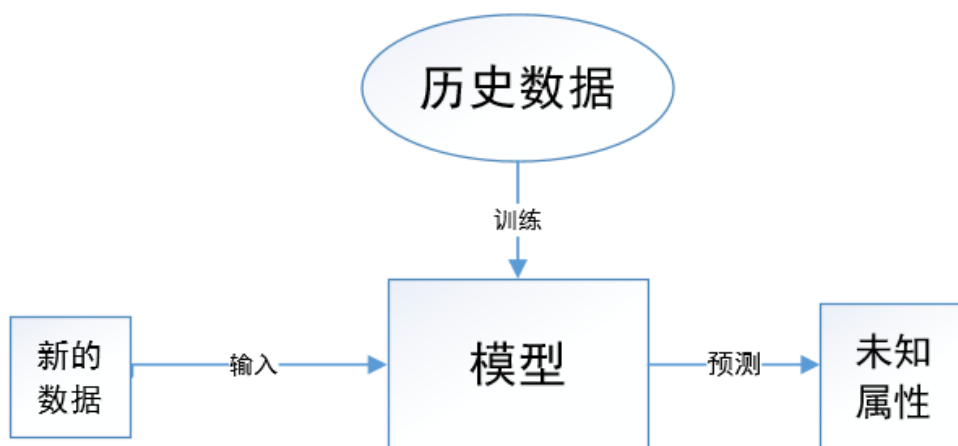


Python 机器学习

绪论

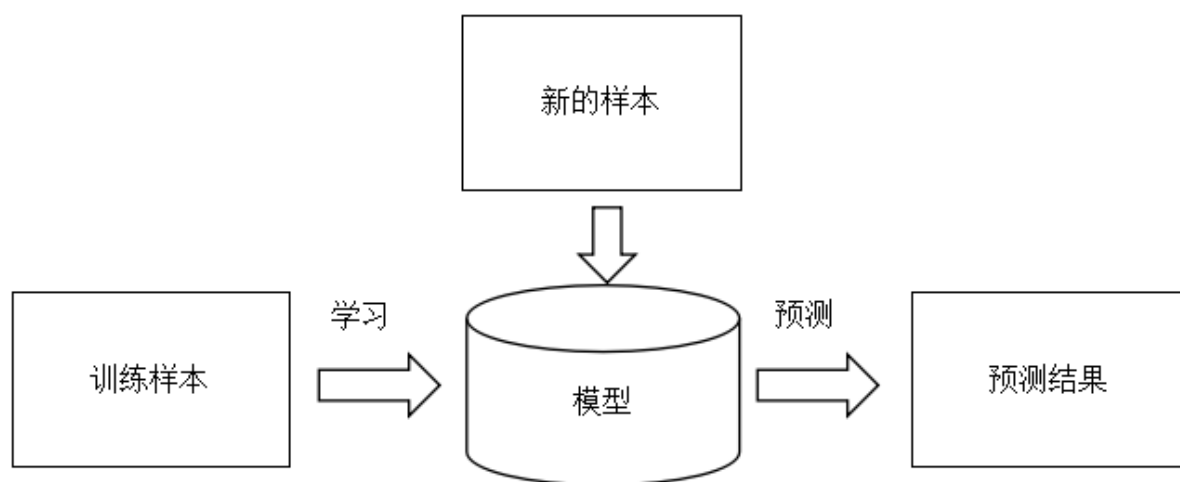
机器学习的定义

机器学习是从人工智能中产生的一个重要学科分支，是实现智能化的关键。



基本概念

算法分类



- 有监督学习
 - 样本带有标签值，称为监督信号
训练样本为 (x, y) , x 为特征向量, y 为标签值
 - 有学习过程，根据训练样本学习，得到模型，然后用于预测
机器学习模型实现从特征向量到标签值得映射 $y = f(x)$
 - 按照标签值得类型可以进一步分为两类
 - 分类问题 —— 标签值为整数编号

例如手写数字的识别： 十分类问题

- **回归问题 —— 标签值为实数**（例如预测一个人的收入）
 - 可以分为 生成模型 与 判别模型

类别	算法
生成模型	贝叶斯分类器；高斯混合模型；贝叶斯网络；隐马尔可夫模型；受限玻尔兹曼机；生成对抗网络；变分自动编码器
判别模型	决策树；KNN算法；人工神经网络；支持向量机；logistic回归；softmax回归；随机森林；boosting算法；条件随机场

• 无监督学习

样本没有标签值，没有训练过程，机器学习算法直接对样本进行处理，得到某种结果

无监督学习算法分类：

- **聚类问题** —— 无监督分类，将一组样本划分成多个子集
- **数据降维问题** —— 将向量映射到更低维的空间

将n维空间中的向量 \mathbf{x} 通过某种映射函数映射到更低维的m维空间中

$$\mathbf{y} = \phi(\mathbf{x}), m \ll n$$

• 半监督学习（介于有监督学习与无监督学习之间）

有些训练样本有标签值，有些没有标签值，用这些样本进行训练得到模型，然后用于预测。

• 强化学习

模拟人的行为，源自于行为主义心理学，确定在每种状态下要执行的动作，以达到某种目标。

过拟合与欠拟合

训练集上的表现	测试集上的表现	结论	原因
不好	不好	欠拟合：也称为欠学习，指模型在训练集上的精度差	模型简单 特征数太少
好	不好	过拟合：也称为过学习，指模型在训练集上表精度高，但在测试集上精度低，泛化性能差	模型过于复杂；训练样本太少，缺乏代表性；训练样本噪声的干扰
好	好	适度拟合	

机器学习涉及到的数学基础

- 导数
- 向量与矩阵
 - 向量的范数：

$$||\boldsymbol{x}||_p = \left(\sum_{i=1}^n |x_i|^p\right)^{\frac{1}{p}}$$

L1范数：所有分量的绝对值之和

$$||\boldsymbol{x}||_1 = \sum_{i=1}^n |x_i|$$

L2范数也称为向量的模，即向量的长度

$$||\boldsymbol{x}||_2 = \sqrt{\sum_{i=1}^n |x_i|^2}$$

- 偏导与梯度
- 雅克比矩阵
- Hessian矩阵
- 泰勒公式
- 行列式
- 特征值和特征向量
- 二次型

- 迭代法
- 梯度下降法
- 牛顿法
- 拉格朗日乘数法
- 凸优化问题
- 拉格朗日对偶

概率论：

- 随机事件
- 条件概率
- 随机变量 [离散型随机变量 连续型随机变量]
- 数学期望
- 方差与标准差
- 随机向量

贝叶斯

- 训练数据集：模拟学习问题的输入和输出
- 决策函数集：备选决策函数集合（可行域）
- 表现度量：目标函数
- 优化算法：计算优化问题最优解
- 超参数调整：目标函数与优化算法中的超参数调整

最大似然估计（MLE）和最大后验估计（MAP）

- 训练数据集：模拟学习问题的输入和输出
- 决策函数集：概率模型
- 表现度量：MLE/MAP
- 优化算法：？
- 超参数调整：？

最大似然估计 MLE

- 训练数据集：模拟学习问题的输入和输出
- 决策函数集： $p(y|x;w)$
- 表现度量：

$$\max_w p(Y|X;w)$$

- 优化算法：？
- 超参数调整：？

贝叶斯公式

链式法则：

$$P(X,Y) = P(X|Y)P(Y) = P(Y|X)P(X)$$

贝叶斯规则：

$$P(X|Y) = \frac{P(X|Y)P(X)}{P(Y)}$$

最大后验估计 MAP

- 训练数据集：模拟学习问题的输入和输出
- 决策函数集： $p(y|x;w)p(w)$
- 表现度量：

$$\max_w p(Y|X;w)p(w)$$

- 优化算法：概率推断
- 超参数调整：？

K-近邻分类（KNN）

一、KNN 算法简介

- KNN（k-Nearest Neighbor），也称K-近邻分类算法。分类的目的是学会一个**分类器**。该分类器能把数据映射到事先给定类别中的某一个类别。分类属于一种**监督学习**方式，分类器的学习是在被告知每一个**训练样本**属于哪个类别后进行的。每个训练样本都有一个特定的标签，与之相对应。在学习过程中，从这些给定的训练数据集中**学习一个函数**。等新的数据到来时，可以根据这个函数判断结果。
- 监督学习的训练及要求，包括**输入和输出**。也可以说是既有有数据的特征，也要有对应的目标，训练集中的目标是由人们事先进行标注的。
- 通过测量不同特征值之间的距离进行分类的
- 基本思路：**如果一个样本在特征空间中的k个最邻近样本中的大多数属于某一个类别，则该样本也属于这个类别**，该算法在决定类别上只依据最近的一个或几个样本的类别来决定待分类样本所属的类别，KNN算法中所选择的**邻居都是已经正确分类的对象**

二、KNN算法大致步骤

1. 算距离：给定测试对象，计算它与训练集中的每个对象的距离；
2. 找邻居：圈定距离最近的k个训练对象，作为测试对象的近邻；
3. 做分类：根据这k个近邻归属的**主要类别**，来对测试对象分类。

三、相似度度量方法

闵可夫斯基距离：

假设数值点 P 和 Q 坐标为 $P(x_1, x_2, \dots, x_n)$ 和 $Q(y_1, y_2, \dots, y_n)$ ，则闵可夫斯基距离为：

$$\left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

- $p = 1$ 曼哈顿距离

$$\sum_{i=1}^n |x_i - y_i|$$

- $p = 2$ 欧几里得距离

$$\sqrt{\sum_{i=1}^n |x_i - y_i|^2}$$

- $p = \infty$ 切比雪夫距离

$$\left(\sum_{i=1}^n |x_i - y_i|^p\right)^{\frac{1}{p}} = \max_i |x_i - y_i|$$

四、KNN一般流程

1. 计算测试数据与各个训练数据之间的距离
2. 按照距离的递增关系进行排序
3. 选取距离最小的K个点
4. 确定最小k个点所在类别的出现频率
5. 返回前K个点中出现频率最高的类别，作为测试数据的预测分类

五、k的选取

k 小：分类结果容易因为噪声点的干扰而出现错误，此时方差较大

k 大：偏差过大

六、KNN算法举例

- 下方数据集中序号1-12为已知电影分类，分为喜剧片、动作片、爱情片三个种类，使用的特征值分别为搞笑镜头、打斗镜头、拥抱镜头数量。那么来了一部新电影（唐人街探案2），它属于上述电影分类中的那个类型？利用KNN是怎么做？

序号	电影名称	搞笑镜头	拥抱镜头	打斗镜头	电影类型
1.	宝贝当家	45	2	9	喜剧片
2.	美人鱼	21	17	5	喜剧片
3.	澳门风云3	54	9	11	喜剧片
4.	功夫熊猫3	39	0	31	喜剧片
5.	谍影重重	5	2	57	动作片
6.	叶问3	3	2	65	动作片
7.	伦敦陷落	2	3	55	动作片
8.	我的特工爷爷	6	4	21	动作片
9.	奔爱	7	46	4	爱情片
10.	夜孔雀	9	39	8	爱情片
11.	代理情人	9	38	2	爱情片
12.	新步步惊心	8	34	17	爱情片
13.	唐人街探案	23	3	17	?

1-12为**训练集**；13为**测试集**；喜剧片、动作片、爱情片为**标签**；搞笑镜头、打斗镜头、拥抱镜头数量为**特征值**

- 代码：

```
#!/usr/bin/env python
```

```

# -*- coding: utf-8 -*-
# @Time : 2020/5/30 22:33
# @Author : Kai
# @File : KnnMain.py
# @Software: PyCharm

from pandas import *
import math

# 1. 使用字典dict构造数据集
movie_data = {
    "宝贝当家": [45, 2, 9, "喜剧片"],
    "美人鱼": [21, 17, 5, "喜剧片"],
    "澳门风云3": [54, 9, 11, "喜剧片"],
    "功夫熊猫3": [39, 0, 31, "喜剧片"],
    "谍影重重": [5, 2, 57, "动作片"],
    "叶问3": [3, 2, 57, "动作片"],
    "伦敦陷落": [2, 3, 55, "动作片"],
    "我的特工爷爷": [6, 4, 21, "动作片"],
    "奔爱": [7, 46, 4, "爱情片"],
    "夜孔雀": [9, 39, 8, "爱情片"],
    "代理情人": [9, 38, 2, "爱情片"],
    "新步步惊心": [8, 34, 17, "爱情片"]
}

# 2. 计算新样本与数据集中所有数据的距离
# 新样本: 唐人街探案[23,3,17,"?片"]
# 欧式距离
x = [23, 3, 17]
KNN = []
for key, v in movie_data.items():
    d = math.sqrt((x[0] - v[0]) ** 2 + (x[1] - v[1]) ** 2 + (x[2] - v[2]) ** 2)
    KNN.append([key, round(d, 2)])
print(KNN)

# 3. 按照距离大小进行递增排序
KNN.sort(key = lambda dis: dis[1])

# 4. 选取距离最小的k个样本, 这里k取5
KNN = KNN[:5]
print(KNN)

# 5. 确定前k个样本所在类别出现的频率, 并输出出现频率最高的类别
labels = {"喜剧片": 0, "动作片": 0, "爱情片": 0}
for s in KNN:
    label = movie_data[s[0]] # 按照电影名取出movie_data中的整个电影信息
    labels[label[3]] += 1
labels = sorted(labels.items(), key = lambda l: l[1], reverse = True)
print(labels, labels[0][0], sep='\n')

```

- 运行结果:

```
[['宝贝当家', 23.43], ['美人鱼', 18.55], ['澳门风云3', 32.14], ['功夫熊猫3', 21.47],
['谍影重重', 43.87], ['叶问3', 44.73], ['伦敦陷落', 43.42], ['我的特工爷爷', 17.49],
['奔爱', 47.69], ['夜孔雀', 39.66], ['代理情人', 40.57], ['新步步惊心', 34.44]]
[['我的特工爷爷', 17.49], ['美人鱼', 18.55], ['功夫熊猫3', 21.47], ['宝贝当家', 23.43],
['澳门风云3', 32.14]]
[('喜剧片', 4), ('动作片', 1), ('爱情片', 0)]
喜剧片
```

- 运行截图：

```
D:\360MoveData\Users\38004\Desktop\MyGitFile\Python学习\MachineLearningFile\KNN_Example\venv\Scripts\python.exe D:/360Mc
[['宝贝当家', 23.43], ['美人鱼', 18.55], ['澳门风云3', 32.14], ['功夫熊猫3', 21.47], ['谍影重重', 43.87], ['叶问3', 44.73],
[['我的特工爷爷', 17.49], ['美人鱼', 18.55], ['功夫熊猫3', 21.47], ['宝贝当家', 23.43], ['澳门风云3', 32.14]]
[('喜剧片', 4), ('动作片', 1), ('爱情片', 0)]
喜剧片
```

七、KNN经典实例 —— 约会网站

1. 背景

海伦女士一直使用在线约会网站寻找适合自己的约会对象。尽管约会网站会推荐不同的人选，但她并不是喜欢每一个人。经过一番总结，她发现自己交往过的人可以进行如下分类：

- 不喜欢的人
- 魅力一般的人
- 极具魅力的人

海伦收集的样本数据主要包含以下3种特征：

- 每年获得的飞行常客里程数
- 玩视频游戏所消耗时间百分比
- 每周消费的冰淇淋公升数

现在就通过这3中特征来对样本进行分类

2.数据预处理：规范化（归一化）

- 方法：

1. 最大-最小规范化

假设 min_A 和 max_A 分别为属性 A 的最大值和最小值。最大 — 最小规范化通过计算：

$$v'_i = \frac{v_i - min_A}{max_A - min_A} (newmax_A - newmin_A) + newmin_A$$

把 A 的值 v_i 映射到区间 $newmin_A, newmax_A$ 中的 v'_i

2. z分数(z - score)规范化(或零均值规范化)

属性 A 的值基于 A 的均值（即平均值）和标准差规范化。 A 的值 v_i 被规范化为 v'_i ，由下式计算：

$$v'_i = \frac{v_i - \bar{A}}{\sigma_A}$$

其中 \bar{A} 和 σ_A 分别为 A 的均值和方差

- 代码


```

#-*- coding: utf-8 -*-
#数据规范化
import pandas as pd
import numpy as np

datafile = '../data/normalization_data.xls' #参数初始化
data = pd.read_excel(datafile, header = None) #读取数据

(data - data.min())/(data.max() - data.min()) #最小-最大规范化
(data - data.mean())/data.std() #零-均值规范化

```

3. “约会网站”代码

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Time : 2020/5/31 11:25
# @Author : Kai
# @File : KnnAppointmentWebsite.py
# @Software: PyCharm

from numpy import *
import operator

# 1. 读取数据 处理数据 返回特征值列表和结果列表
# returnMat 返回特征值列表
# classLabelVector 返回的结果(标签)列表
def file2matrix(filename):
    fr = open(filename)
    content = fr.readlines()
    numberOfLines = len(content)
    returnMat = zeros((numberOfLines, 3))
    classLabelVector = []
    index = 0
    for line in content:
        line = line.strip()
        listFromLine = line.split('\t')
        returnMat[index, :] = listFromLine[0:3]
        classLabelVector.append(int(listFromLine[-1]))
        index += 1
    return returnMat, classLabelVector

# 2. 归一化数值 使用了最大-最小规范化
def autoNorm(dataSet):
    minVals = dataSet.min(0) #返回dataSet矩阵中所有列中元素的最小值[0.,0.,0.001156]
    maxVals = dataSet.max(0)
    ranges = maxVals - minVals
    m = dataSet.shape[0] # 读取矩阵的长度
    normDataSet = dataSet - tile(minVals, (m, 1))
    normDataSet = normDataSet / tile(ranges, (m, 1))
    return normDataSet, ranges, minVals

# 3. KNN分类器
# now 测试数据
# dataSet 训练集

```

```

# labels 训练集的标签 (结果)
# k KNN参数k
def KNN_classify(now, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
    n = tile(now, (dataSetSize, 1))
    DiffMat = n - dataSet
    sqDiffMat = DiffMat ** 2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances ** 0.5 # 到这里求解了欧式距离(并构成了一个ndarray)

    sortedDistances = distances.argsort() # 根据排名作为索引 Index
    classCount = {}
    for i in range(k):
        voteIlabel = labels[sortedDistances[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1
    # 选出了距离最小的k个点, 并且做了一个简单的统计

    sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1),
reverse=True) # 按照第一个(从0开始数)进行排序
    return sortedClassCount[0][0] # 返回的出现次数最多的那个标签

# 4. 验证分类器, 使用10%的数据作为测试集 后90%作为训练集
def datingClassTest():
    # 测试集的比例10%
    hoRatio = 0.1
    datingDataMat, datingLabels = file2matrix("datingTestSet2.txt")
    normMat, ranges, minVals = autoNorm(datingDataMat)
    m = normMat.shape[0]
    numTestVecs = int(m * hoRatio)
    errorCount = 0.0
    for i in range(numTestVecs):
        now = normMat[i, :]
        dataSet = normMat[numTestVecs:m, :]
        labels = datingLabels[numTestVecs:m]
        k = 3
        classifierResult = KNN_classify(now, dataSet, labels, k)
        print("分类器预测结果:%d, 正确结果是:%d" % (classifierResult, datingLabels[i]))
        if (classifierResult != datingLabels[i]): errorCount += 1.0
    print("错误率%f" % (errorCount / float(numTestVecs)))

# 5. 给出一个人的三种特征信息并进行预测
def classifyPerson(x1, x2, x3):
    resultList = ['不喜欢的人', '魅力一般的人', '极具魅力的人']
    datingDataMat, datingLabels = file2matrix("datingTestSet2.txt")
    normMat, ranges, minVals = autoNorm(datingDataMat)
    now = array([x1, x2, x3])
    # 将预测值归一化
    now = (now - minVals) / ranges
    k = 3
    classifierResult = KNN_classify(now, normMat, datingLabels, k)
    print("结果:", resultList[classifierResult - 1])

# 测试
# print("-----验证分类器-----")
# datingClassTest()

# 预测
print("-----按照特征预测-----")

```

```

classifyPerson(33338, 10, 0.5)

# 可视化
# datingDataMat, datingLabels = file2matrix("datingTestSet2.txt")
# fig = plt.figure()
# plt.rcParams['font.family']=['STFangsong']
# plt.figure(figsize=(8, 5), dpi=80)
# ax = plt.subplot(111)
#
# datingLabels = array(datingLabels)
# idx_1 = where(datingLabels==1)
# p1 = ax.scatter(datingDataMat[idx_1,0],datingDataMat[idx_1,1],marker = '*',color =
# 'r',label='1',s=10)
# idx_2 = where(datingLabels==2)
# p2 = ax.scatter(datingDataMat[idx_2,0],datingDataMat[idx_2,1],marker = 'o',color
# ='g',label='2',s=20)
# idx_3 = where(datingLabels==3)
# p3 = ax.scatter(datingDataMat[idx_3,0],datingDataMat[idx_3,1],marker = '+',color
# ='b',label='3',s=30)
#
# plt.xlabel(u'每年获取的飞行里程数')
# plt.ylabel(u'玩视频游戏所消耗的事件百分比')
# ax.legend((p1, p2, p3), ('不喜欢', '魅力一般', '极具魅力'), loc=2)
# plt.show()

```

4.分类结果可视化

981	18991	0.454750	1.033280	2
982	9193	0.510310	0.016395	2
983	2285	3.864171	0.616349	2
984	9493	6.724021	0.563044	2
985	2371	4.289375	0.012563	2
986	13963	0.000000	1.437030	2
987	2299	3.733617	0.698269	2
988	5262	2.002589	1.380184	2
989	4659	2.502627	0.184223	2
990	17582	6.382129	0.876581	2
991	27750	8.546741	0.128706	3
992	9868	2.694977	0.432818	2
993	18333	3.951256	0.333300	2
994	3780	9.856183	0.329181	2
995	18190	2.068962	0.429927	2
996	11145	3.410627	0.631838	2
997	68846	9.974715	0.669787	1
998	26575	10.650102	0.866627	3
999	48111	9.134528	0.728045	3
1000	43757	7.882601	1.332446	3

- 根据数据中的分类进行可视化 (数据中第四列为标签)

选择属性 每年获取的飞行里程数 和 玩视频游戏所消耗的事件百分比 进行可视化

代码:

```

# 可视化
datingDataMat, datingLabels = file2matrix("datingTestSet2.txt")

```

```

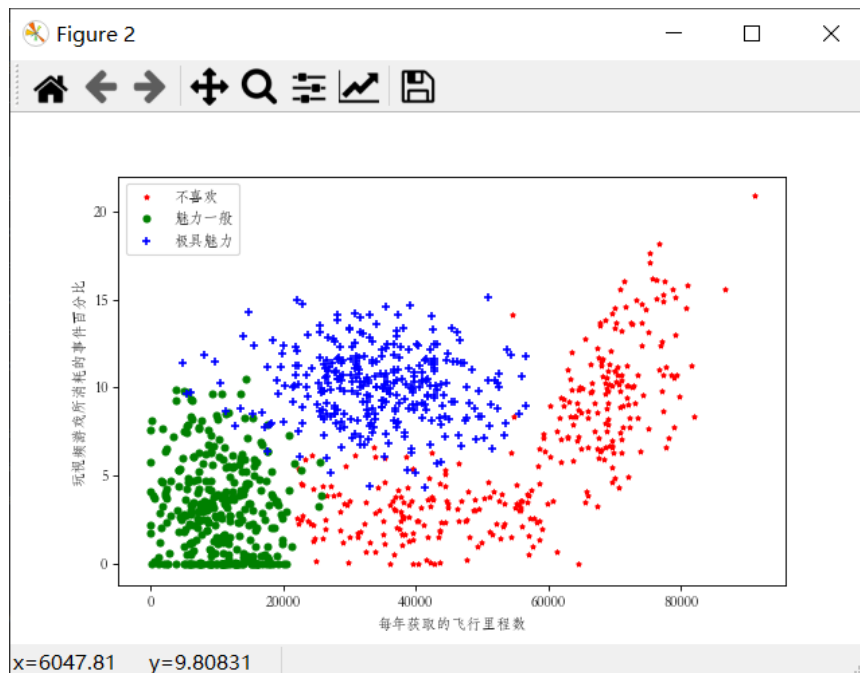
fig = plt.figure()
plt.rcParams['font.family']=['STFangsong']
plt.figure(figsize=(8, 5), dpi=80)
ax = plt.subplot(111)

datingLabels = array(datingLabels)
idx_1 = where(datingLabels==1)
p1 = ax.scatter(datingDataMat[idx_1,0],datingDataMat[idx_1,1],marker = '*',color =
'r',label='1',s=10)
idx_2 = where(datingLabels==2)
p2 = ax.scatter(datingDataMat[idx_2,0],datingDataMat[idx_2,1],marker = 'o',color
='g',label='2',s=20)
idx_3 = where(datingLabels==3)
p3 = ax.scatter(datingDataMat[idx_3,0],datingDataMat[idx_3,1],marker = '+',color
='b',label='3',s=30)

plt.xlabel(u'每年获取的飞行里程数')
plt.ylabel(u'玩视频游戏所消耗的事件百分比')
ax.legend((p1, p2, p3), ('不喜欢', '魅力一般', '极具魅力'), loc=2)
plt.show()

```

- 运行截图



5.“预测”运行截图

```

KnnAppointmentWebsite x
D:\Anaconda\python.exe D:/360MoveData/Users/38004/Desktop/MyGitFile/Python学习/Machin
-----按照特征预测-----
结果：极具魅力的人

```

6.“测试”运行截图

使用了“留出法(hold-out)”将数据集D划分为两个集合

前10%为测试集

后90%为训练集

```
# 测试
print("-----验证分类器-----")
datingClassTest()
```



```
D:\Anaconda\python.exe D:/360MoveData/Users/38004/Desktop/
-----验证分类器-----
分类器预测结果:3, 正确结果是:3
分类器预测结果:2, 正确结果是:2
分类器预测结果:1, 正确结果是:1
分类器预测结果:1, 正确结果是:1
分类器预测结果:1, 正确结果是:1
分类器预测结果:1, 正确结果是:1
分类器预测结果:3, 正确结果是:3
分类器预测结果:3, 正确结果是:3
分类器预测结果:1, 正确结果是:1
分类器预测结果:3, 正确结果是:3
分类器预测结果:3, 正确结果是:3
分类器预测结果:2, 正确结果是:2
分类器预测结果:1, 正确结果是:1
分类器预测结果:3, 正确结果是:1
错误率0.050000
```

7.以上 [测试运行截图](#) 中的k值为3 按照测试集10% 训练集90%的比例进行测试错误率

现选择不同的k测试错误率

k值	错误率
3	0.05
4	0.04
5	0.05
10	0.06
15	0.06
20	0.06
50	0.07
100	0.07

8.以上测试均为规范化后的测试，现去除规范化的步骤，再次进行测试

- 修改验证分类器代码：

```
# 4. 验证分类器，使用10%的数据作为测试集 后90%作为训练集
def datingClassTest():
    # 测试集的比例10%
    hoRatio = 0.1
    datingDataMat, datingLabels = file2matrix("datingTestSet2.txt")
    m = datingDataMat.shape[0]
    numTestVecs = int(m * hoRatio)
    errorCount = 0.0
    for i in range(numTestVecs):
        now = datingDataMat[i, :]
        dataSet = datingDataMat[numTestVecs:m, :]
        labels = datingLabels[numTestVecs:m]
        k = 3
        classifierResult = KNN_classify(now, dataSet, labels, k)
        print("分类器预测结果:%d, 正确结果是:%d" % (classifierResult, datingLabels[i]))
        if (classifierResult != datingLabels[i]): errorCount += 1.0
    print("错误率%f" % (errorCount / float(numTestVecs)))
```

- 运行截图：

```
分类器预测结果:2, 正确结果是:2
分类器预测结果:1, 正确结果是:1
分类器预测结果:1, 正确结果是:3
分类器预测结果:3, 正确结果是:3
分类器预测结果:2, 正确结果是:2
分类器预测结果:3, 正确结果是:1
分类器预测结果:3, 正确结果是:1
错误率0.240000
```

- 现选择不同的k对未规范化的KNN分类器测试错误率

k值	错误率
3	0.24
4	0.20
5	0.27
10	0.24
15	0.23
20	0.22
50	0.24
100	0.24

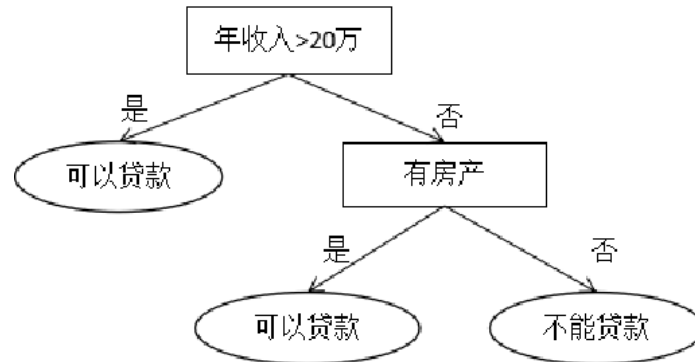
可以明显看出，未进行数据规范化的分类器的错误率明显大于数据规范化后的错误率。

决策树

决策树是一种有监督的分类方法，它是用已有的数据构造出一棵树，用这棵树，对新的数据进行预测

树形决策过程

举例



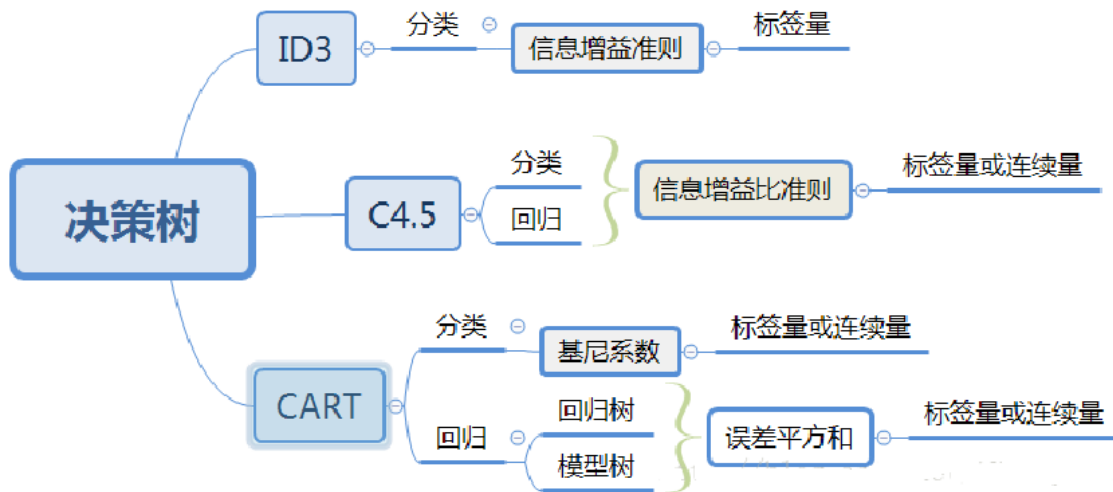
1. 决策之前，先获取客户的两个指标-**年收入**，**是否有房产**，形成**特征向量**
2. 从树的根节点开始，在内部节点处需要做判断，直到到达一个**叶子节点**处，得到**决策结果**

- 特征：有 **数值型特征** 和 **类别型特征**
- 节点：
 - 内部节点-对应判定规则
 - 叶子节点-对应预测结果

决策树构造过程

1. 特征选择：特征选择是指从训练数据中众多的特征中选择一个特征作为当前节点的**分裂标准**
2. 决策树生成：根据选择的特征评估标准，从上至下递归地生成子节点，**直到数据集不可分则决策树停止生长。**
3. 剪枝：决策树容易**过拟合**，一般来需要剪枝，缩小树结构规模、缓解过拟合。剪枝技术有**预剪枝**和**后剪枝**两种。

根据**特征选择的准则**可将决策树分类：



1. 树节点属性的选取

• 信息增益

信息熵：是度量样本集合“纯度”最常用的一种指标

假设当前样本集合 D 中第 k 类样本所占的比例为 p_i ，则 D 的信息熵为：

$$Ent(D) = - \sum_{k=1}^n p_i \ln p_i = - \sum_{k=1}^n \frac{|C_k|}{|D|} \log_2 \frac{|C_k|}{|D|}$$

n 表示样本标签值的取值状态数

C_k 表示 D 中所标注值为 k 的训练样本组成的集合

$|D|$ 和 $|C_k|$ 分别表示集合 D 和 C_k 的基数

$H(D)$ 值越大， D 中包含样本标签的取值越杂乱，越小则越纯净

信息增益：(ID3)

离散属性 a 的取值： $\{a^1, a^2, \dots, a^V\}$ ， D^v 是 D 中在 a 上取值 a^v 的样本集合

以属性 a 对数据集 D 进行划分所获得的信息增益为：

$$Gain(D, a) = Ent(D) - \sum_{v=1}^V \frac{|D^v|}{|D|} Ent(D^v)$$

$Ent(D)$ 为划分前的信息熵

$\sum_{v=1}^V \frac{|D^v|}{|D|} Ent(D^v)$ 为划分后的信息熵

2. 决策树构造过程

- 策略：分而治之；自根至叶的过程；在每个中间节点找一个“划分”的属性
- 三种停止条件：

当前结点包含的样本全属于同一类别，无需划分；

当前属性集为空，或是所有样本在所有属性上取值相同，无法划分；

当前结点包含的样本集合为空，不能划分。

3. 不同的树节点选择标准

- **增益率** (C4.5)

-]基尼指数 (CART)

4. 树模型简化

剪枝可去掉一些分支来降低过拟合的风险

- 基本策略

剪枝策略	预剪枝	后剪枝
方式	提前终止某些分支的生长	生成一棵完全树，再“回头”剪枝
时间开销	训练时间开销降低，测试时间开销降低	训练时间开销增加，测试时间开销降低
过/欠拟合风险	过拟合风险降低，欠拟合风险增加	过拟合风险降低，欠拟合风险增加

使用决策树完成对iris数据的分类

1. iris数据集描述

- 特征数据:

```
[ 5.1  3.5  1.4  0.2]
[ 4.9  3.   1.4  0.2]
[ 4.7  3.2  1.3  0.2]
[ 4.6  3.1  1.5  0.2]
[ 5.   3.6  1.4  0.2]
[ 5.4  3.9  1.7  0.4]
[ 4.6  3.4  1.4  0.3]
[ 5.   3.4  1.5  0.2]
[ 4.4  2.9  1.4  0.2]
[ 4.9  3.1  1.5  0.1]
...
[ 5.9  3.   5.1  1.8]
```

4个特征名称:

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

- 类数据:

```
[000000000000000000000000000000000000000000000  
00000000000000001111111111111111111111111111  
11111111111111111111111111111111222222222222  
2222222222222222222222222222222222222222222  
22]
```

3个类的名称:

```
['setosa' 'versicolor' 'virginica']
```

- 可视化数据集

代码:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Time : 2020/6/2 11:13
# @Author : Kai
# @File : 可视化数据集.py
# @Software: PyCharm

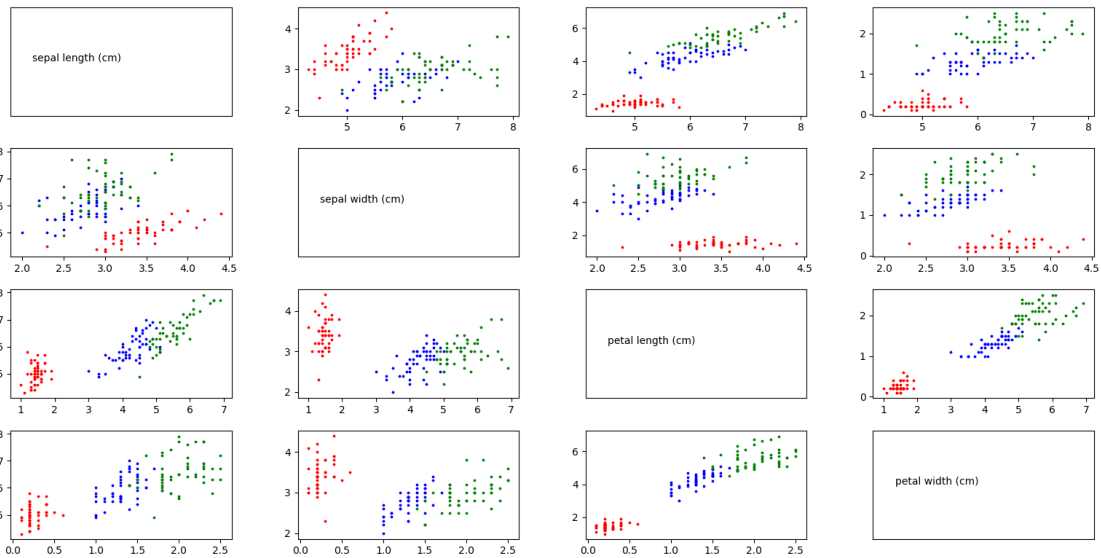
from matplotlib import pyplot as plt
from sklearn.datasets import load_iris

if __name__ == '__main__':
    # show data info
    data = load_iris() # 加载 IRIS 数据集
    feature_names = data.get('feature_names')
    x = data.get('data') # 获取样本矩阵
    y = data.get('target') # 获取与样本对应的 label 向量

    f = []
    f.append(y == 0) # 类别为第一类的样本的逻辑索引 若为第一类则为true,否则false
    f.append(y == 1) # 类别为第二类的样本的逻辑索引
    f.append(y == 2) # 类别为第三类的样本的逻辑索引

    color = ['red', 'blue', 'green']
    fig, axes = plt.subplots(4, 4) # 绘制四个属性两两之间的散点图
    for i, ax in enumerate(axes.flat):
        row = i // 4
        col = i % 4
        if row == col:
            ax.text(.1, .5, feature_names[row])
            ax.set_xticks([])
            ax.set_yticks([])
            continue
        for k in range(3):
            ax.scatter(x[f[k], row], x[f[k], col], c=color[k], s=3)
    fig.subplots_adjust(hspace=0.3, wspace=0.3) # 设置间距
    plt.show()
```

运行截图:



2. 使用ID3的特征选择策略

- 整体代码:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Time : 2020/6/2 9:39
# @Author : Kai
# @File : ID3.py
# @Software: PyCharm

# 数据集
from sklearn import datasets
# 分类器
from sklearn import tree
# 训练集测试集分割模块
from sklearn.model_selection import train_test_split
# 绘制决策树
import graphviz
from sklearn import metrics

# 自定义导入数据集函数
def get_data(total_data):
    # 显示total_data包含的内容
    # print("传入数据集包含内容有: ", [x for x in total_data.keys()])

    # 样本
    x_true = total_data.data
    # 标签
    y_true = total_data.target
    # 特征名称 -> ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)']
    feature_names = total_data.feature_names
    # 类名 -> ['setosa' 'versicolor' 'virginica']
    target_names = total_data.target_names
    return x_true, y_true, feature_names, target_names

# 定义主函数
def main():
```

```

# 利用自定义函数导入Iris数据集
total_iris = datasets.load_iris()
x_true, y_true, feature_names, target_names = get_data(total_iris)

# 分割数据集 “留出法”测试
rate_test = 0.2 # 训练集比例
x_train, x_test, y_train, y_test =
train_test_split(x_true,y_true,test_size=rate_test)
print("训练集样本大小: ", x_train.shape)
print("训练集标签大小: ", y_train.shape)
print("测试集样本大小: ", x_test.shape)
print("测试集标签大小: ", y_test.shape)

# 设置决策树分类器 ID3
clf = tree.DecisionTreeClassifier(criterion="entropy")
# 训练模型
clf.fit(x_train, y_train)

#1. 预测 输出属于哪一类
print("\n给定提供的属性值根据训练模型预测结果: ")
category = clf.predict([[5, 1.5, 2, 3]])
category_name = target_names[category]
print("类别%d, 类名%s" % (category, category_name))

#2. 评价模型：准确率、召回率、f1-score、精度
score = clf.score(x_test, y_test)
print("\n模型测试集准确率为: ", score)

y_predict = clf.predict(x_test)
result = metrics.classification_report(y_test, y_predict,
target_names=target_names)
print(result)

#3. 绘制决策树模型
clf_dot = tree.export_graphviz(clf,
                                out_file=None,
                                feature_names=feature_names,
                                class_names=target_names,
                                filled=True,
                                rounded=True)

# 显示绘制的模型，在当前目录下，保存为png模式
graph = graphviz.Source(clf_dot,
                        filename="iris_decisionTree.gv",
                        format="png")

graph.view()

# 显示特征重要程度
print("\n特征重要程度为: ")
info = [*zip(feature_names, clf.feature_importances_)]
for cell in info:
    print(cell)

# 调用主函数
if __name__ == "__main__":
    main()

```

• 运行截图：

因使用“**留出法**”对数据进行测试时训练集和测试集的数据是随机分的，所以运行数据每次不同

```
可视化数据集 × ID3 ×
D:\Anaconda\python.exe D:/360MoveData/Users/38004/Desktop/MyGitFile/Python学习/MachineLearningFile/决策树/Demo/DesisionTree/ID3.py
训练集样本大小: (120, 4)
训练集标签大小: (120,)
测试集样本大小: (30, 4)
测试集标签大小: (30,)

给定提供的属性值根据训练模型预测结果:
类别2, 类名['virginica']

模型测试集准确率为: 0.9666666666666667
precision    recall  f1-score   support

   setosa      1.00      1.00      1.00        10
  versicolor  1.00      0.92      0.96        13
   virginica   0.88      1.00      0.93         7

 accuracy      0.97        30
 macro avg      0.96        30
weighted avg      0.97        30

特征重要程度为:
('sepal length (cm)', 0.0)
('sepal width (cm)', 0.0)
('petal length (cm)', 0.0672596787139177)
('petal width (cm)', 0.9327403212860823)

Process finished with exit code 0
```

```
D:\Anaconda\python.exe D:/360MoveData/Users/38004/Desktop/MyGitFile/Python学习/MachineLearningFile/决策树/Demo/DesisionTree/ID3.py
训练集样本大小: (120, 4)
训练集标签大小: (120,)
测试集样本大小: (30, 4)
测试集标签大小: (30,)

给定提供的属性值根据训练模型预测结果:
类别2, 类名['virginica']

模型测试集准确率为: 0.9
precision    recall  f1-score   support

   setosa      1.00      1.00      1.00        10
  versicolor  0.80      1.00      0.89        12
   virginica   1.00      0.62      0.77         8

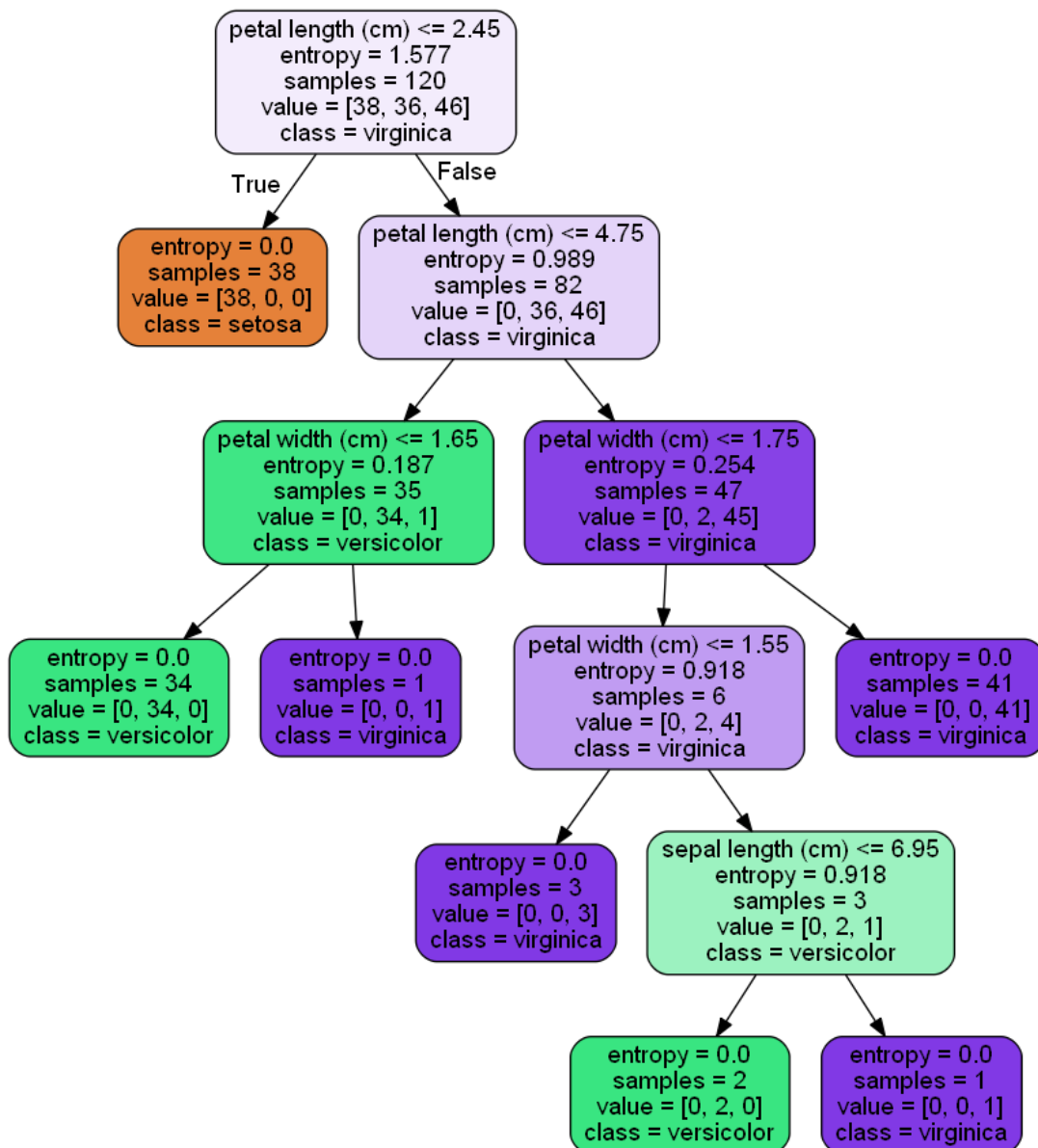
 accuracy      0.90        30
 macro avg      0.93        30
weighted avg      0.92        30

特征重要程度为:
('sepal length (cm)', 0.02104696325400949)
('sepal width (cm)', 0.03051951244114077)
('petal length (cm)', 0.025164319606822114)
('petal width (cm)', 0.9232692046980276)

Process finished with exit code 0
```

• ID3决策树可视化：

数据集中有120条数据用于构造此决策树



- 关键代码:

#1. 预测 输出属于哪一类

```
print("\n给定提供的属性值根据训练模型预测结果: ")
category = clf.predict([[5, 1.5, 2, 3]])
category_name = target_names[category]
print("类别%d, 类名%s" % (category, category_name))
```

-----输出结果-----

给定提供的属性值根据训练模型预测结果:
类别0, 类名['setosa']

#2. 评价模型: 准确率、召回率、f1-score、精度

```
score = clf.score(x_test, y_test)
print("\n模型测试集准确率为: ", score)

y_predict = clf.predict(x_test)
result = metrics.classification_report(y_test, y_predict, target_names=target_names)
print(result)
```

-----输出结果-----

模型测试集准确率为: 0.8666666666666667

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	38
versicolor	0.97	0.93	0.95	34
virginica	0.86	0.86	0.86	41
accuracy	0.87	0.87	0.87	113

setosa	1.00	1.00	1.00	12
versicolor	1.00	0.71	0.83	14
virginica	0.50	1.00	0.67	4
accuracy			0.87	30
macro avg	0.83	0.90	0.83	30
weighted avg	0.93	0.87	0.88	30

#3. 绘制决策树模型

```
clf_dot = tree.export_graphviz(clf,
                               out_file=None,
                               feature_names=feature_names,
                               class_names=target_names,
                               filled=True,
                               rounded=True)

# 显示绘制的模型，在当前目录下，保存为png模式
graph = graphviz.Source(clf_dot,
                        filename="iris_decisionTree.gv",
                        format="png")

graph.view()

# 输出：决策树可视化png图片
```

3. 使用CART的特征选择策略

- 修改 `DecisionTreeClassifier` 的参数

使用ID3的特征选择策略为：`clf = tree.DecisionTreeClassifier(criterion="entropy")`

使用CART的特征选择策略改为：`clf = tree.DecisionTreeClassifier(criterion='gini')`

- 改动后的代码截图

```
44 print("测试集样本大小: ", x_test.shape)
45 print("测试集标签大小: ", y_test.shape)
46
47 # 设置决策树分类器 ID3
48 clf = tree.DecisionTreeClassifier(criterion='gini')
49 # 训练模型
50 clf.fit(x_train, y_train)
```

- 运行截图

```

D:\Anaconda\python.exe D:/360MoveData/Users/38004/Desktop/My
训练集样本大小: (120, 4)
训练集标签大小: (120,)
测试集样本大小: (30, 4)
测试集标签大小: (30,)

给定提供的属性值根据训练模型预测结果:
类别1, 类名['versicolor']

模型测试集准确率为: 0.9666666666666667

```

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	11
versicolor	0.86	1.00	0.92	6
virginica	1.00	0.92	0.96	13
accuracy			0.97	30
macro avg	0.95	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30

```

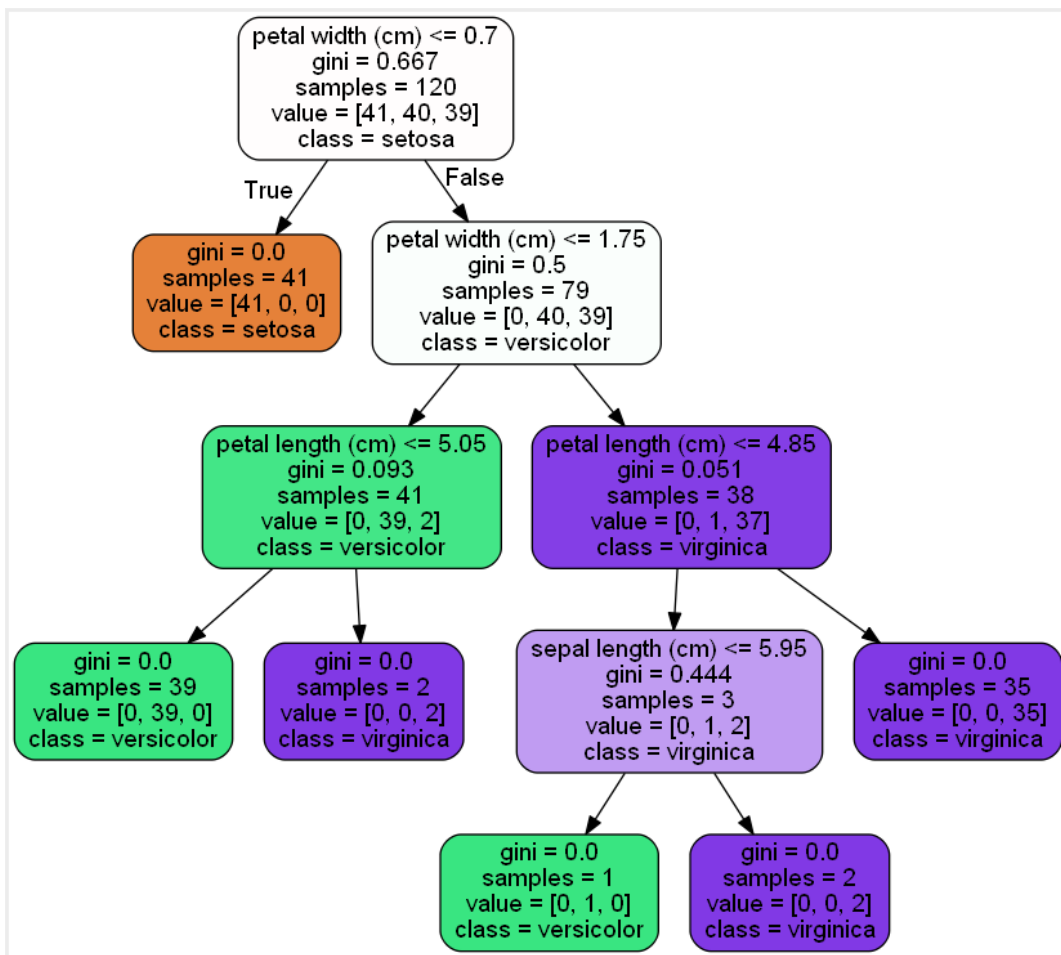
特征重要程度为:
('sepal length (cm)', 0.016711928138709)
('sepal width (cm)', 0.0)
('petal length (cm)', 0.044803983134040035)
('petal width (cm)', 0.938484088727251)

```

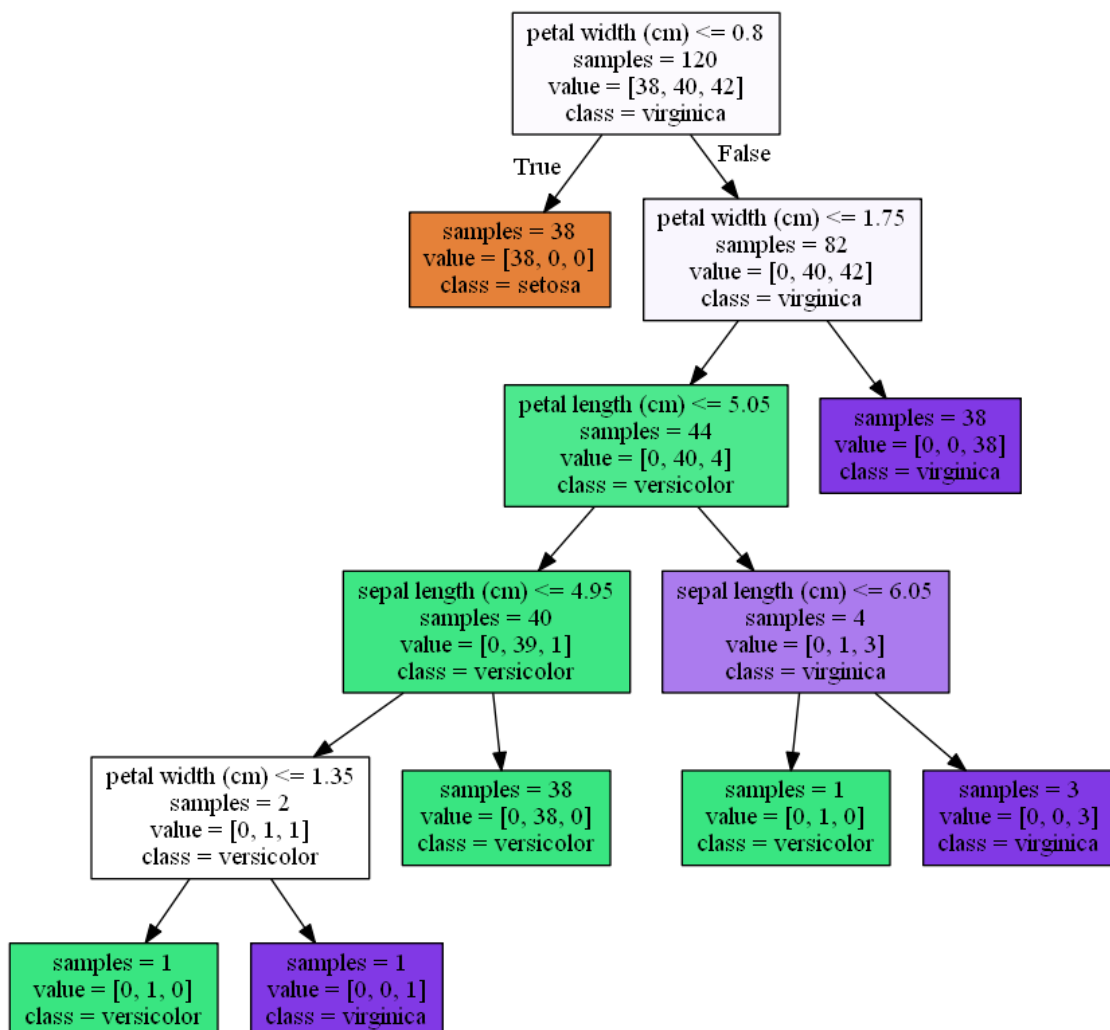
- 生成的决策树可视化

因使用“**留出法**”对数据进行测试时训练集和测试集的数据是随机分的，所以生成的决策树会有不同

- 决策树1



o 决策树2



随机森林

- BootStrap采样
- 集成学习
 - Bagging
 - Boosting (Bagging改进) 更专注被分类错误的样本

SVM支持向量机

2012年以前SVM占主流、占统治地位，2012年之后Deep Learning主流，效果好于SVM

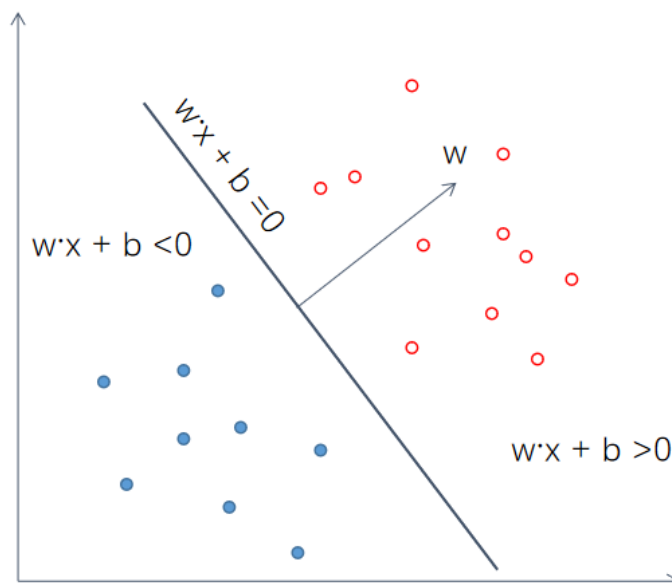
Machine Learning 最难的一块 —— SVM

一、线性可分支持向量机

(1). 基本概念

- **输入**：假设特征空间上的训练数据集
- 正例和负例
- 学习的目标：找到分类超平面和决策函数
- **输出1**：线性可分支持向量机：给定线性可分训练数据集，通过间隔最大化或等价地求解相应的凸二次规划问题学习得到的分离超平面为
- **输出2**：决策函数：
- **求解**：构造并求解约束最优化问题

硬间隔最大化：分类直线离支持向量的距离是最远的



(2). 点到超平面的距离

(3). 函数间隔和几何间隔

- 函数间隔
 - 样本点的函数间隔
 - 训练数据集的函数间隔
- 几何间隔
 - 样本点的几何距离
 - 训练数据集的几何间隔
- 函数间隔和几何间隔的关系

(4). 支持向量和Margins(边界)

支持向量：在线性可分情况下，训练数据集的样本点中与分离超平面距离最近的样本点的实例称为支持向量(support vector)

(5). 线性可分支持向量机学习算法

输入：线性可分训练数据集

输出：最大间隔分离超平面和分类决策函数

1. 构造并求解约束最优化问题
2. 计算
3. 求得分离超平面
分类决策函数

二、线性不可分支持向量机

线性支持向量机与软间隔最大化

- 训练数据中有一些特异点 (outlier) , 不能满足函数间隔大于等于1的约束条件。
- 解决方法：
 - 使得函数间隔加上松弛变量大于等于1, 约束条件变为：
目标函数变为：

三、非线性支持向量机与核函数

- 如果如果能用 R^n 中的一个**超曲面**将正负例正确分开, 则称这个问题为**非线性可分问题**。
- **非线性变换**, 将非线性问题变换为线性问题, 通过变换后的线性问题的方法求解原来的非线性问题。
- 用线性分类方法求解非线性分类问题分为两步:
首先使用一个变换将**原空间**的数据映射到**新空间**;
然后在新空间里用**线性分类学习方法**从训练数据中学习分类模型。

设 X 是输入空间(欧氏空间 R^n 的子集或离散集合), 又设 H 为特征空间(希尔伯特空间), 如果存在一个从 X 到 H 的映射

四、二分类支持向量机SVM实例

数据:

属性1	属性2	标签
-0.017612	14.053064	0
-1.395634	4.662541	1
-0.752157	6.538620	0
-1.322371	7.152853	0
0.423363	11.054677	0
.....		
0.761349	10.693862	0
-2.168791	0.143632	1
1.388610	9.341997	0
0.317029	14.739025	0

整体代码:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Time : 2020/6/8 18:23
# @Author : Kai
# @File : 二分类.py
# @Software: PyCharm

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import metrics

def my_svm():
    #1. 数据预处理
    data = pd.read_table(r'./testSet.txt', header=None, delim_whitespace=True)

    X = np.array(data.loc[:, [0, 1]]) # 特征值
    Y = np.array(data[2]) # 标签
    Y = np.where(Y == 1, 1, -1) # 标签分为正例和负例

    #2. 预测
    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)
    print("训练集样本大小: ", x_train.shape)
    print("训练集标签大小: ", y_train.shape)
    print("测试集样本大小: ", x_test.shape)
    print("测试集标签大小: ", y_test.shape)
    svm_model = SVC(kernel="linear", C = 1.0)
    svm_model.fit(X, Y)
```

```

score = svm_model.score(x_test, y_test)
print("\n模型测试集准确率为: ", score)

# 评价模型: 准确率、召回率、f1-score、精度
y_predict = svm_model.predict(x_test)
result = metrics.classification_report(y_test, y_predict)
print(result)

# 单个预测
# pre_result = svm_model.predict([[1.25, 0.26]])
# print(pre_result)

#3. 选择三种不同的核方法训练SVM并进行可视化
# linear:线性核函数
# poly:多项式核函数
# rbf: 高斯核函数
x_min = X[:, 0].min()
x_max = X[:, 0].max()
y_min = X[:, 1].min()
y_max = X[:, 1].max()
plt.figure(figsize=(15, 15))
for fig_num, kernel in enumerate(('linear', 'poly', 'rbf')):
    # 训练SVM
    svm_ = SVC(kernel=kernel)
    svm_.fit(X, Y)

    # 输出支持向量
    print("%s_SVM支持向量个数: %d" % (kernel, len(svm_.support_vectors_)))
    print(svm_.support_vectors_)

    # 可视化
    plt.subplot(222 + fig_num)
    plt.scatter(x = X[Y == 1, 0], y = X[Y == 1, 1],
                s = 30, marker = 'o', color = 'yellow', zorder = 10)
    plt.scatter(x = X[Y == -1, 0], y = X[Y == -1, 1],
                s = 30, marker = 'x', color = 'blue', zorder = 10)
    plt.scatter(x = [x[0] for x in svm_.support_vectors_], y = [x[1] for x in
svm_.support_vectors_], s = 80, facecolors='none', zorder = 10)
    plt.title(kernel)
    plt.xlabel('support vectors ' + str(len(svm_.support_vectors_)))
    plt.xticks([])
    plt.yticks([])
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    XX, YY = np.mgrid[x_min:x_max:200j, y_min:y_max:200j]
    Z = svm_.decision_function(np.c_[XX.ravel(), YY.ravel()])
    Z = Z.reshape(XX.shape)
    plt.pcolormesh(XX, YY, Z > 0, cmap=plt.cm.Paired)
    plt.contour(XX, YY, Z, colors=['black', 'k', 'white'], linestyles=['--', '-',
'--'], levels=[-.5, 0, .5])

# plot data
plt.subplot(221)
plt.title('data')
plt.scatter(x=X[Y == 1, 0], y=X[Y == 1, 1],
            s=30, marker='o', color='red', zorder=10)
plt.scatter(x=X[Y == -1, 0], y=X[Y == -1, 1],
            s=30, marker='x', color='blue', zorder=10)

```

```
plt.xticks([])
plt.yticks([])
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)

plt.show()

if __name__ == '__main__':
    my_svm()
```

“预测 准确率+ 评价模型：准确率、召回率、f1-score、精度”输出结果：

部分相关代码：

```
#2. 预测
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)
print("训练集样本大小: ", x_train.shape)
print("训练集标签大小: ", y_train.shape)
print("测试集样本大小: ", x_test.shape)
print("测试集标签大小: ", y_test.shape)
svm_model = SVC(kernel="linear", C = 1.0)
svm_model.fit(X, Y)
score = svm_model.score(x_test, y_test)
print("\n模型测试集准确率为: ", score)

# 评价模型：准确率、召回率、f1-score、精度
y_predict = svm_model.predict(x_test)
result = metrics.classification_report(y_test, y_predict)
print(result)
```

输出：

```
训练集样本大小:  (70, 2)
训练集标签大小:  (70,)
测试集样本大小:  (30, 2)
测试集标签大小:  (30,)

模型测试集准确率为:  0.9666666666666667
```

	precision	recall	f1-score	support
-1	0.92	1.00	0.96	11
1	1.00	0.95	0.97	19
accuracy			0.97	30
macro avg	0.96	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30

- 单个数据预测结果

```
# 单个预测
pre_result = svm_model.predict([[1.25, 0.26]])
print(pre_result)
```

-----输出-----

```
[1]  # 即“正例”
```

输出支持向量

```
# 输出支持向量
print("%s_SVM支持向量个数: %d" % (kernel, len(svm_.support_vectors_)))
print(svm_.support_vectors_)
```

- 核方法: linear

```
linear_SVM支持向量个数: 12
[[-0.752157  6.53862 ]
 [-1.322371  7.152853]
 [-0.397822  8.058397]
 [ 1.015399  7.571882]
 [-1.510047  6.061992]
 [ 1.38861   9.341997]
 [ 0.406704  7.067335]
 [-2.46015   6.866805]
 [-0.566606  5.749003]
 [ 0.556921  8.294984]
 [ 0.099671  6.835839]
 [ 1.785928  7.718645]]
```

- 核方法: poly

```
poly_SVM支持向量个数: 15
[[-0.752157  6.53862 ]
 [-1.322371  7.152853]
 [ 0.184992  8.721488]
 [-0.397822  8.058397]
 [ 1.015399  7.571882]
 [-1.510047  6.061992]
 [ 1.38861   9.341997]
 [ 0.406704  7.067335]
 [-2.46015   6.866805]
 [ 0.850433  6.920334]
 [-0.024205  6.151823]
 [ 0.556921  8.294984]
 [ 0.099671  6.835839]
 [ 1.785928  7.718645]
 [ 3.01015   8.401766]]
```

- 核方法: rbf

```
rbf_SVM支持向量个数: 26
[[-7.521570e-01  6.538620e+00]
 [-1.322371e+00  7.152853e+00]
 [ 1.217916e+00  9.597015e+00]
 [-7.339280e-01  9.098687e+00]
 [ 1.416614e+00  9.619232e+00]
 [ 4.705750e-01  9.332488e+00]
 [ 1.849920e-01  8.721488e+00]
 [-3.978220e-01  8.058397e+00]
 [-7.194000e-03  9.075792e+00]
 [ 1.015399e+00  7.571882e+00]
 [-1.510047e+00  6.061992e+00]
```

```
[ 1.388610e+00  9.341997e+00]
[ 4.067040e-01  7.067335e+00]
[-2.460150e+00  6.866805e+00]
[ 8.504330e-01  6.920334e+00]
[-5.666060e-01  5.749003e+00]
[-2.420500e-02  6.151823e+00]
[-3.642001e+00 -1.618087e+00]
[ 5.569210e-01  8.294984e+00]
[ 1.042222e+00  6.105155e+00]
[ 2.294560e-01  5.921938e+00]
[ 9.967100e-02  6.835839e+00]
[ 1.785928e+00  7.718645e+00]
[ 2.530777e+00  6.476801e+00]
[-1.076637e+00 -3.181888e+00]
[ 3.010150e+00  8.401766e+00]]
```

- 可视化

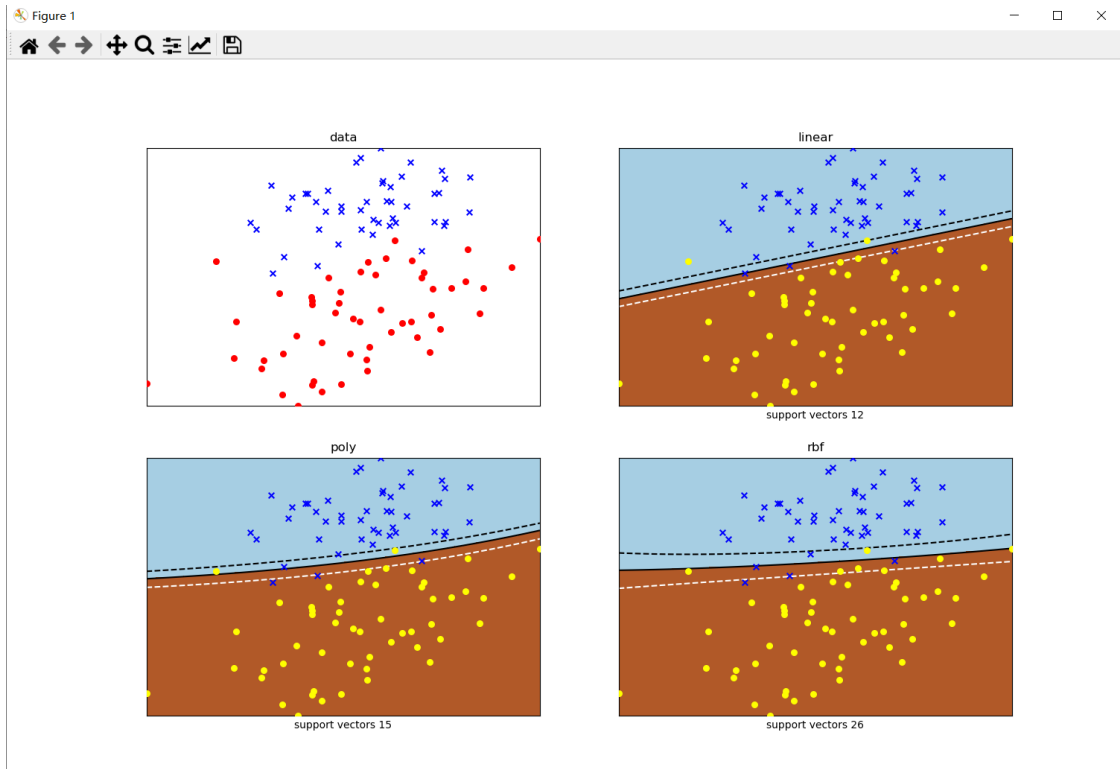
```
for fig_num, kernel in enumerate(('linear', 'poly', 'rbf')):
    # 训练SVM
    svm_ = SVC(kernel=kernel)
    svm_.fit(X, Y)

    # 可视化
    plt.subplot(222 + fig_num)
    plt.scatter(x = X[Y == 1, 0], y = X[Y == 1, 1],
                s = 30, marker = 'o', color = 'yellow', zorder = 10)
    plt.scatter(x = X[Y == -1, 0], y = X[Y == -1, 1],
                s = 30, marker = 'x', color = 'blue', zorder = 10)
    plt.scatter(x = [x[0] for x in svm_.support_vectors_], y = [x[1] for x in
svm_.support_vectors_], s = 80, facecolors='none', zorder = 10)
    plt.title(kernel)
    plt.xlabel('support vectors ' + str(len(svm_.support_vectors_)))
    plt.xticks([])
    plt.yticks([])
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    XX, YY = np.mgrid[x_min:x_max:200j, y_min:y_max:200j]
    Z = svm_.decision_function(np.c_[XX.ravel(), YY.ravel()])
    Z = Z.reshape(XX.shape)
    plt.pcolormesh(XX, YY, Z > 0, cmap=plt.cm.Paired)
    plt.contour(XX, YY, Z, colors=['black', 'k', 'white'], linestyles=['--', '-',
'--'], levels=[-.5, 0, .5])

    # plot data
    plt.subplot(221)
    plt.title('data')
    plt.scatter(x=X[Y == 1, 0], y=X[Y == 1, 1],
                s=30, marker='o', color='red', zorder=10)
    plt.scatter(x=X[Y == -1, 0], y=X[Y == -1, 1],
                s=30, marker='x', color='blue', zorder=10)
    plt.xticks([])
    plt.yticks([])
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)

    plt.show()
```


可视化图像：



设置不同软间隔输出预测结果

通过修改惩罚参数 C 值，修改软间隔 (使用线性模型)

```
svm_model = SVC(kernel="linear", C = 1.0)
```

- $C=0.1$

预测结果：

模型测试集准确率为： 0.9333333333333333

	precision	recall	f1-score	support
-1	0.94	0.94	0.94	18
1	0.92	0.92	0.92	12
accuracy			0.93	30
macro avg	0.93	0.93	0.93	30
weighted avg	0.93	0.93	0.93	30

支持向量个数：18

- $C=0.5$

预测结果：

模型测试集准确率为: 0.9666666666666667					
	precision	recall	f1-score	support	
-1	0.94	1.00	0.97	15	
1	1.00	0.93	0.97	15	
accuracy			0.97	30	
macro avg	0.97	0.97	0.97	30	
weighted avg	0.97	0.97	0.97	30	

支持向量个数: 13

- C=1

预测结果:

模型测试集准确率为: 0.9666666666666667					
	precision	recall	f1-score	support	
-1	0.93	1.00	0.96	13	
1	1.00	0.94	0.97	17	
accuracy			0.97	30	
macro avg	0.96	0.97	0.97	30	
weighted avg	0.97	0.97	0.97	30	

支持向量个数: 12

- C=2

预测结果:

模型测试集准确率为: 0.9333333333333333					
	precision	recall	f1-score	support	
-1	0.93	0.93	0.93	14	
1	0.94	0.94	0.94	16	
accuracy			0.93	30	
macro avg	0.93	0.93	0.93	30	
weighted avg	0.93	0.93	0.93	30	

支持向量个数: 11

- C= 5

预测结果:

模型测试集准确率为: 0.9333333333333333					
	precision	recall	f1-score	support	
-1	0.87	1.00	0.93	13	
1	1.00	0.88	0.94	17	
accuracy			0.93	30	
macro avg	0.93	0.94	0.93	30	
weighted avg	0.94	0.93	0.93	30	

支持向量个数: 11

五、多分类支持向量机SVM实例

数据

使用Iris数据集

```
# 1.读取数据集
d = load_iris()

# 2.划分数据与标签
x = d.data
y = d.target
```

代码

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Time : 2020/6/8 17:00
# @Author : Kai
# @File : 多分类.py
# @Software: PyCharm

from sklearn import svm
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# 1.读取数据集
d = load_iris()

# 2.划分数据与标签
x = d.data
y = d.target
x = x[:, 0:2]
train_data, test_data, train_label, test_label = train_test_split(x, y,
                                                                    random_state=0, train_size=0.7, test_size=0.3)

# 3.训练svm分类器
classifier = svm.SVC(C=2, kernel='rbf', gamma=10, decision_function_shape='ovo') #
ovr:一对多策略
classifier.fit(train_data, train_label.ravel()) # ravel函数在降维时默认是行序优先

# 4.计算svc分类器的准确率
print("训练集: ", classifier.score(train_data, train_label))
print("测试集: ", classifier.score(test_data, test_label))

# 查看决策函数
print('train_decision_function:\n', classifier.decision_function(train_data)) #
(90,3)
print('predict_result:\n', classifier.predict(train_data))

# 输出支持向量
print(classifier.support_vectors_)
```

```

# 5. 绘制图形
# 确定坐标轴范围
x1_min, x1_max = x[:, 0].min(), x[:, 0].max() # 第0维特征的范围
x2_min, x2_max = x[:, 1].min(), x[:, 1].max() # 第1维特征的范围
x1, x2 = np.mgrid[x1_min:x1_max:200j, x2_min:x2_max:200j] # 生成网络采样点
grid_test = np.stack((x1.flat, x2.flat), axis=1) # 测试点
# 指定默认字体
matplotlib.rcParams['font.sans-serif'] = ['SimHei']
# 设置颜色
cm_light = matplotlib.colors.ListedColormap(['#A0FFA0', '#FFA0A0', '#A0A0FF'])
cm_dark = matplotlib.colors.ListedColormap(['g', 'r', 'b'])

grid_hat = classifier.predict(grid_test) # 预测分类值
grid_hat = grid_hat.reshape(x1.shape) # 使之与输入的形状相同

plt.pcolormesh(x1, x2, grid_hat, cmap=cm_light) # 预测值的显示
plt.scatter(x[:, 0], x[:, 1], c=y[:,], s=30, cmap=cm_dark) # 样本
plt.scatter(test_data[:, 0], test_data[:, 1], c=test_label[:,], s=30, edgecolors='k',
            zorder=2,
            cmap=cm_dark) # 圈中测试集样本点
plt.xlabel('花萼长度', fontsize=13)
plt.ylabel('花萼宽度', fontsize=13)
plt.xlim(x1_min, x1_max)
plt.ylim(x2_min, x2_max)
plt.title('鸢尾花SVM二特征分类')
plt.show()

```

准确率输出

```

# 4. 计算svc分类器的准确率
print("训练集: ", classifier.score(train_data, train_label))
print("测试集: ", classifier.score(test_data, test_label))

---输出---
训练集:  0.8666666666666667
测试集:  0.6888888888888889

```

输出决策函数

```

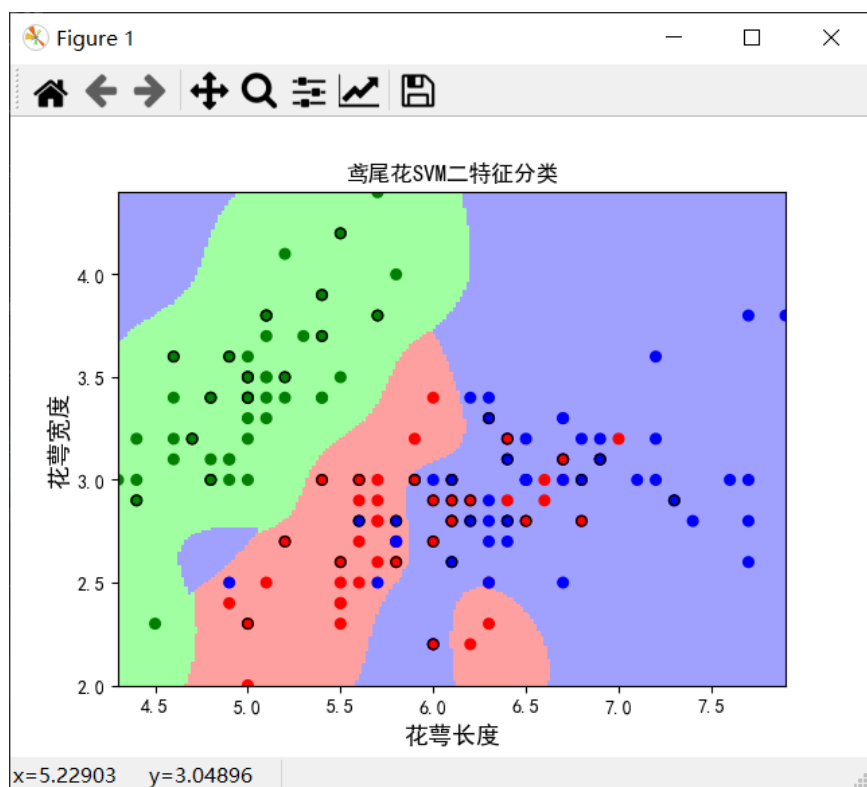
# 查看决策函数
print('train_decision_function:\n', classifier.decision_function(train_data))

```

输出:

```
train_decision_function:
[[-1.00004209 -0.23343617  0.99995092]
 [-1.0436294  -1.00035838 -1.0001592 ]
 [-0.70263184 -1.00000186 -1.10240085]
 [-0.72238801 -1.00000603 -0.70759974]
 [-0.30594852 -0.99963117 -1.00039512]
 [-1.0104992  -0.85005122  0.99959351]
 .....(省略).....
 [-1.00023309 -1.00022104  0.46119875]
 [-0.0498111  -1.00004048 -1.00004235]
 [ 1.06862501  1.09626808 -0.25200951]]
```

多分类可视化



输出支持向量

```
# 输出支持向量
print(classifier.support_vectors_)
```

输出：

```
[[4.3 3. ]  
 [5.7 4.4]  
 [4.6 3.4]  
 [4.8 3. ]  
 [4.5 2.3]  
 ....(省略)..  
 [6.7 3.1]  
 [5.8 2.7]  
 [6.3 2.9]  
 [7.7 3.8]]
```

聚类

一、问题定义

基本表示

- 训练样本集合
- 类（组，簇，cluster）

可能具有一定内在联系的个体群。第 i 个类表示为 c_i ，类的个数一般表示为 K ， $K < m$ 。

- 聚类中心

聚类本身是**集合的划分**问题

无标签 无监督

距离度量

闵可夫斯基距离：

- $p = 1$ 曼哈顿距离
- $p = 2$ 欧几里得距离

常见聚类方法

- 层次聚类
 - 单连接(single linkage)：最相似的成员
 - 全连接(complete linkage)：最不相似成员
 - 平均连接: 所有样本之间的平均值
- 基于质心的聚类
- 基于概率分布的聚类

- 基于密度的聚类
- 基于图的聚类

样本数据作为**图的顶点**，根据数据之间的距离构造边，形成带权图
对图进行切割，**分成多个子图，就是多个类**

二、基于质心的算法K-means算法

K-mean算法

- 1、初始化K个聚类中心
- 2、为每个个体分配聚类中心

把每个样本分类，该样本**离哪个聚类中心近**，放入哪个类中

- 3、移动聚类中心
- 4、重复迭代, 直到聚类中心不变或者变化很小

如何保证收敛：

- J函数表示每个**样本点到其质心的距离平方和**。
- 假设当前J没有达到最小值，那么首先可以**固定每个类的质心，调整每个样例所属的类别**来让J函数减少
- **固定所属的类别，调整每个类的质心**也可以使J减小。
- 这两个过程就是内循环中使J单调递减的过程。当J递减到最小时，聚类中心和所属类别同时收敛。

- 优点
 - 1.原理简单，实现方便，收敛速度快；
 - 2.聚类效果较优；
 - 3.模型的可解释性较强；
 - 4.调参只需要簇数k；
- 缺点
 - 1.k的选取不好把握；
 - 2.初始聚类中心的选择；
 - 3.如果数据的类型不平衡，比如数据量严重失衡或者类别的方差不同，则聚类效果不佳；
 - 4.采用的是迭代的方法，只能得到局部最优解；
 - 5.对于噪声和异常点比较敏感。

三、基于概率分布的聚类

EM算法

- E步是确定隐含类别变量所属类别

- E步就是估计隐含类别 y 的期望值
- M步更新其他参数（聚类中心）来使J最小化。
 - 极大似然估计 $P(x,y)$ 能够达到极大值。
 - 然后在其他参数确的情况下，重新估计 y 。

- 算法步骤：
 - E步：计算期望（E），利用对隐藏变量（类别）的现有估计值，计算最大似然估计值
 - M步：最大化（M），最大化在E步上求得的最大似然值来计算参数的值。
 - M步上找到的参数估计值被用于下一个E步计算中，这个过程不断交替进行。

四、基于密度的聚类

此类算法假设聚类结构能通过样本分布的**紧密程度**来决定。

通常情况下，密度聚类算法从**样本密度**的角度来考察样本之间的可连接性，并**基于可连接样本不断扩展聚类簇**来获得最终的聚类结果。

DBSCAN算法

- 基本概念
- 簇：由密度可达关系导出的最大密度相连样本集合。
- 算法步骤：

五、性能度量

- 外部指标：Jaccard系数、FM系数、Rand系数
- 内部指标：簇C内样本间的平均距离、簇C内样本间的最远距离、簇 C_i 与簇 C_j 最近样本间的距离、簇 C_i 与簇 C_j 最近样本间的距离
- 基本思想：“**簇内相似度**”高且“**簇间相似度**”低

六、K-means以及DBSCAN实例

数据集描述

源于google的**广告关键词推荐页面**，样本属性包括**关键词、编号、月均搜索量、竞争力、估价以及关键词排名**，包含六种属性。

该数据集的 **shape** 为 **266*6**（即266条数据，每个样例六个属性）

部分数据集截图：

256	财经网	251	1900	0.22	13.38	123
257	今日股市行情	255	2400	0.06	8.12	67
258	股價	257	3600	0.08	11.08	97
259	股票分析	258	4400	0.06	11.38	100
260	财经	259	5400	0.15	10.31	91
261	股票行情	262	8100	0.15	10.22	90
262	基金	263	22200	0.41	9.81	87
263	美股	264	27100	0.06	8.62	73
264	股票	265	74000	0.27	14.87	130
265	股票交易平台	87	50	0.43	253.84	265
266	股票期权	222	390	0.05	380.25	266

部分数据如下：

关键词	编号	搜索量	竞争价值	估价	排名
股票学习网	8	20	0.11	27.19	193
股票初学	15	20	0.16	22.41	171
指数股票	16	20	0.07	26.66	191
怎样看股票	18	20	0.14	18.93	155
股票入门教程	30	20	0.11	17.5	149
购买股票	31	20	0.2	23.75	180
股票交流	35	30	0.11	19.17	160
中国股市论坛	44	30	0.16	23.98	182
上海股票指数	50	30	0.04	29.41	196
股票开户流程	54	30	0.1	25.71	187

本实验中选择了 **竞争价值** 和 **股价** 两个属性进行聚类 and 可视化。

肘方法选择k值

手肘法核心思想

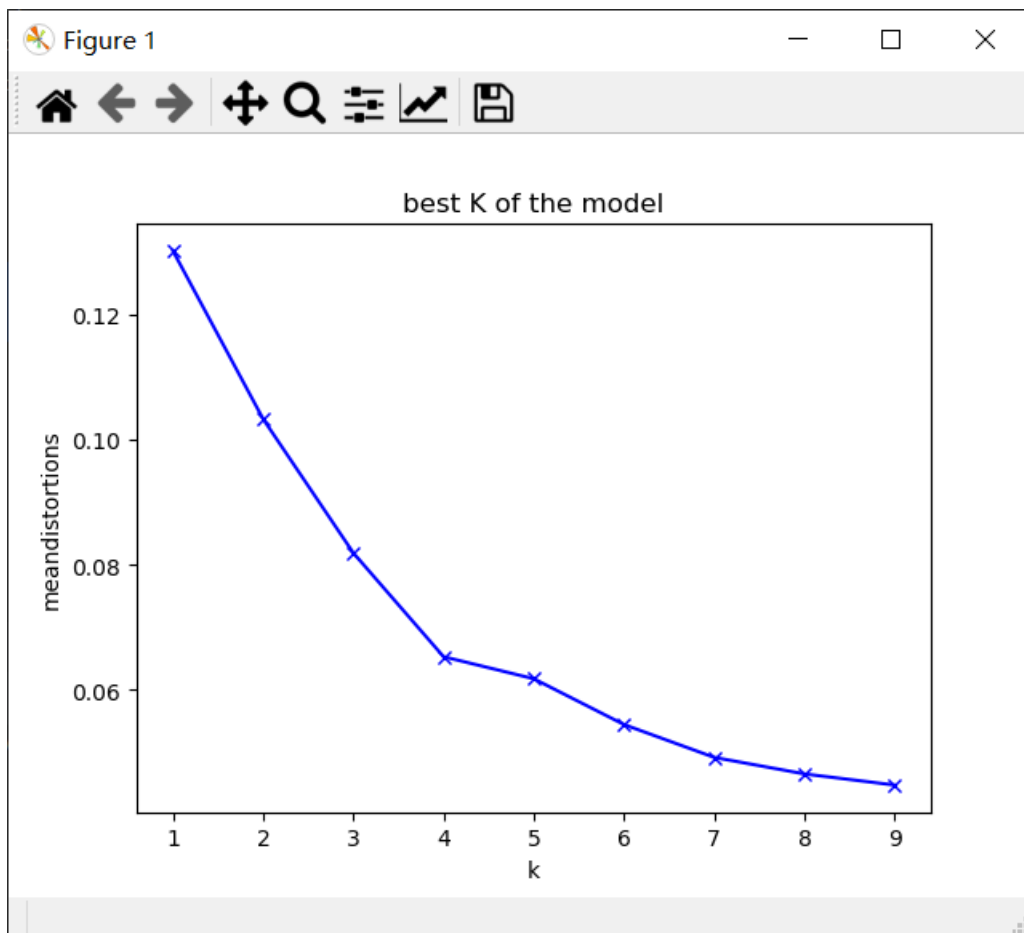
- 随着聚类数k的增大，样本划分会更加精细，每个簇的聚合程度会逐渐提高，那么误差平方和SSE自然会逐渐变小。
- 当k小于真实聚类数时，由于k的增大会大幅增加每个簇的聚合程度，故SSE的下降幅度会很大，而当k到达真实聚类数时，再增加k所得到的聚合程度回报会迅速变小，所以SSE的下降幅度会骤减，然后随着k值的继续增大而趋于平缓，也就是说SSE和k的关系图是一个手肘的形状，而这个肘部对应的k值就是数据的真实聚类数

```
• 代码

# 肘方法选择最优的K值
```

```
def try_K():
    X = loadData()
    K = range(1, 10)
    meandistortions = []
    for k in K:
        kmeans = KMeans(n_clusters=k)
        kmeans.fit(X)
        meandistortions.append(sum(np.min(cdist(X, kmeans.cluster_centers_,
'euclidean'), axis=1)) / X.shape[0])
    plt.plot(K, meandistortions, 'bx-')
    plt.xlabel('k')
    plt.ylabel('meandistortions')
    plt.title('best K of the model');
    plt.show()
```

- 运行效果



根据上图可知当k=4时对应“肘部”，所以 k=4 是最优的 k 值

聚类并可视化聚类结果

- 代码

```
# 聚类 并可视化聚类结果
def Kmeans():
    data = loadData()
    # 类簇的数量
    n_clusters = 6
    # 调用Kmeans聚类
```

```

cls = KMeans(n_clusters).fit(data)
# 输出X中每项所属分类的一个列表
print("样本所属分类: ",cls.labels_)

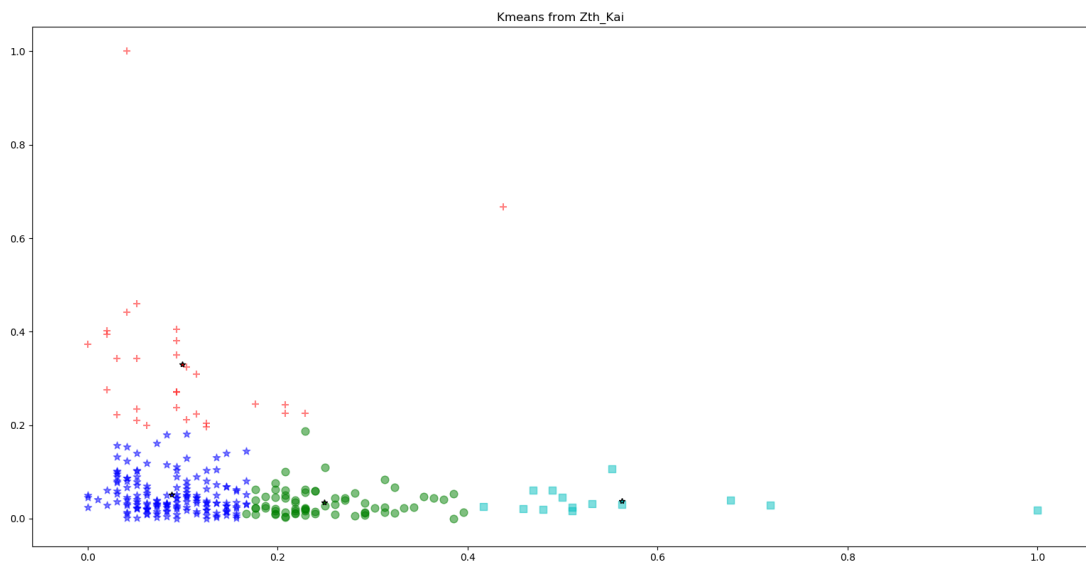
# 可视化
markers = ['*', 'o', '+', 's', 'v', '8']
cl = ['b', 'g', 'r', 'c', 'm', 'y', 'k']
centers = cls.cluster_centers_ # 各类别中心
print("聚类中心: ", centers)
for i in range(n_clusters):
    members = cls.labels_ == i # members是布尔数组 表示是否是此i类的
    plt.scatter(data[members, 0], data[members, 1], s=60, marker=markers[i],
c=cl[i], alpha=0.5) # 画与members数组中匹配的点
    plt.plot(centers[i][0], centers[i][1], '*', markerfacecolor=cl[i],
markeredgecolor='k', markersize=6) # 画出聚类中心

plt.title('Kmeans from Zth_Kai')
plt.show()

```

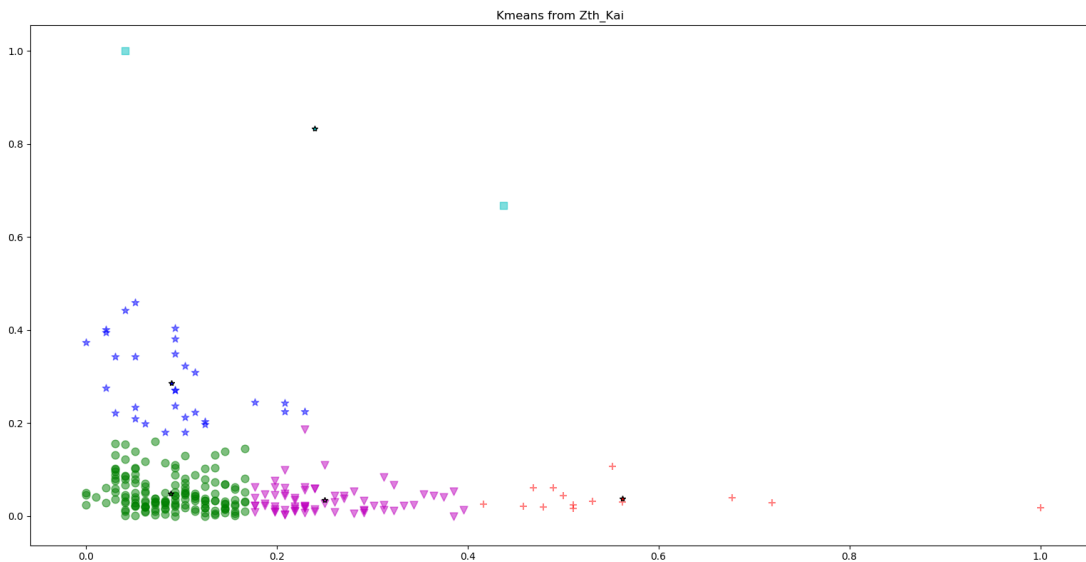
- 运行效果截图

- o k=4

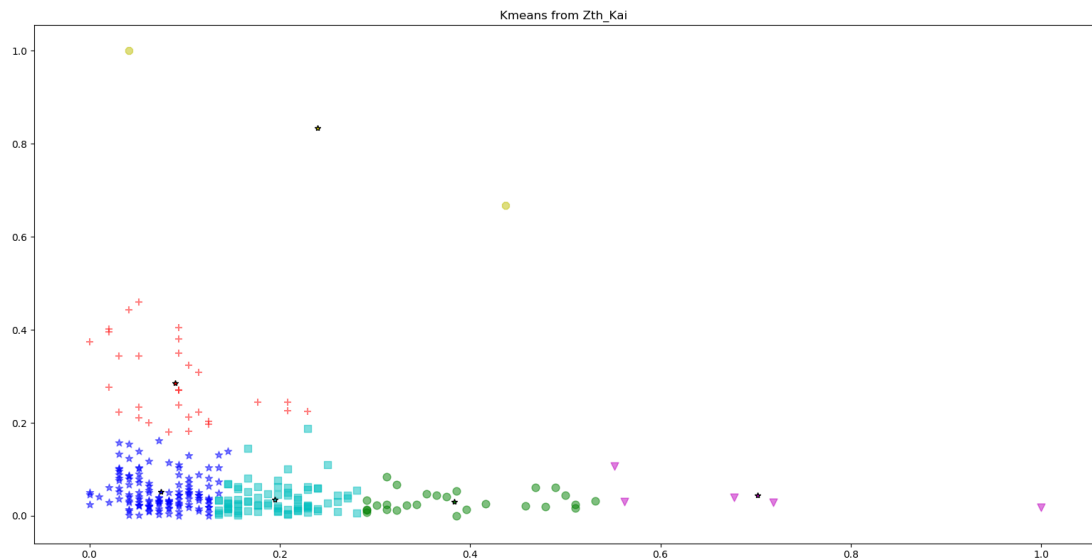


其中的 * 表示聚类中心

- o k=5



- o $k=6$



- 样本所属分类和聚类中心输出 ($k=4$) :

```
D:\Anaconda\python.exe D:/360MoveData/Users/38004/Desktop/MyGitFile/Python学习/MachineLea
样本所属分类: [0 0 0 0 0 3 0 0 0 0 3 3 0 0 3 0 0 0 3 0 0 0 3 0 0 3 3 0 3 0 0 0
0 1 0 0 1 0 3 0 0 0 3 3 0 3 3 3 3 0 0 0 0 1 0 3 0 2 2 2 2 0 2 2 2
2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 3 0 3 0 0 3 0 3 3 3 0 0 0 3 0
0 3 0 3 0 1 3 3 3 3 0 0 0 0 3 1 3 0 0 0 0 3 0 0 0 0 0 0 3 3 0 0 3 3
0 3 0 1 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 3 0 0 0 0 1 3 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 3 0 3 1 3 1 0 3 0 0
3 0 3 3 0 0 1 0 3 0 0 3 0 3 1 0 3 1 0 0 0 3 3 0 3 3 0 0 0 3 0 0 3 3 0 0 0
0 0 1 0 3 2 2]
聚类中心: [[0.08860931 0.05109962]
[0.5625      0.03791361]
[0.09965278 0.32978386]
[0.24908088 0.0347359 ]]
```

规范化前后对比 使用平均轮廓系数分析聚类效果

- 规范化代码 (在读取代码部分)

```
# 读取数据
def loadData():
    data = []
    fr = open('KmeansDataset.data', 'r', encoding='UTF-8')
    # 选择竞争价值和估价两个属性
    for data_i in fr:
        data.append([float(data_i.split()[3]), float(data_i.split()[4])])

    data = np.array(data)

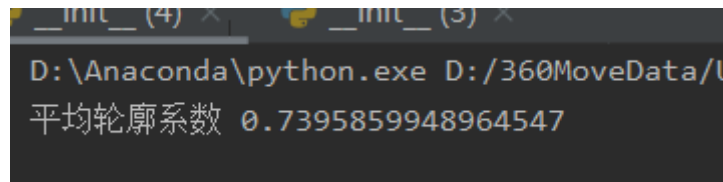
    # 数据规范化
    min_max_scaler = preprocessing.MinMaxScaler()
    data = min_max_scaler.fit_transform(data)
    # print(data)
    return data
```

规范化数据为**竞争价值**和**估价**两个属性

- 使用平均轮廓系数分析聚类效果代码 ($k = 4$)

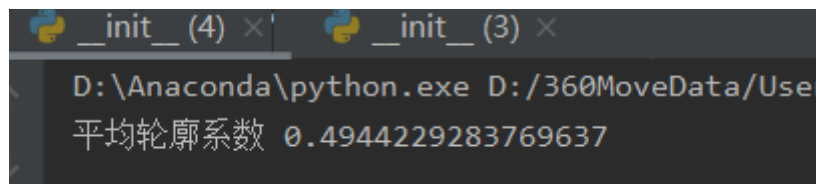
```
# 用平均轮廓系数分析聚类效果
def showClusteringEffect():
    X = loadData()
    cls = KMeans(4).fit(X)
    y_pre = cls.predict(X)
    silhouette_s = metrics.silhouette_score(X, y_pre, metric='euclidean') # 平均轮廓系数
    print("平均轮廓系数", silhouette_s)
```

规范化前:



```
__init__(4) x | __init__(3) x
D:\Anaconda\python.exe D:/360MoveData/User
平均轮廓系数 0.7395859948964547
```

规范化后:



```
__init__(4) x | __init__(3) x
D:\Anaconda\python.exe D:/360MoveData/User
平均轮廓系数 0.4944229283769637
```

选做：DBSCAN算法实现该数据集聚类

- 选择“邻域”参数
- DBSCAN代码

```
def myDbscan():
    X = loadData()
    print(X.shape)

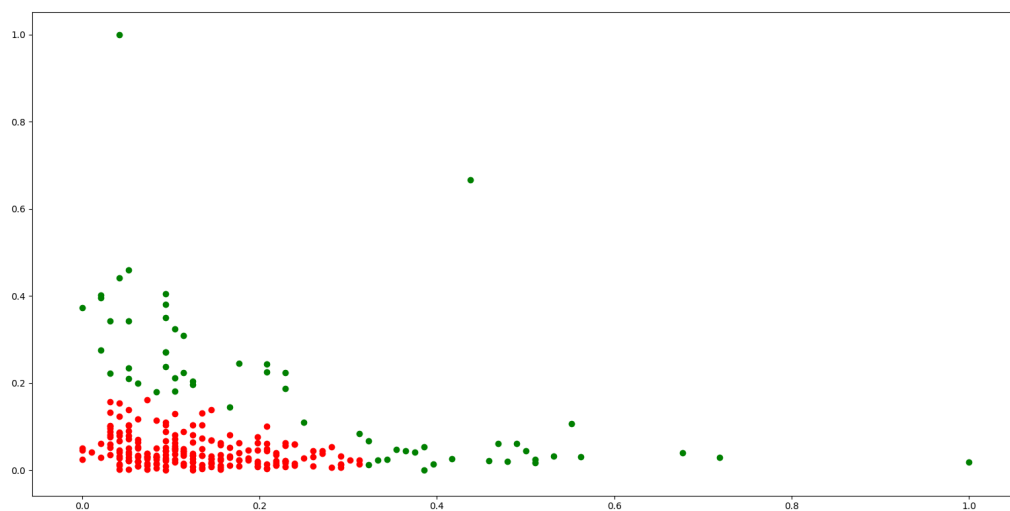
    dbscan = DBSCAN(eps=0.05, min_samples=20)
    dbscan.fit(X)

    label_pred = dbscan.labels_
    print(label_pred)
    x0 = X[label_pred == 0]
    x1 = X[label_pred == -1]
    plt.scatter(x0[:, 0], x0[:, 1], c="red", marker='o', label='label0')
    plt.scatter(x1[:, 0], x1[:, 1], c="green", marker='o', label='label1')
    plt.show()
```

- 输出:

```
__init__ (4) × __init__ (3) ×
D:\Anaconda\python.exe D:/360MoveData/Users/38004/Desktop/MyGitFile/Python学
(266, 2)
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0 -1  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0 -1  0  0  0  0 -1
 -1  0  0 -1  0 -1  0  0  0  0 -1  0  0  0 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  0 -1
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0
  0  0  0  0  0  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0
  0  0  0  0  0  0  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0 -1  0  0  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 -1  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0  0 -1
 -1 -1  0  0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0 -1 -1  0  0 -1
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0
 -1 -1]
```

- 可视化截图



该基于密度的聚类算法将样本分成了两个类

完整代码

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Time : 2020/6/10 11:04
# @Author : Kai
# @File : __init__.py.py
# @Software: PyCharm

import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn import preprocessing
from scipy.spatial.distance import cdist
from sklearn import metrics
from sklearn.cluster import DBSCAN
# 读取数据
def loadData():
    data = []
    fr = open('KmeansDataset.data', 'r', encoding='UTF-8')
    # 选择竞争价值和估价两个属性
```

```

for data_i in fr:
    data.append([float(data_i.split()[3]), float(data_i.split()[4])])

data = np.array(data)

# 数据规范化
min_max_scaler = preprocessing.MinMaxScaler()
data = min_max_scaler.fit_transform(data)
# print(data)
return data

# 聚类 并可视化聚类结果
def Kmeans():
    data = loadData()
    # 类簇的数量
    n_clusters = 4
    # 调用Kmeans聚类
    cls = KMeans(n_clusters).fit(data)
    # 输出X中每项所属分类的一个列表
    print("样本所属分类: ", cls.labels_)

    # 可视化
    markers = ['*', 'o', '+', 's', 'v', '8']
    cl = ['b', 'g', 'r', 'c', 'm', 'y', 'k']
    centers = cls.cluster_centers_ # 各类别中心
    print("聚类中心: ", centers)
    for i in range(n_clusters):
        members = cls.labels_ == i # members是布尔数组 表示是否是此i类的
        plt.scatter(data[members, 0], data[members, 1], s=60, marker=markers[i],
c=cl[i], alpha=0.5) # 画与members数组中匹配的点
        plt.plot(centers[i][0], centers[i][1], '*', markerfacecolor=cl[i],
markedgecolor='k', markersize=6) # 画出聚类中心

    plt.title('Kmeans from Zth_Kai')
    plt.show()

# 肘方法选择最优的K值
def try_K():
    X = loadData()
    K = range(1, 10)
    meandistortions = []
    for k in K:
        kmeans = KMeans(n_clusters=k)
        kmeans.fit(X)
        meandistortions.append(sum(np.min(cdist(X, kmeans.cluster_centers_,
'euclidean'), axis=1)) / X.shape[0])
    plt.plot(K, meandistortions, 'bx-')
    plt.xlabel('k')
    plt.ylabel('meandistortions')
    plt.title('best K of the model');
    plt.show()

# 用平均轮廓系数分析聚类效果
def showClusteringEffect():
    X = loadData()
    cls = KMeans(4).fit(X)
    y_pre = cls.predict(X)

```

```

silhouette_s = metrics.silhouette_score(X, y_pre, metric='euclidean') # 平均轮廓系
数
print("平均轮廓系数", silhouette_s)

# DBSCAN 基于密度的聚类
def myDbscan():
    X = loadData()
    print(X.shape)

    dbscan = DBSCAN(eps=0.05, min_samples=20)
    dbscan.fit(X)

    label_pred = dbscan.labels_
    print(label_pred)
    x0 = X[label_pred == 0]
    x1 = X[label_pred == -1]
    plt.scatter(x0[:, 0], x0[:, 1], c="red", marker='o', label='label0')
    plt.scatter(x1[:, 0], x1[:, 1], c="green", marker='o', label='label1')
    plt.show()

if __name__ == "__main__":
    try_K()
    Kmeans()
    showClusteringEffect()
    # myDbscan()

```

机器学习
