import matplotlib
print('matplotlib: {}'.format(matplotlib.__version__))
# numpy
import numpy as np # linear algebra
print('numpy: {}'.format(np.__version__))
# pandas
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
print('pandas: {}'.format(pd.__version__))
import seaborn as sns
print('seaborn: {}'.format(sns.__version__))
sns.set(color_codes=True)
import matplotlib.pyplot as plt
print('matplotlib: {}'.format(matplotlib.__version__))
%matplotlib inline
# scikit-learn
import sklearn
print('sklearn: {}'.format(sklearn.__version__))
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory
import os
%matplotlib inline
from sklearn.metrics import accuracy_score
# Importing metrics for evaluation
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

```
Python: 3.6.6 |Anaconda custom (64-bit)| (default, Jun 28 2018, 11:27:44) [MSC v.1900 64 bit (AMD64)]
scipy: 1.1.0
matplotlib: 2.2.3
numpy: 1.15.1
pandas: 0.20.3
seaborn: 0.9.0
matplotlib: 2.2.3
sklearn: 0.19.2
```

6-探索性数据分析

- 在本节中，学习如何使用图形和数值技术来探索数据的结构
  - 哪些变量暗示了有趣的关系
  - 哪些变量是不寻常的
- 分析和统计操作的基本步骤
  - 6-1 数据收集
  - 6-2 可视化
  - 6-3 数据预处理
  - 6-4 数据清理

6-1 数据搜集与简单探索

- 数据收集是收集和测量数据、信息或任何感兴趣的变量的过程，以标准化和确定的方式，使收集器能够回答或测试假设并评估特定集合的结果
- Iris数据集由3种不同类型的鸢尾花（Setosa、versicolor和virginica）组成，它的花瓣和萼片长度，储存在150x4的numpy.ndarray中。
- 读取数据

```
In [2]: # import Dataset to play with it
        dataset = pd.read_csv('../input/Iris.csv')
```

- 查看数据类型

```
In [3]: type(dataset)
Out[3]: pandas.core.frame.DataFrame
```

6-1 数据搜集与简单探索

- 数据形状

```
In [4]: # shape
        print(dataset.shape)

        (150, 6)

In [5]: #columns*rows
        dataset.size

Out[5]: 900
```

- 数据统计

```
In [6]: print(dataset.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
Id               150 non-null int64
SepalLengthCm    150 non-null float64
SepalWidthCm     150 non-null float64
PetalLengthCm    150 non-null float64
PetalWidthCm     150 non-null float64
Species          150 non-null object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.1+ KB
```

6-1 数据搜集与简单探索

- 数据内容

```
In [7]: dataset['Species'].unique()
Out[7]: array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)

In [8]: dataset["Species"].value_counts()
Out[8]: Iris-virginica     50
        Iris-versicolor    50
        Iris-setosa        50
        Name: Species, dtype: int64
```

- 数据样例

开头

```
In [9]: dataset.head(5)
```

Out[9]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

结尾

```
In [10]: dataset.tail()
```

随机

```
In [11]: dataset.sample(5)
```

6-1 数据搜集与简单探索

- 数据描述

```
In [12]: dataset.describe()
```

Out[12]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

- 数据操作

```
In [16]: dataset.where(dataset ['Species']='Iris-setosa')
```

Out[16]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1.0	5.1	3.5	1.4	0.2	Iris-setosa
1	2.0	4.9	3.0	1.4	0.2	Iris-setosa
2	3.0	4.7	3.2	1.3	0.2	Iris-setosa
3	4.0	4.6	3.1	1.5	0.2	Iris-setosa

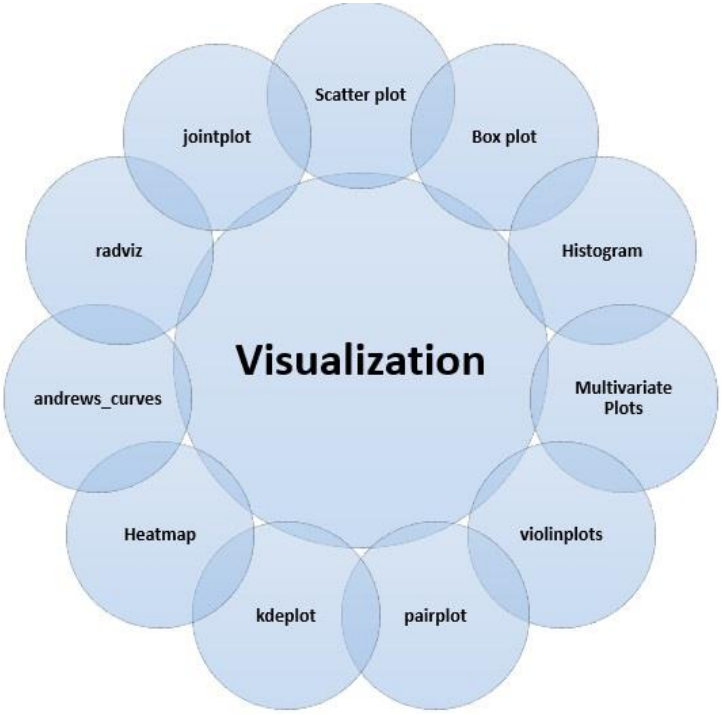
```
In [17]: dataset[dataset['SepalLengthCm']>7.2]
```

Out[17]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
105	106	7.6	3.0	6.6	2.1	Iris-virginica
107	108	7.3	2.9	6.3	1.8	Iris-virginica
117	118	7.7	3.8	6.7	2.2	Iris-virginica
118	119	7.7	2.6	6.9	2.3	Iris-virginica
122	123	7.7	2.8	6.7	2.0	Iris-virginica
130	131	7.4	2.8	6.1	1.9	Iris-virginica
131	132	7.9	3.8	6.4	2.0	Iris-virginica
135	136	7.7	3.0	6.1	2.3	Iris-virginica

6-2 可视化

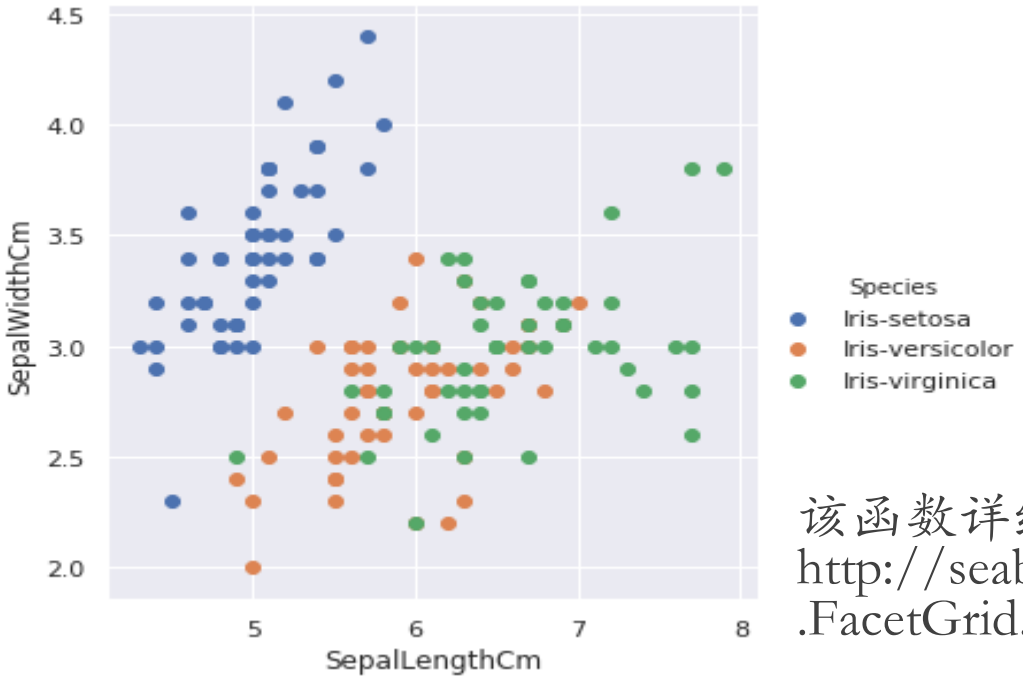
- IRIS的可视化技术
  - 数据可视化是用图形或图形格式表示数据。它使决策者能够直观地看到分析，这样他们就能掌握困难的概念或识别新的模式。通过交互式可视化，可以更进一步地使用技术，深入到图表和图表中，以获得更详细的信息，在这一节中将展示了多个使用matplotlib包绘制的图



6-2-1 散点图

- 散点图目的是确定两个量化变量之间的关系类型（如果有的话）

```
sns.FacetGrid(dataset, hue="Species", size=5) \
   .map(plt.scatter, "SepalLengthCm", "SepalWidthCm") \
   .add_legend()
plt.show()
```

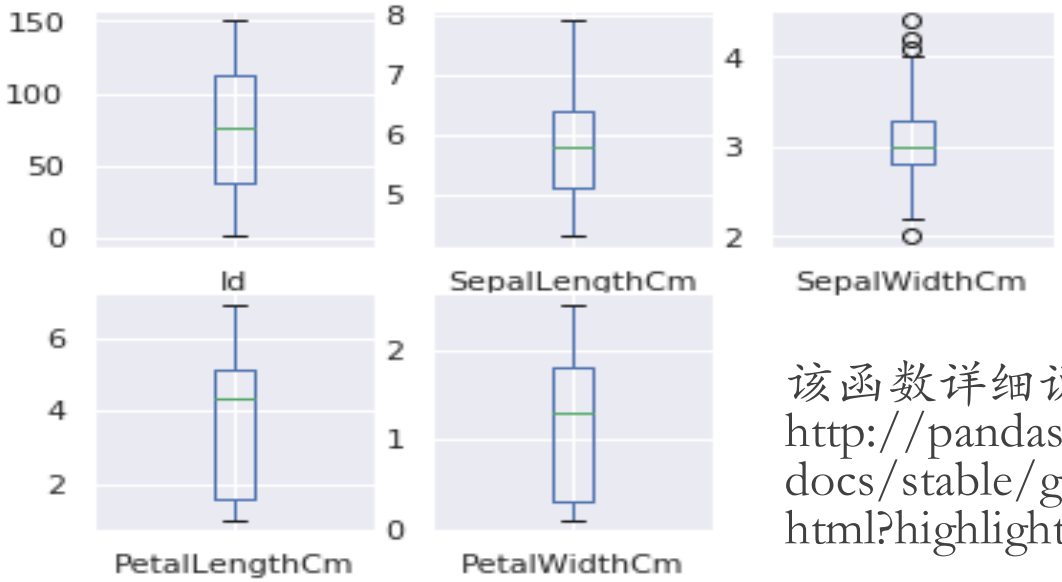


该函数详细说明参考：
http://seaborn.pydata.org/generated/seaborn.FacetGrid.map.html#seaborn.FacetGrid.map

6-2-2 盒图

- 在描述性统计中，一个箱形图或箱线图是一种通过它们的四分位图来图形化地描述数字数据组的方法。
- 箱形图也可能有从盒子（胡须）垂直延伸的线，表示在上和下四分位之外的变化，因此术语盒须图和盒须图。

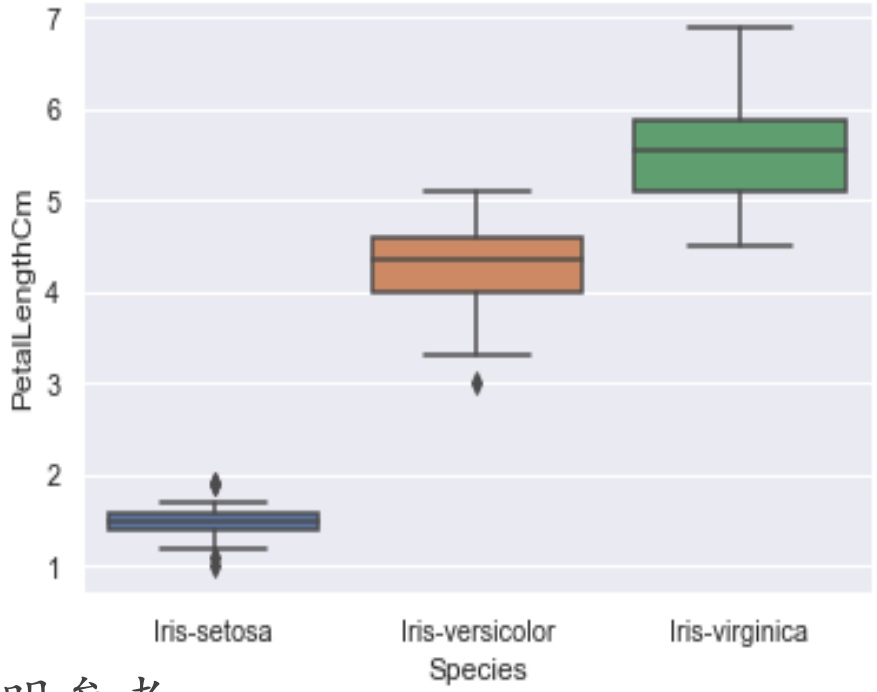
```
dataset.plot(kind='box', subplots=True, layout=(2,3), sharex=False, sharey=False)
plt.figure()
```



该函数详细说明参考：
http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html?highlight=plot#pandas.DataFrame.plot

6-2-2 盒图

In [20]:
```
# To plot the species data using a box plot:

sns.boxplot(x="Species", y="PetalLengthCm", data=dataset )
plt.show()
```

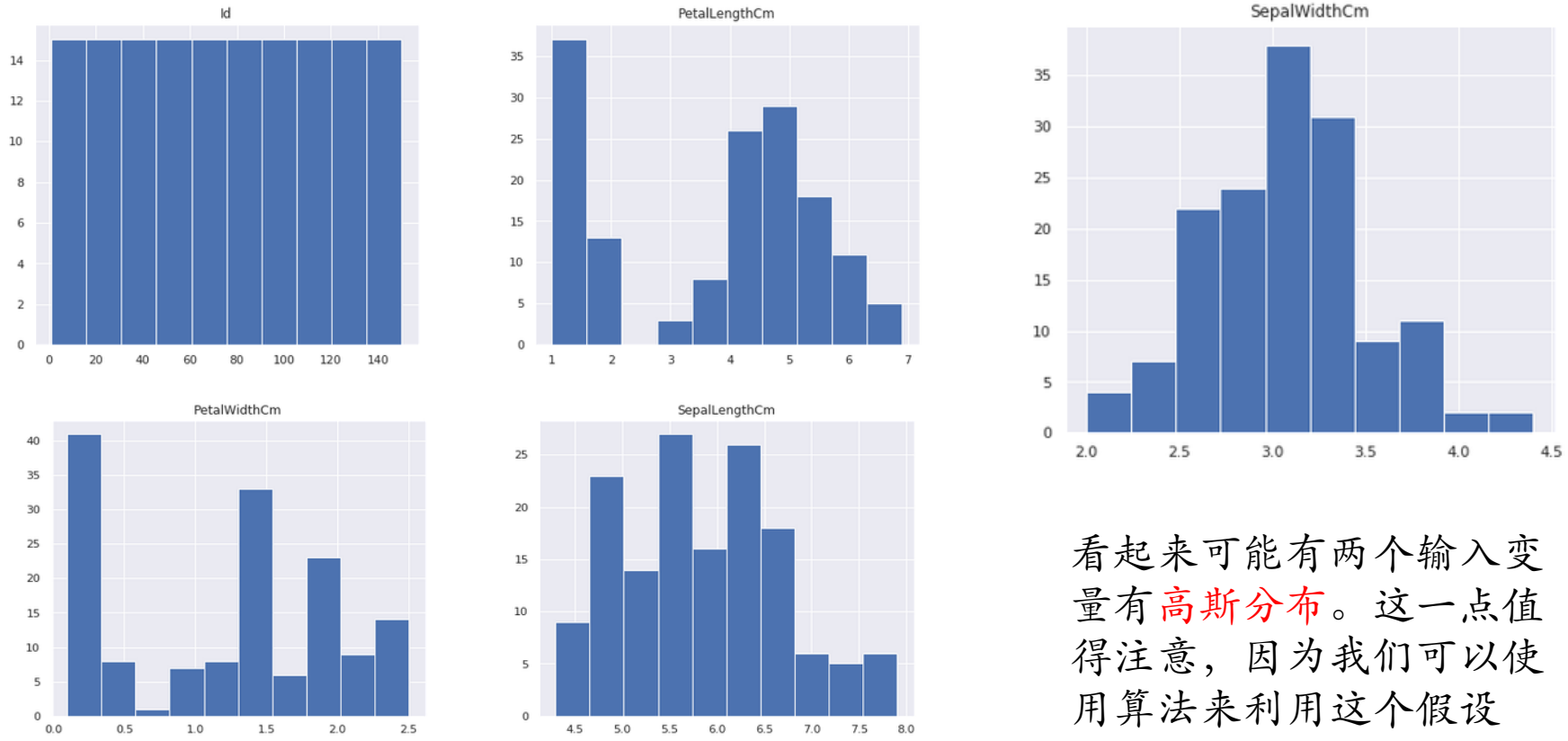


该函数详细说明参考：
http://seaborn.pydata.org/generated/seaborn.boxplot.html#seaborn.boxplot

6-2-3 条形图

- 我们也可以创建每个输入变量的直方图来获得分布的概念

```
dataset.hist(figsize=(15,20))
plt.figure()
```



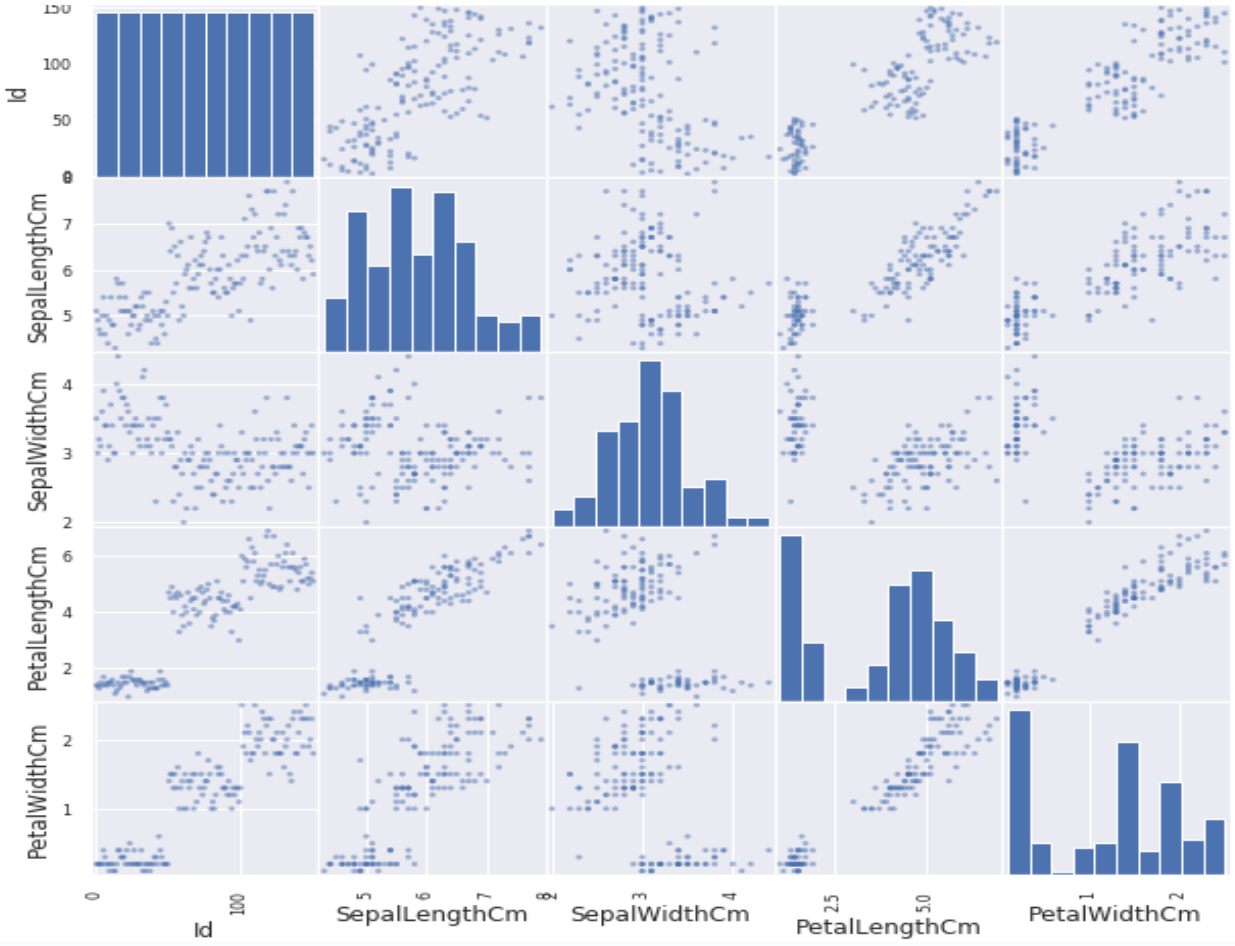
看起来可能有两个输入变量有高斯分布。这一点值得注意，因为我们可以使用算法来利用这个假设

该函数详细说明参考：http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.hist.html?highlight=hist#pandas.DataFrame.hist

6-2-4 双变量的散点图

- 对角线是直方图，其它的部分是两个变量之间的散点图

In [24]:
```
# scatter plot matrix
pd.plotting.scatter_matrix(dataset,figsize=(10,10))
plt.figure()
```



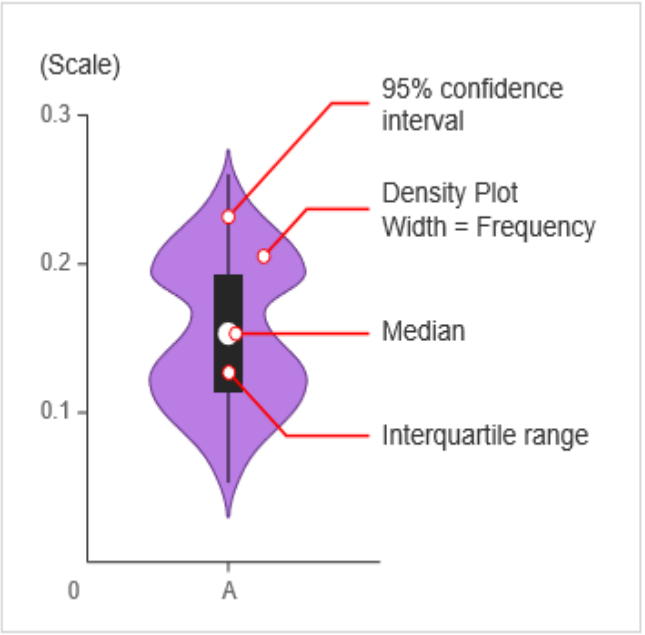
注意一些属性对的对角分组。这表明了一种高相关性和可预测的关系。

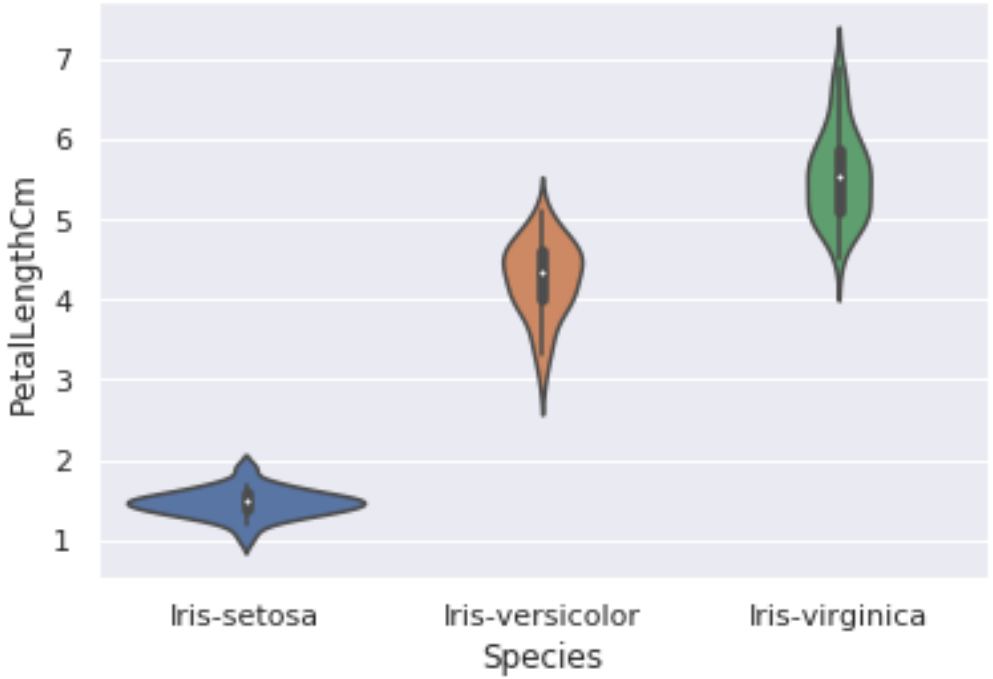
该函数详细说明参考：
http://pandas.pydata.org/pandas-docs/stable/generated/pandas.plotting.scatter_matrix.html?highlight=plotting

6-2-5 小提琴图

- 小提琴图又称核密度图，它是结合了箱形图和核密度图的图，将箱形图和密度图用一个图来显示，因为形状很像小提琴，所以被称作小提琴图。

```
sns.violinplot(data=dataset,x="Species", y="PetalLengthCm")
```



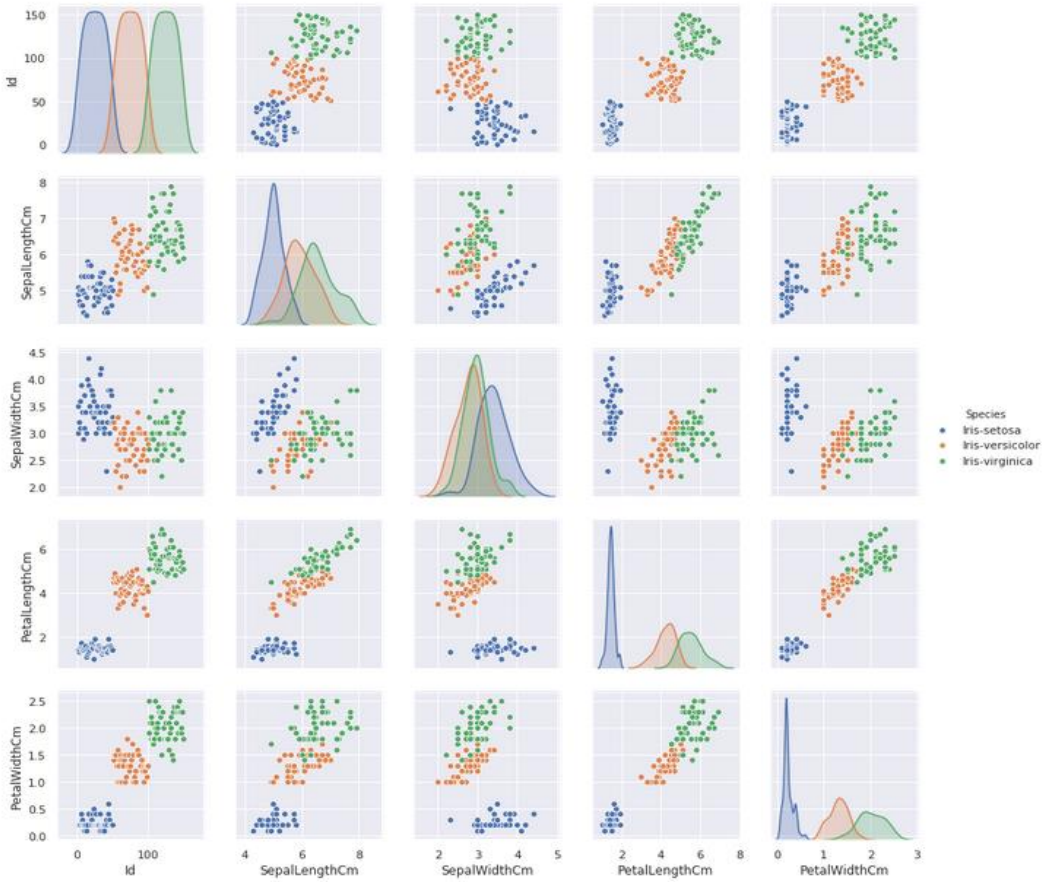


该函数详细说明参考：
http://seaborn.pydata.org/generated/seaborn.violinplot.html?highlight=violinplot#seaborn.violinplot

6-2-6 成对图

- 可以绘制成对图

```
In [26]: # Using seaborn pairplot to see the bivariate relation between each pair of features
         sns.pairplot(dataset, hue="Species")
```

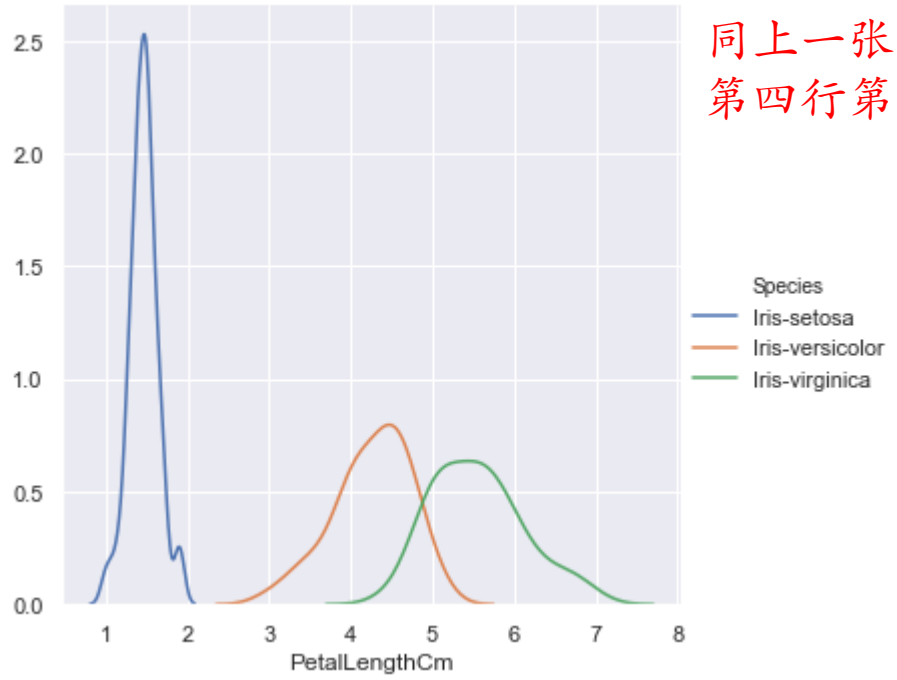


该函数详细说明参考：
http://seaborn.pydata.org/generated/seaborn.pairplot.html?highlight=pairplot#seaborn.pairplot

6-2-7 kdeplot

- 核密度估计(kernel density estimation)是在概率论中用来估计未知的密度函数，属于非参数检验方法之一，用于研究单变量关系

```
In [28]: # seaborn's kdeplot, plots univariate or bivariate density estimates.
         #Size can be changed by tweeking the value used
         sns.FacetGrid(dataset, hue="Species", size=5).map(sns.kdeplot, "PetalLengthCm").add_legend()
         plt.show()
```



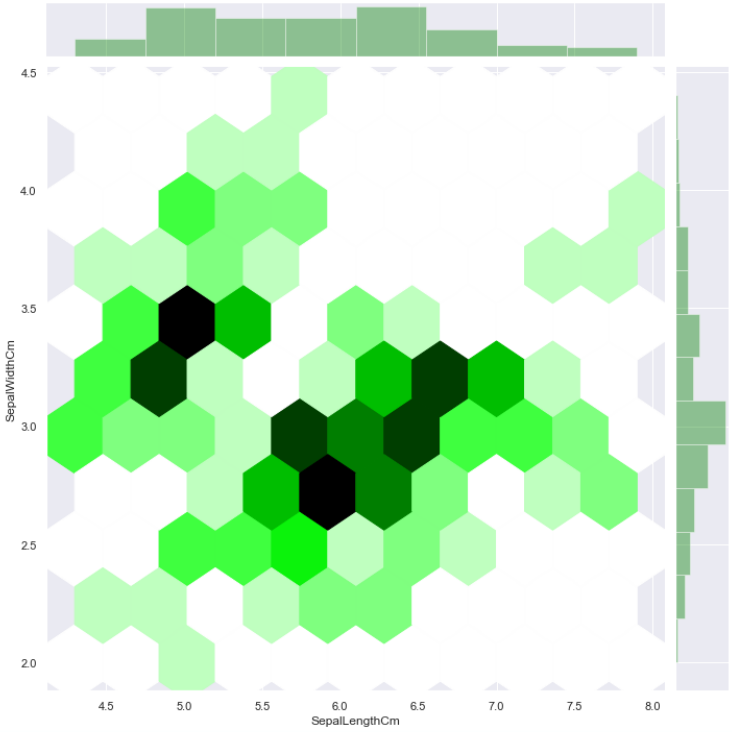
同上一张图的
第四行第四列

该函数详细说明参考：
http://seaborn.pydata.org/generated/seaborn.FacetGrid.html#seaborn.FacetGrid

6-2-8 jointplot

- 拟合并绘制一个六边箱图
  - 六边箱图又名高密度散点图，如果数据点太密集，绘制散点图太过密集，六边箱图是更好的选择。

In [29]:
```
# Use seaborn's jointplot to make a hexagonal bin plot
#Set desired size and ratio and choose a color.
sns.jointplot(x="SepalLengthCm", y="SepalWidthCm", data=dataset, size=10,ratio=10, kind='hex',color='green')
plt.show()
```



该函数详细说明参考：
http://seaborn.pydata.org/generated/seaborn.jointplot.html#seaborn.jointplot

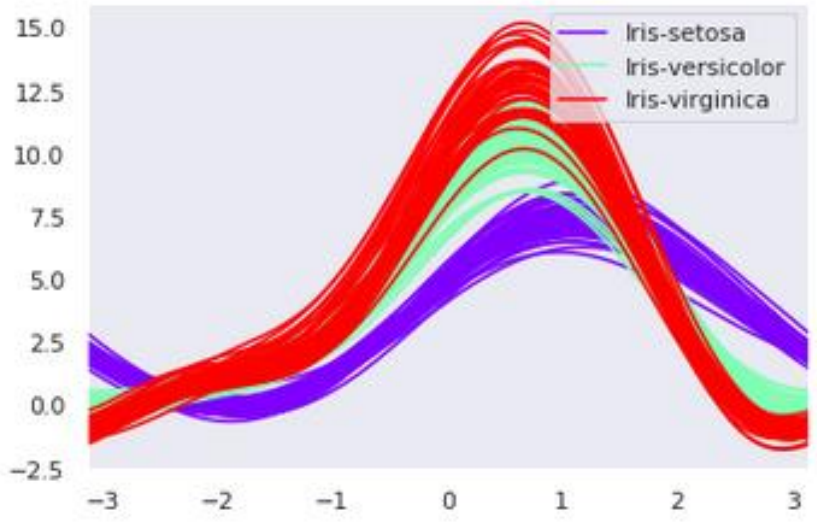
6-2-9 安德鲁斯曲线

- 安德鲁斯曲线[2]是在高维数据中可视化结构的一种方式
  - Andrews曲线将每个样本的属性值转化为傅里叶序列的系数来创建曲线
  - 通过将每一类曲线标成不同颜色可以可视化聚类数据，属于相同类别的样本的曲线通常更加接近并构成了更大的结构

每个点$x = \{x_1, x_2, \ldots, x_d\}$定义一个有限傅里叶序列

$$f(t) = \frac{x_1}{\sqrt{2}} + x_2 \sin(t) + x_3 \cos(t) + x_4 \sin(2t) + x_5 \cos(2t) + \ldots$$

```
from pandas.tools.plotting import andrews_curves
andrews_curves(dataset.drop("Id", axis=1), "Species",colormap='rainbow')
plt.show()
```



原理参考：
http://www.jucs.org/jucs_11_11/visualization_of_high_dimensional/jucs_11_11_1806_1819_garc_a_osorio.pdf

该函数详细说明参考：
http://pandas.pydata.org/pandas-docs/stable/generated/pandas.plotting.andrews_curves.html?highlight=andrews_curves

6-2-10 热图

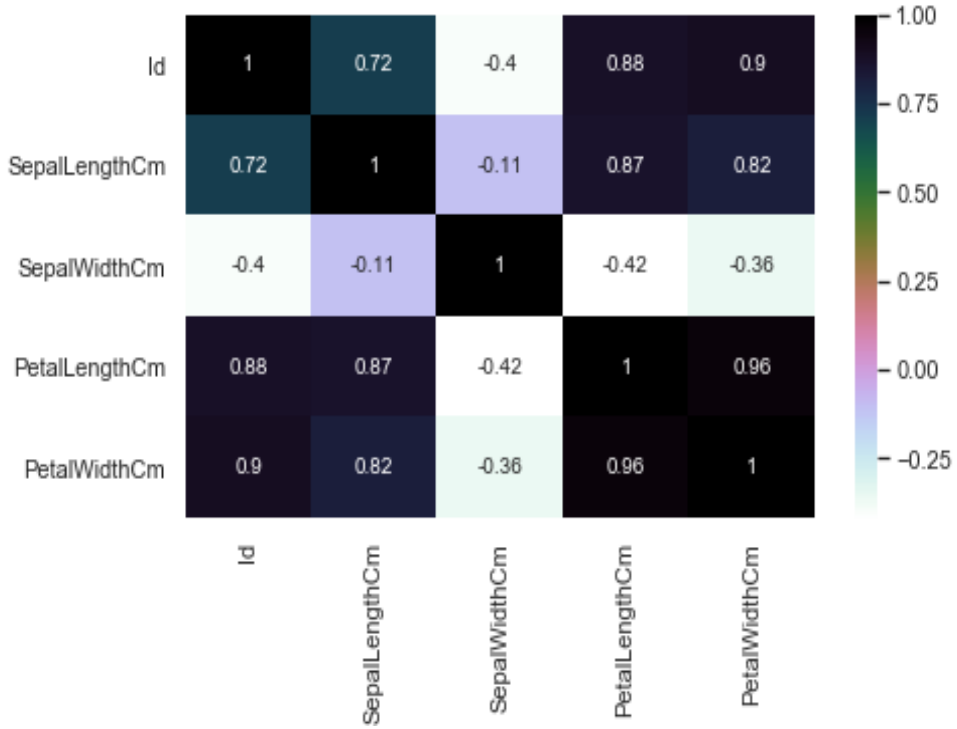
- 绘制热图

定义式 [1]

$$r(X,Y)=\frac{Cov(X,Y)}{\sqrt{Var[X]\,Var[Y]}}$$

其中，Cov(X,Y)为X与Y的协方差，Var[X]为X的方差，Var[Y]为Y的方差

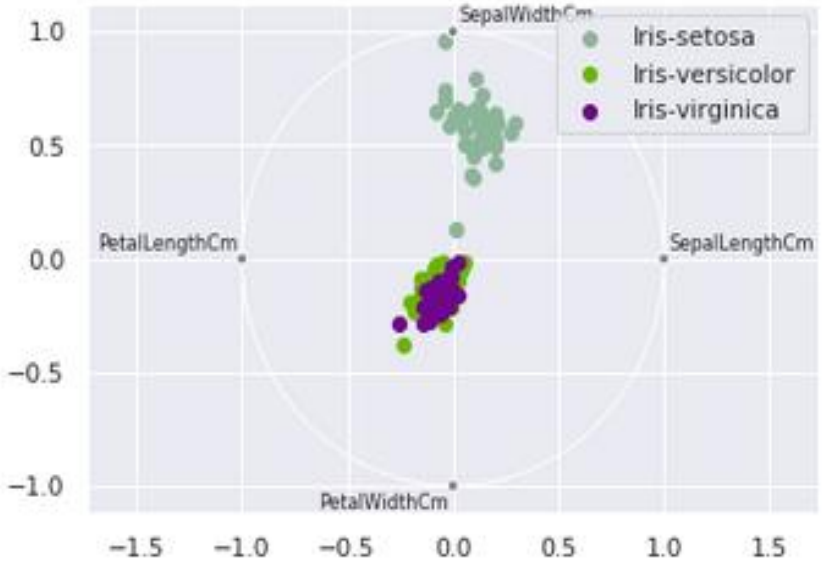
```
plt.figure(figsize=(7,4))
sns.heatmap(dataset.corr(),annot=True,cmap='cubehelix_r') #draws  heatmap with input as the cor
relation matrix calculted by(iris.corr())
plt.show()
```



该函数详细说明参考：
http://seaborn.pydata.org/generated/seaborn.heatmap.html?highlight=heatmap#seaborn.heatmap

6-2-11 雷达图

- 绘制雷达图
- radviz图也是一种多维的可视化图。它是基于基本的弹簧压力最小化算法，它将数据集的特征映射到二维目标空间单位圆中的一个点，点的位置由系在点上的特征决定。将实例投入到圆的中心，特征会朝园中此实例的位置（实例对应的归一化数值）“拉”实例



该函数详细说明参考：
http://pandas.pydata.org/pandas-docs/stable/generated/pandas.plotting.radviz.html?highlight=radviz#pandas.plotting.radviz

6-3 数据预处理

- 数据预处理是指在将数据输入到算法之前应用于我们的数据的转换，是一种用于将原始数据转换为规整的数据集的技术，数据预处理有很多步骤，这里我们只列出了其中的一些
  - 移除目标列（id）
  - 部分鸢尾花不平衡和平衡（采样不足）
  - 引入缺失的值并对其进行处理（以平均值替换）
  - 噪声过滤
  - 数据离散化标准化和标准化
  - PCA分析特性选择

6-4 数据清洗

- 数据清理的主要目标
  - 检测和消除错误和异常
  - 增加分析和决策中的数据的价值

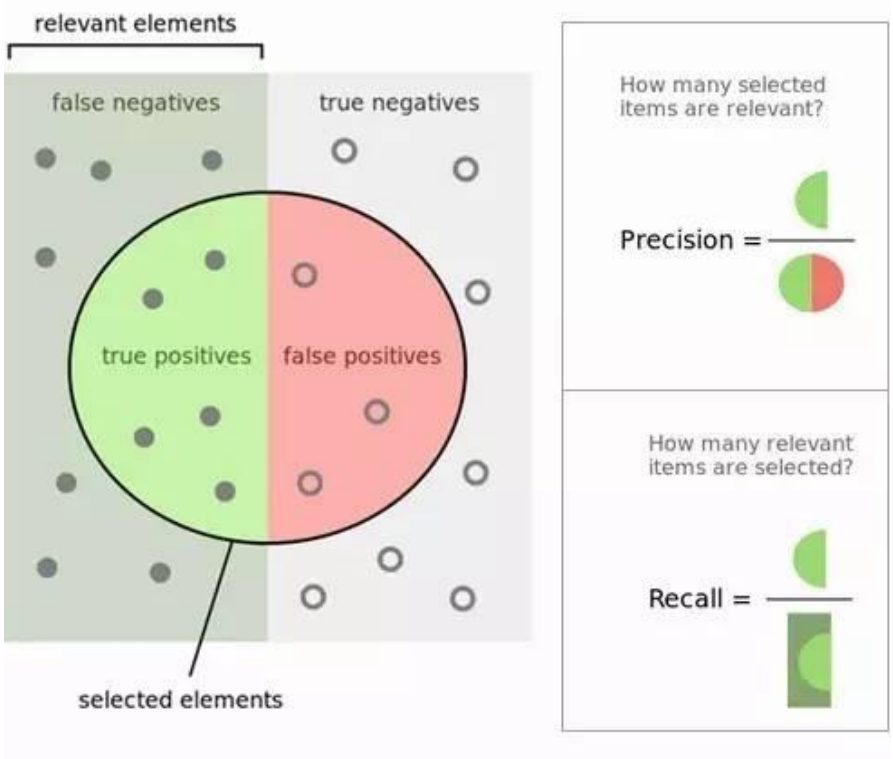
- 需要解决问题包括
  - 缺失值估算
  - 异常值检测

7 模型探索

- 在本节中，应用了将近20个学习算法

- 分类算法的评价方法

- TP, FP, TN, FN
  - True Positives,TP：预测为正样本，实际也为正样本的特征数
  - False Positives,FP：预测为正样本，实际为负样本的特征数
  - True Negatives,TN：预测为负样本，实际也为负样本的特征数
  - False Negatives,FN：预测为负样本，实际为正样本的特征数



7 模型探索

- 准确率
  - 精确率（Precision）的定义在上图可以看出，是绿色半圆除以红色绿色组成的圆。严格的数学定义如下：
  - $P = \frac{TP}{TP+FP}$
- 召回率
  - 召回率(Recall)的定义也在图上能看出，是绿色半圆除以左边的长方形。严格的数学定义如下：
  - $R = \frac{TP}{TP+FN}$
- F值
  - 有时也用一个F1值来综合评估精确率和召回率，它是精确率和召回率的调和均值。当精确率和召回率都高时，F1值也会高。严格的数学定义如下：
  - $\frac{2}{F_1} = \frac{1}{P} + \frac{1}{R}$

7 模型探索

- 常用评价指标、描述以及对应函数包位置

指标	描述	Scikit-learn函数
Precision	精准度	from sklearn.metrics import precision_score
Recall	召回率	from sklearn.metrics import recall_score
F1	F1值	from sklearn.metrics import f1_score
Confusion Matrix	混淆矩阵	from sklearn.metrics import confusion_matrix
ROC	ROC曲线	from sklearn.metrics import roc
AUC	ROC曲线下的面积	from sklearn.metrics import auc
precision	查准率	
recall	查全率	
P-R曲线	查准率为纵轴，查全率为横轴，作图	

7-1 K近邻算法

- k-近邻算法(k-NN)是一种用于分类和回归的机器学习方法
- k-近邻算法代码实验如下

In [36]:
```
# K-Nearest Neighbours
from sklearn.neighbors import KNeighborsClassifier

Model = KNeighborsClassifier(n_neighbors=8)
Model.fit(X_train, y_train)

y_pred = Model.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score

print('accuracy is',accuracy_score(y_pred,y_test))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00         6

    avg / total       1.00      1.00      1.00        30

[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
accuracy is 1.0
```

K近邻法算法原理参考：https://www.cnblogs.com/pinard/p/6061661.html

7-2 限定半径最近邻算法

- 在给定半径内实现邻居投票的分类器
- 有时候我们会遇到这样的问题，即样本中某系类别的样本非常的少，甚至少于K，这导致稀有类别样本在找K个最近邻的时候，会把距离其实较远的其他样本考虑进来，而导致预测不准确。为了解决这个问题，我们限定最近邻的一个最大距离，也就是说，我们只在一个距离范围内搜索所有的最近邻，这避免了上述问题。这个距离我们一般称为限定半径。

In [37]:
```
from sklearn.neighbors import  RadiusNeighborsClassifier
Model=RadiusNeighborsClassifier(radius=8.0)
Model.fit(X_train,y_train)
y_pred=Model.predict(X_test)
#summary of the predictions made by the classifier
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
#Accouracy score
print('accuracy is ', accuracy_score(y_test,y_pred))
```

```
                precision    recall  f1-score   support

   Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00         6

    avg / total       1.00      1.00      1.00        30

[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
accuracy is  1.0
```

K近邻法和限定半径最近邻法类库参数 https://www.cnblogs.com/pinard/p/6065607.html

7-3 逻辑回归(线性模型)

- 逻辑回归是在因变量为二分类(二元)时进行的合适的回归分析。像所有的回归分析一样，逻辑回归是预测分析

In [38]:
```
# LogisticRegression
from sklearn.linear_model import LogisticRegression
Model = LogisticRegression()
Model.fit(X_train, y_train)

y_pred = Model.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score
print('accuracy is',accuracy_score(y_pred,y_test))
```

```
                precision    recall  f1-score   support

   Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      0.85      0.92        13
 Iris-virginica       0.75      1.00      0.86         6

    avg / total       0.95      0.93      0.94        30

[[11  0  0]
 [ 0 11  2]
 [ 0  0  6]]
accuracy is 0.9333333333333333
```

逻辑回归算法原理参考：https://www.cnblogs.com/pinard/p/6029432.html

7-4 被动攻击分类 (线性模型)

- Passive Aggressive Classifier实验

In [39]:
```
from sklearn.linear_model import PassiveAggressiveClassifier
Model = PassiveAggressiveClassifier()
Model.fit(X_train, y_train)

y_pred = Model.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score
print('accuracy is',accuracy_score(y_pred,y_test))
```

```
                precision    recall  f1-score   support

    Iris-setosa       0.42      1.00      0.59        11
Iris-versicolor       0.00      0.00      0.00        13
 Iris-virginica       1.00      0.50      0.67         6

    avg / total       0.36      0.47      0.35        30

[[11  0  0]
 [13  0  0]
 [ 2  1  3]]
accuracy is 0.4666666666666667
```

被动攻击算法原理参考：http://scikit-learn.org/0.19/modules/linear_model.html#passive-aggressive

7-5 朴素贝叶斯

- 在机器学习中，朴素贝叶斯分类器是一组简单的“概率分类器”，基于贝叶斯定理，特征之间有很强的(朴素的)独立性假设，下面是高斯朴素贝叶斯实验:
- GaussianNB假设特征的先验概率为正态分布，即如下式:

$$P(X_j = x_j|Y = C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} exp(-\frac{(x_j - \mu_k)^2}{2\sigma_k^2})$$

```
In [40]: # Naive Bayes
from sklearn.naive_bayes import GaussianNB
Model = GaussianNB()
Model.fit(X_train, y_train)

y_pred = Model.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score
print('accuracy is',accuracy_score(y_pred,y_test))
```

```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00         6

    avg / total       1.00      1.00      1.00        30

[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
accuracy is 1.0
```

朴素贝叶斯算法原理参考：https://www.cnblogs.com/pinard/p/6069267.html

7-6 多项式朴素贝叶斯分类器

- 多项式朴素贝叶斯分类器实验
- MultinomialNB假设特征的先验概率为多项式分布，即如下式：

$$P(X_j = x_{jl}|Y = C_k) = \frac{x_{jl} + \lambda}{m_k + n\lambda}$$

In [41]:
```
# MultinomialNB
from sklearn.naive_bayes import MultinomialNB
Model = MultinomialNB()
Model.fit(X_train, y_train)

y_pred = Model.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score
print('accuracy is',accuracy_score(y_pred,y_test))
```

	precision	recall	f1-score	support
Iris-setosa	1.00	0.91	0.95	11
Iris-versicolor	0.83	0.77	0.80	13
Iris-virginica	0.62	0.83	0.71	6
avg / total	0.85	0.83	0.84	30

```
[[10  1  0]
 [ 0 10  3]
 [ 0  1  5]]
accuracy is 0.8333333333333334
```

多项式朴素贝叶斯算法原理参考：同上

7-7 伯努利朴素贝叶斯分类器

- 伯努利朴素贝叶斯分类器 实验
- BernoulliNB假设特征的先验概率为二元伯努利分布，即如下式：

$$P(X_j = x_{jl} | Y = C_k) = \frac{x_{jl} + \lambda}{m_k + n\lambda}$$

In [42]:
```
# BernoulliNB
from sklearn.naive_bayes import BernoulliNB
Model = BernoulliNB()
Model.fit(X_train, y_train)

y_pred = Model.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score
print('accuracy is',accuracy_score(y_pred,y_test))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       0.00      0.00      0.00        11
Iris-versicolor       0.00      0.00      0.00        13
 Iris-virginica       0.20      1.00      0.33         6

    avg / total       0.04      0.20      0.07        30

[[ 0  0 11]
 [ 0  0 13]
 [ 0  0  6]]
accuracy is 0.2
```

伯努利朴素贝叶斯算法原理参考：同上

朴素贝叶斯类算法小结

- 这三个类适用的分类场景各不相同
  - 如果样本特征的分布大部分是连续值，使用GaussianNB会比较好。
  - 如果如果样本特征的分大部分是多元离散值，使用MultinomialNB比较合适。
  - 如果样本特征是二元离散值或者很稀疏的多元离散值，应该使用BernoulliNB

7-8 支持向量机

- 基本支持向量机实验

In [43]:
```
# Support Vector Machine
from sklearn.svm import SVC

Model = SVC()
Model.fit(X_train, y_train)

y_pred = Model.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score

print('accuracy is',accuracy_score(y_pred,y_test))
```

```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00         6

    avg / total       1.00      1.00      1.00        30

[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
accuracy is 1.0
```

支持向量机算法原理参考：http://www.cnblogs.com/pinard/p/6097604.html

7-9 Nu-Support Vector Classification

- Nu支持向量机实验

In [44]:
```
# Support Vector Machine's
from sklearn.svm import NuSVC

Model = NuSVC()
Model.fit(X_train, y_train)

y_pred = Model.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score

print('accuracy is',accuracy_score(y_pred,y_test))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00         6

    avg / total       1.00      1.00      1.00        30

[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
accuracy is 1.0
```

NU支持向量机算法原理参考：http://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation

7-10 线性支持向量机

- 线性支持向量机实验

In [45]:
```
# Linear Support Vector Classification
from sklearn.svm import LinearSVC

Model = LinearSVC()
Model.fit(X_train, y_train)

y_pred = Model.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score

print('accuracy is',accuracy_score(y_pred,y_test))
```

```
                precision    recall  f1-score   support

   Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       0.72      1.00      0.84        13
 Iris-virginica       1.00      0.17      0.29         6

    avg / total       0.88      0.83      0.79        30

[[11  0  0]
 [ 0 13  0]
 [ 0  5  1]]
accuracy is 0.8333333333333334
```

支持向量机算法原理参考：同上

支持向量机小结

- 支持向量机的优点是:
  - 在高维空间中有效
  - 在维度数量大于样本数量的情况下仍然有效
  - 可以为决策函数指定不同的核函数。提供了通用的核，但也可以指定定制的核
- 支持向量机的缺点是:
  - 如果特征数远大于样本数，则在选择核函数时避免过拟合，正则项至关重要

7-11 决策树

- 决策树是一种用于分类和回归的非参数监督学习方法。目标是创建一个模型，通过学习从数据特性推断出的简单决策规则来预测目标变量的值

In [46]:
```
# Decision Tree's
from sklearn.tree import DecisionTreeClassifier

Model = DecisionTreeClassifier()

Model.fit(X_train, y_train)

y_pred = Model.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score
print('accuracy is',accuracy_score(y_pred,y_test))
```

```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       0.93      1.00      0.96        13
 Iris-virginica       1.00      0.83      0.91         6

    avg / total       0.97      0.97      0.97        30

[[11  0  0]
 [ 0 13  0]
 [ 0  1  5]]
accuracy is 0.9666666666666667
```

决策树算法原理参考：https://www.cnblogs.com/pinard/p/6050306.html

7-12 额外决策树

- 额外决策树实验

In [47]:
```
# ExtraTreeClassifier
from sklearn.tree import ExtraTreeClassifier

Model = ExtraTreeClassifier()

Model.fit(X_train, y_train)

y_pred = Model.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score
print('accuracy is',accuracy_score(y_pred,y_test))
```

```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00         6

    avg / total       1.00      1.00      1.00        30

[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
accuracy is 1.0
```

决策树小结

- 决策树算法的优点：
  - 简单直观，生成的决策树很直观
  - 基本不需要预处理，不需要提前归一化，处理缺失值
  - 使用决策树预测的代价是O(log2m)。 m为样本数
  - 既可以处理离散值也可以处理连续值。很多算法只是专注于离散值或者连续值
  - 可以处理多维度输出的分类问题
  - 相比于神经网络之类的黑盒分类模型，决策树在逻辑上可以得到很好的解释
- 我们再看看决策树算法的缺点：
  - 决策树算法非常容易过拟合，导致泛化能力不强。可以通过设置节点最少样本数量和限制决策树深度来改进
  - 决策树会因为样本发生一点点的改动，就会导致树结构的剧烈改变。这个可以通过集成学习之类的方法解决
  - 寻找最优的决策树是一个NP难的问题，我们一般是通过启发式方法，容易陷入局部最优。可以通过集成学习之类的方法来改善
  - 有些比较复杂的关系，决策树很难学习，比如异或。这个就没有办法了，一般这种关系可以换神经网络分类方法来解决

7-13 神经网络

- 即多层感知器分类器
- 模型采用随机梯度下降等方法对对数损失函数进行优化

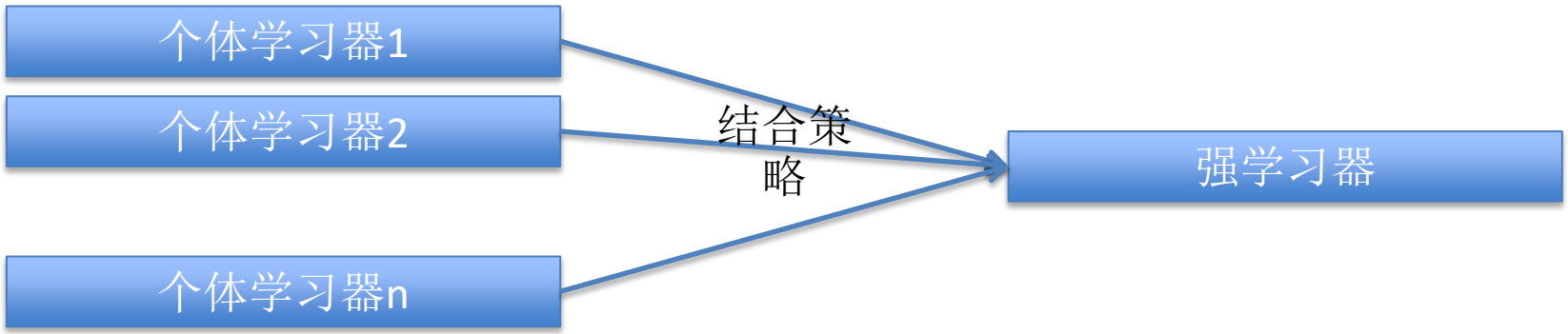
```
In [48]: from sklearn.neural_network import MLPClassifier
         Model=MLPClassifier()
         Model.fit(X_train,y_train)
         y_pred=Model.predict(X_test)
         # Summary of the predictions
         print(classification_report(y_test,y_pred))
         print(confusion_matrix(y_test,y_pred))
         #Accuracy Score
         print('accuracy is ',accuracy_score(y_pred,y_test))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       0.00      0.00      0.00        11
Iris-versicolor       0.43      1.00      0.60        13
 Iris-virginica       0.00      0.00      0.00         6

    avg / total       0.19      0.43      0.26        30

[[ 0 11  0]
 [ 0 13  0]
 [ 0  6  0]]
accuracy is  0.43333333333333335
```

该函数详细说明参考：http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier

集成学习

- 集成学习概述
  - 从下图，我们可以对集成学习的思想做一个概括。对于训练集数据，我们通过训练若干个个体学习器，通过一定的结合策略，就可以最终形成一个强学习器，以达到博采众长的目的。



- 集成学习之boosting
  - Boosting算法的工作机制是首先从训练集用初始权重训练出一个弱学习器1，根据弱学习的学习误差率表现来更新训练样本的权重，使得之前弱学习器1学习误差率高的训练样本点的权重变高，使得这些误差率高的点在后面的弱学习器2中得到更多的重视。然后基于调整权重后的训练集来训练弱学习器2.，如此重复进行，直到弱学习器数达到事先指定的数目T，最终将这T个弱学习器通过集合策略进行整合，得到最终的强学习器（AdaBoost、GDBT）
- 集成学习之bagging
  - Bagging的算法原理和boosting不同，它的弱学习器之间没有依赖关系，可以并行生成（随机森林、bagging）

7-14 随机森林

- 随机森林的实验

```
In [49]: from sklearn.ensemble import RandomForestClassifier
         Model=RandomForestClassifier(max_depth=2)
         Model.fit(X_train,y_train)
         y_pred=Model.predict(X_test)
         print(classification_report(y_test,y_pred))
         print(confusion_matrix(y_pred,y_test))
         #Accuracy Score
         print('accuracy is ',accuracy_score(y_pred,y_test))
```

```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00         6

    avg / total       1.00      1.00      1.00        30

[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
accuracy is  1.0
```

随机森林算法原理参考：https://www.cnblogs.com/pinard/p/6156009.html

7-15 Bagging

- Bagging的实验

```
In [50]: from sklearn.ensemble import BaggingClassifier
         Model=BaggingClassifier()
         Model.fit(X_train,y_train)
         y_pred=Model.predict(X_test)
         print(classification_report(y_test,y_pred))
         print(confusion_matrix(y_pred,y_test))
         #Accuracy Score
         print('accuracy is ',accuracy_score(y_pred,y_test))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       0.93      1.00      0.96        13
 Iris-virginica       1.00      0.83      0.91         6

    avg / total       0.97      0.97      0.97        30

[[11  0  0]
 [ 0 13  1]
 [ 0  0  5]]
accuracy is  0.9666666666666667
```

Bagging算法原理参考：https://www.cnblogs.com/pinard/p/6156009.html

7-16 自适应提示分类器Adaboost

- 自适应提示分类器实验

In [51]:
```
from sklearn.ensemble import AdaBoostClassifier
Model=AdaBoostClassifier()
Model.fit(X_train,y_train)
y_pred=Model.predict(X_test)
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_pred,y_test))
#Accuracy Score
print('accuracy is ',accuracy_score(y_pred,y_test))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       0.93      1.00      0.96        13
 Iris-virginica       1.00      0.83      0.91         6

    avg / total       0.97      0.97      0.97        30

[[11  0  0]
 [ 0 13  1]
 [ 0  0  5]]
accuracy is  0.9666666666666667
```

Adaboost算法原理参考：https://www.cnblogs.com/pinard/p/6133937.html

7-17 梯度提升分类器GradientBoosting

- 梯度提升分类器（GBDT）实验

In [52]:
```
from sklearn.ensemble import GradientBoostingClassifier
Model=GradientBoostingClassifier()
Model.fit(X_train,y_train)
y_pred=Model.predict(X_test)
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_pred,y_test))
#Accuracy Score
print('accuracy is ',accuracy_score(y_pred,y_test))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       0.93      1.00      0.96        13
 Iris-virginica       1.00      0.83      0.91         6

    avg / total       0.97      0.97      0.97        30

[[11  0  0]
 [ 0 13  1]
 [ 0  0  5]]
accuracy is  0.9666666666666667
```

梯度提升树(GBDT)算法原理参考：https://www.cnblogs.com/pinard/p/6140514.html

7-18 线性判别分析LDA

- LDA实验

In [53]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
Model=LinearDiscriminantAnalysis()
Model.fit(X_train,y_train)
y_pred=Model.predict(X_test)
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_pred,y_test))
#Accuracy Score
print('accuracy is ',accuracy_score(y_pred,y_test))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00         6

    avg / total       1.00      1.00      1.00        30

[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
accuracy is  1.0
```

线性判别分析LDA算法原理参考：https://www.cnblogs.com/pinard/p/6244265.html

7-19 二次判别分析QDA

- QDA实验

```
In [54]: from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
Model=QuadraticDiscriminantAnalysis()
Model.fit(X_train,y_train)
y_pred=Model.predict(X_test)
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_pred,y_test))
#Accuracy Score
print('accuracy is ',accuracy_score(y_pred,y_test))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00         6

    avg / total       1.00      1.00      1.00        30

[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
accuracy is  1.0
```

二次判别分析QDA算法原理参考：http://scikit-learn.org/stable/modules/lda_qda.html#dimensionality-reduction-using-linear-discriminant-analysis

8 总结

- 在这个PPT中，用各种Python包覆盖ML过程中所有相关的部分
- 实际Kaggle比赛使用的方法
  - Gradient Boosting 本身优秀的性能加上 Xgboost 高效的实现，使得它在 Kaggle 上广为使用
  - 几乎每场比赛的获奖者都会用 Xgboost 作为最终 Model 的重要组成部分
  - 在实战中，往往会以 Xgboost 为主来建立我们的模型并且验证 Feature 的有效性
- 实践代码简述与思考
  - 方法加入部分代码

```
MethodName = {}
# 1. 最近邻算法 (http://scikit-learn.org/stable/modules/neighbors.html)
## 1.1 K近邻算法 (http://scikit-learn.org/stable/modules/neighbors.html#classification)
from sklearn.neighbors import KNeighborsClassifier
MethodName.update({'KNeighborsClassifier':{'n_neighbors':8}})
## 1.2 固定半径法 (方法说明网址同上)
from sklearn.neighbors import  RadiusNeighborsClassifier
MethodName.update({'RadiusNeighborsClassifier':{'radius':8.0}})
# 2. 线性模型 (http://scikit-learn.org/stable/modules/linear_model.html
```

8 总结

- 调用方法部分代码

```
def testAllMethod():
    for s,v in MethodName.items():
        printTip('使用'+s+'方法的结果如下：')
        Model = eval(s)(**v)
        Model.fit(X_train, y_train)
        y_pred = Model.predict(X_test)
        # Summary of the predictions made by the classifier
        print(classification_report(y_test, y_pred))
        print(confusion_matrix(y_test, y_pred))
        # Accuracy score
        print('accuracy is',accuracy_score(y_test,y_pred))
```

- 主函数部分代码

```
if __name__ == '__main__':
    dataset = pd.read_csv('Iris.csv')   #读取数据

    X = dataset.iloc[:, :-1].values
    y = dataset.iloc[:, -1].values

    # Splitting the dataset into the Training set and Test set
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

    testAllMethod()
```

9 参考资料

[1] Fisher R A. THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS[J]. Annals of Human Genetics, 2012, 7(2):179-188.

[2] Andrews D F. Plots of High-Dimensional Data[J]. Biometrics, 1972, 28(1):125-136.

Thank you!

中国科学院
计算机网络信息中心
Computer Network Information Center,
Chinese Academy of Sciences