第 1 章 机器学习概念理解 1
1.1 基本术语 1
第 2 章 算法分类 2
2.1 聚类 2
2.1.1 聚类任务 2
2.1.2 聚类的性能指标 2
2.1.3 Kmean 算法 2
数据集一：HTRU_2 5
数据集二：seeds 5
数据集三：Wholesale customers 6
2.2 多元分类 7
决策树 7
2.2.1 C4.5 算法 7
数据集一：Iris 9
数据集二：Car 10
数据集三：Balance Scale 11
2.2.2 CART 算法 12
数据集一：Iris 14
数据集二：Car 15
数据集三：absenteeism_at_work 16
2.2.3 KNN 算法 18
数据集一：Iris 19
数据集二：absenteeism_at_work 20
数据集三：Seeds 21
2.2.4 朴素贝叶斯 22
2.3 统计学习 22
2.3.1 支持向量机 SVM 22
2.4 集成学习 23
2.4.1 AdaBoost 算法 23
2.4.2 随机森林 24

第 1 章 机器学习概念理解

1.1 基本术语

关于西瓜的数据

编号	色泽	根蒂	敲声	纹理	密度	含糖率	好瓜
1	青绿	蜷缩	浊响	清晰	0.697	0.460	是
2	乌黑	稍缩	沉闷	清晰	0.360	0.370	否
3	乌黑	蜷缩	沉闷	清晰	0.774	0.376	是
4	青绿	硬挺	清脆	清晰	0.243	0.267	否
5	浅白	蜷缩	浊响	模糊	0.343	0.099	否

数据集(data set)：如上图表关于西瓜信息记录的一个**集合**。通常可将数据集分为训练集和测试集。

训练集(train set)：在数据中习得模型的过程中所使用的数据称为**训练数据**，其中的每一个样本称为**训练样本**，样本组成的集合即为**训练集**。

测试集(testing set)：通过训练集学的模型后，使用该模型进行预测评估的过程称为“测试”，被预测的样本称为**测试样本**。**测试样本**组成的集合称为**测试集**。

样本(sample)：数据集中关于一个事件或对象(一个西瓜)的描述，例如上图表的关于某个西瓜的一组数据，如(色泽=青绿，根蒂=蜷缩，敲声=浊响，纹理=清晰，密度=0.697，含糖率=0.460)。**样本**又称为**示例**(instance)。

属性(attribute)：反应事件或对象(一个西瓜)在某方面的表现或性质，例如“色泽“，”纹理”等。**属性**又称为**特征**(feature)。根据属性值的性质可将属性分为连续属性与离散属性。

连续属性：属性值在定义区间内可任意取值的属性。例如西瓜的密度与含糖率。

离散属性：属性值为离散，在定义区间取值有限的属性。例如色泽有“青绿”，“乌黑”等取向。

属性值(attribute value)：属性值上的取值，例如“青绿”，“清晰”。

属性空间(attribute space：由属性张成的空间，又称为**样本空间**(sample space)或**输入空间**。例如将“色泽”，：根蒂“，”敲声“作为坐标轴，则他们张成一个描述西瓜的三维空间。由于空间中每个点(一个西瓜示例)对 k 应一个坐标向量，也将一个示例称为一个特征向量。

标签(label)：训练集合的结果信息，例如”好瓜“作为示例结果：”（(色泽=青绿，根蒂=蜷缩，敲声=浊响，纹理=清晰，密度=0.697，含糖率=0.460)，好瓜）“。

样例(example)：拥有了标签信息的样本。

监督学习：训练数据包含标签的学习任务。典型代表“分类”和“回归”。

无监督学习：训练样本的标记信息未知，目标是通过对无标记的学习来揭示数据的内在性质及规律。典型代表“聚类”。

泛化能力：学的模型的适用于新样本的能力。泛化能力强的模型能更好的适用于整个样本空间。

分类：分类方法是一种对离散型随机变量建模或预测的监督学习算法。常用的分类算法有决策树，朴素贝叶斯，支持向量机等。

回归：回归方法是一种对数值型连续随机变量进行预测和建模的监督学习算法。常用的回归算法有，最近邻居法(KNN)等。

1.2 机器学习与理解

我们正处于一个大数据的时代，通过查询，全球每日产生的数据已经达到了 ZB 量级（万亿 GB）,这是个天文数字，而且这个数字还在以指数的增长趋势膨胀着。这就带来了新的问题，如何有效地，在数据的沙滩上淘金呢，即如何有效地提取出关键的信息并加以分析，由此大数据应运而生。通过这个阶段的学习，我对于数据的挖掘与分析有了更加深刻的认知。在实验中，我虽然只做了 4 个算法的具体实现，但是其中碰到了许许多多的问题，首先时算法的问题，例如在决策树算法法，如何表示树的存储形式，如何对数据进行有效的处理，代码怎样写的简练而不冗余，我查询了大量的资料，对于每个算法仔细斟酌，一步步推导。其中也有了不少发现，在 CS 课程中，我们常听老师说程序的核心是算法与数据结构。但这个论述也可以放在机器学习中。当然机器学习中最重要的还是算法，而算法的核心我觉得时数学。不管多酷炫的算法其实有与数学息息相关，你离不开我，我离不开你。仔细回想，到目前的算法学习中，我用到了高数的知识，线代的知识，还有概率论以及数值分析。有些我还正在学习，而更多的，其实以往的差不多了，所以经常碰到某个公式不得不拿起原来的课本在温习一遍，同时也比较懊恼原来那些通识课程学的并不是很踏实。所以啊，数学的加强巩固仍然任重道远，不过我很喜欢这种感觉，就是将所学课本知识不在囿于理论学习，能够在实际学习与生活中用来解决问题，那种感觉真好

第 2 章 算法分类

2.1 聚类

2.1.1 聚类任务

聚类算法试图将数据集中的样本划分为若干个通常是不相交的子集，每个子集为一个簇。作为无监督学习的一个典型代表，聚类的对象为无标记信息的训练样本，聚类通过对样本的学习来揭示数据的内在性质及规律。聚类过程中自动生成簇结构，我们可根据簇分析归类诸如“浅色瓜”，“扇形输液”等，但是聚类前的结果无法预知，换言之，分类簇的概念语义由人来归纳发觉。

2.1.2 聚类的性能指标

聚类性能度量可根据有无参考模型分为外部指标和内部指标。外部指标，即根据一些领域专家的划分结果作为参考模型与通过算法聚类的结果作相似度的比较。常有的性能度量外部指标有：Jaccard 系数、FM 指数，Rand 指数等；与之相反内部指标的性能度量仅直接根据聚类的结果来分析簇内的相似度以及簇键的相似度进行性能评价，常有的聚类性能度量内部指标有：DBI 及 DI。DBI 的值越小越好，DI 则越大越好。

2.1.3 Kmean 算法

K-mean 算法又称 K 均值算法。该算法首先初始化一个 K 阶均值向量(随机抽样至数据集)，通过对测试集样本的层层迭代，对每一个样本根据其距离 K 个均值向量的“距离”，按照最小距离进行分类处理成 K 阶簇。再对 K 阶簇取其平均向量，再进行新一轮的迭代，直到分类簇的变化趋于稳定。最后根据性能度量单位来评估聚类结果。

代码：

```python
import csv
import random
import math
from functools import reduce

class Kmeans:
    def __init__(self,file_name,k):
        self.k = k #聚类为K簇
        self.file_reader(file_name)
        self.set_meanvec() #随机生成初始k个均值向量
        self.H = [[] for i in range(k)]  #H装载k个分类簇

    def file_reader(self,file_name):#读取文件提取数据集
        self.D = [] #数据集dadaset
        with open(file_name, newline='') as csvfile:
            # List = csvfile.readline()
            dataset = csv.reader(csvfile)
            for row in dataset:
                for i in range(len(row) - 1):
                    try:
                        row[i] = float(row[i])
                    except ValueError:
                        row[i] = str(row[i])
                    else:
                        pass
                self.D.append(row)



    def set_meanvec(self): #随机生成初始k个均值向量
        randomset = random.sample(range(len(self.D)), self.k) #在数据集中随机选取K个作为初始向量
        self.meanvector = []
        for i in sorted(randomset):
            self.meanvector.append(self.D[i])
```

```python

    def avgC(self,C):
        """计算一个簇的所有样本的均值样本"""
        return reduce(self.avgc,C)#使用reduce来对簇内所有元素求平均

    def DBI(self):
        """计算DBI性能度量"""
        maxTmp = 0
        for i in range(self.k):
            maxI = 0
            for j in range(self.k):
                if i != j:
                    db = (self.avgdist(self.H[i])+self.avgdist(self.H[j]))/\
                         self.dist(self.avgC(self.H[i]),self.avgC(self.H[j]))
                    if db > maxI:
                        maxI = db
            maxTmp += maxI
        return maxTmp/self.k

    def DI(self):
        """计算DI性能度量"""
        diams = []
        dmins = []
        for k in range(self.k):
            diams.append(self.diam(self.H[k]))
        maxdiam = max(diams)
        for i in range(0,self.k - 1,1):
            for j in range(i + 1,self.k,1):
                it = self.dmin(self.H[i],self.H[j])
                dmins.append(it)
        mindmin = min(dmins)

        return mindmin / maxdiam
```

```
68
69      def dmin(self,C1,C2):
70          """求得两个簇之间的最小距离"""
71          dmin_ = self.dist(C1[0],C2[0])
72          for c1 in C1:
73              for c2 in C2:
74                  distanse = self.dist(c1,c2)
75                  if distanse < dmin_:
76                      dmin_ = distanse
77          return dmin_
78
79      def diam(self,C):
80          """计算一个簇中两个任意两个分类样本的最大距离"""
81          for i in range(len(C) - 1):
82              diamC = self.dist(C[0], C[1])
83              for j in range(i + 1, len(C)):
84                  d = self.dist(C[i], C[j])
85                  if d > diamC:
86                      diamC = d
87          return diamC
88
89      def a_loop(self):
90          """一次迭代"""
91          for Ci in self.D: #迭代过程中为每一个样本根据与均值向量的最小距离聚类成簇列表(self.H)
92              key = 0
93              dis_min = self.dist(self.D[0],self.meanvector[0])
94              for i in range(len(self.meanvector)):
95                  distance = self.dist(Ci,self.meanvector[i])
96                  if distance <= dis_min:
97                      dis_min = distance
98                      key = i
99              self.H[key].append(Ci)
```

```
101     def loop_for_n_times(self,n):
102         """多重迭代""" #迭代的终止情况有1迭代此处到达指定次数2分类簇内的平均距离与上次差异不大(趋于稳定)
103         for i in range(n):
104             self.H = [[] for j in range(self.k)] #每次循环后重置k阶分类簇
105             self.a_loop() #一次循环将测试集聚类
106             test_mvc = self.meanvector[:]
107             self.avgvecs() #将循环得到的簇求其均值向量
108             if self.k * abs(self.avgdist(test_mvc) - self.avgdist(self.meanvector)) < 1e-10:
109                 break
110         return i+1
111
112     def avgvecs(self):
113         """根据聚类的簇,求得平均向量并将对应下标的均值限量覆盖"""
114         self.meanvector = [] #重置均值向量
115         for x in range(self.k):
116             self.meanvector.append(self.avgC(self.H[x]))
117
118
119     def dist(self,a, b):
120         """计算两个样本a,b之间的距离度量"""
121         i = s = 0
122         while i < len(a):
123             if type(a[i]) is float or type(a[i]) is int:
124                 s += (a[i] - b[i])**2
125                 i += 1
126             else:
127                 break
128         return math.sqrt(s)
129
130     def avgdist(self,C):
131         """计算一个簇的所有距离的均值"""
132         s = t = 0
133         for i in range(len(C) - 1):
134             for j in range(i + 1, len(C)):
135                 s += self.dist(C[i], C[j])
136                 t += 1
137         return s / t
```

```
    def avgc(self,a,b):
        """计算两个样本向量的均值样本向量"""
        i = 0
        c = []
        while i < len(a):
            if type(a[i]) is float or type(a[i]) is int:
                c.append((a[i] + b[i]) / 2)
            i += 1
        return c

    def show_result(self):
        """菜单函数,展示聚类结果"""
        print("测试数量：",len(self.D),"属性数量:",len(self.D[0]))
        print("聚类结果" + str(self.k) + "簇.")
        for i,h in enumerate(self.H):
            print("第{0}簇,数量为:{1}".format(i+1,len(self.H[i])))
        print("聚类后的均值向量:")
        for mv in self.meanvector:
            print(mv)
        print("DBI:",self.DBI(),"DI",self.DI())
```

数据集一：HTRU_2

```
140.5625,55.68378214,-0.234571412,-0.699648398,3.199832776,19.11042633,7.975531794,74.24222492,0
102.5078125,58.88243001,0.465318154,-0.515087909,1.677257525,14.86014572,10.57648674,127.3935796,0
103.015625,39.34164944,0.323328365,1.051164429,3.121237458,21.74466875,7.735822015,63.17190911,0
136.75,57.17844874,-0.068414638,-0.636238369,3.642976589,20.9592803,6.89649891,53.59366067,0
88.7265625,40.67222541,0.600866079,1.123491692,1.178929766,11.4687196,14.26957284,252.5673058,0
93.5703125,46.69811352,0.53190485,0.416721117,1.636287625,14.54507425,10.6217484,131.3940043,0
119.484375,48.76505927,0.03146022,-0.112167573,0.99916388,9.279612239,19.20623018,479.7565669,0
130.3828125,39.84405561,-0.158322759,0.389540448,1.220735786,14.37894124,13.53945602,198.2364565,0
107.25,52.62707834,0.452688025,0.170347382,2.331939799,14.48685311,9.001004441,107.9725056,0
107.2578125,39.49648839,0.465881961,1.162877124,4.079431438,24.98041798,7.397079948,57.78473789,0
142.078125,45.28807262,-0.320328426,0.283952506,5.376254181,29.00989748,6.076265849,37.83139335,0
133.2578125,44.05824378,-0.081059862,0.115361506,1.632107023,12.00780568,11.97206663,195.5434476,0
134.9609375,49.55432662,-0.135303833,-0.080469602,10.69648829,41.34204361,3.893934139,14.13120625,0
117.9453125,45.50657724,0.325437564,0.661459458,2.836120401,23.11834971,8.943211912,82.47559187,0
```

测试代码：

```
file_name = "/home/ubuntu/PycharmProjects/Kmeans/venv/Data/HTRU_2.csv"
test = Kmeans(file_name,k=2)
print("HTRU数据集","迭代次数:",test.loop_for_n_times(n=10000))
test.show_result()
```

测试结果：

```
/home/ubuntu/PycharmProjects/Kmeans/venv/bin/python /home/ubuntu/PycharmProjects/Kmeans/venv/bin/Iris.py
HTRU数据集 迭代次数: 7
测试数量： 17898 属性数量: 9
聚类结果3簇.
第1簇,数量为:247
第2簇,数量为:7069
第3簇,数量为:10582
聚类后的均值向量:
[117.85803749019298, 51.32640938930792, 0.08433008267339058, -0.031893930741458326, 0.48212365966899273, 8.36181809272329, 25.67269585426412, 745.
[89.23218512267398, 69.75133164112299, 0.7786976756078139, -0.05568907014975069, 103.42060215349596, 56.88499982103271, 0.8473103381977772, 6.7004
[118.66818440656354, 53.66762033209456, 0.11572981724335432, -0.1772798394082649, 1.6767436083783527, 13.23370910820839, 12.055716613229453, 189.4
DBI: 0.6497155237225192 DI 0.008222835269250238
```

数据集二：seeds

```
15.26,14.84,0.871,5.763,3.312,2.221,5.22,1
14.88,14.57,0.8811,5.554,3.333,1.018,4.956,1
14.29,14.09,0.905,5.291,3.337,2.699,4.825,1
13.84,13.94,0.8955,5.324,3.379,2.259,4.805,1
16.14,14.99,0.9034,5.658,3.562,1.355,5.175,1
14.38,14.21,0.8951,5.386,3.312,2.462,4.956,1
14.69,14.49,0.8799,5.563,3.259,3.586,5.219,1
14.11,14.1,0.8911,5.42,3.302,2.7,5,1
16.63,15.46,0.8747,6.053,3.465,2.04,5.877,1
16.44,15.25,0.888,5.884,3.505,1.969,5.533,1
15.26,14.85,0.8696,5.714,3.242,4.543,5.314,1
14.03,14.16,0.8796,5.438,3.201,1.717,5.001,1
13.89,14.02,0.888,5.439,3.199,3.986,4.738,1
13.78,14.06,0.8759,5.479,3.156,3.136,4.872,1
13.74,14.05,0.8744,5.482,3.114,2.932,4.825,1
```

测试代码：

```
file_name = "/home/ubuntu/PycharmProjects/Kmeans/venv/Data/seeds.csv"
test = Kmeans(file_name,k=5)
print("seeds数据集","迭代次数:",test.loop_for_n_times(n=10000))
test.show_result()
```

测试结果：

```
seeds数据集 迭代次数: 5
测试数量： 210 属性数量: 8
聚类结果5簇.
第1簇,数量为:21
第2簇,数量为:60
第3簇,数量为:22
第4簇,数量为:35
第5簇,数量为:72
聚类后的均值向量:
[12.576430673599244, 13.43397476196289, 0.8751456881523132, 5.230687118530273, 3.052963471412659, 6.680251635551452, 5.049086643218994]
[15.907597935485953, 15.116877500786025, 0.8745932682229639, 5.837653663523389, 3.3904303985142494, 3.407940763136346, 5.8347468493282 57]
[20.065713205337524, 16.924942202568054, 0.880276208305359, 6.438547664165497, 3.8360832710266113, 3.1931394677162173, 6.2018513584136965]
[17.943435697710957, 15.928743767997947, 0.8885800258420058, 6.097850696019071, 3.6739454417380037, 3.3784636155114276, 5.945289420402492]
[11.770676052209472, 13.133263610886722, 0.8570519117060074, 5.155543338251347, 2.85914667554126, 3.7946745637752692, 4.996914670731276]
DBI: 1.4991666009835694 DI 0.06195250099847528
```

数据集三：Wholesale customers

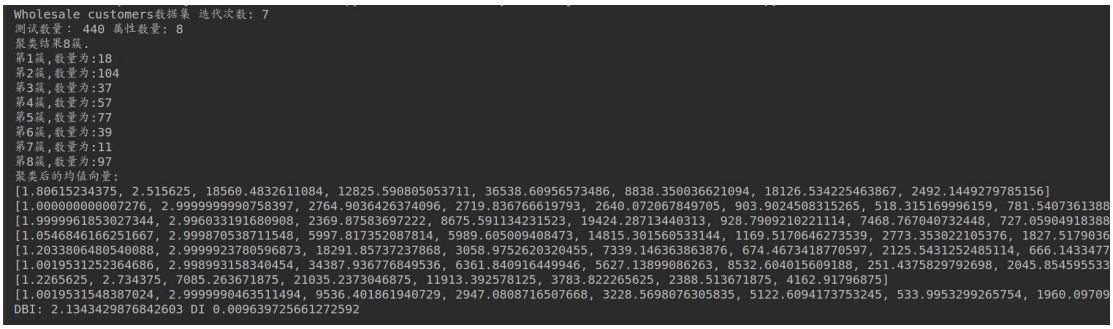
```
Channel,Region,Fresh,Milk,Grocery,Frozen,Detergents_Paper,Delicassen
2,3,12669,9656,7561,214,2674,1338
2,3,7057,9810,9568,1762,3293,1776
2,3,6353,8808,7684,2405,3516,7844
1,3,13265,1196,4221,6404,507,1788
2,3,22615,5410,7198,3915,1777,5185
2,3,9413,8259,5126,666,1795,1451
2,3,12126,3199,6975,480,3140,545
2,3,7579,4956,9426,1669,3321,2566
1,3,5963,3648,6192,425,1716,750
```

（第一行为属性名称，实际在读取数据集时跳过了第一行）

测试代码：

```
file_name = "/home/ubuntu/PycharmProjects/Kmeans/venv/Data/Wholesale customers data.csv"
test = Kmeans(file_name,k=8)
print("Wholesale customers数据集","迭代次数:",test.loop_for_n_times(n=10000))
test.show_result()
```

测试结果：



总结：Kmeans 算法先是随机取得 k 个初始向量，再通过对原始数据集一遍遍的迭代，逼近这些均值向量，从而划分为簇，知道均值向量最终趋于稳定，得到最终的向量及分类簇。这种算法思想十分类似《数值分析》中线性方程的求解（雅克比迭代，高斯-赛德尔迭代），看来数学时自然科学的桂冠还真不假。通过实验分析，k 的取值对分类簇影响深远，但是当 k 取值唯一时，相同的实验随着初始的随机向量不同，实验结果也有相当的差距，这也是我疑惑不解的地方，因为只找到了关于内部指标的数据集，只有 DBI，DI 等性能指标，没有外部指标的数据集，也就无法对测试结果进行有效验证。但是通过分析 DBI,DI 等指标，似乎没有太大的问题。

2.2 多元分类

在机器学习中，常常对数据集进行分类处理。多元分类即将数据集中的一个示例归类为多个类别中的一个（将示例归为两类中的一类称为二分类）。常见的分类算法有决策树和 KNN(K 近邻算法)。

决策树

决策树由一个决策图和可能的结果组成，用来创建到达目标的规划。在机器学习中，决策树是一个预测模型，他代表对象属性与对象值之间的一种映射关系，树中的每一个节点表示某个对象，而每一个分叉路径则代表某个可能的属性值，每个叶节点则对应从根节点到该节点所经历的路径所表示的对象的值。

在对数据集进行属性划分时，常根据划分的决策树分支节点的纯度来选择最优划分属性。衡量样本集纯度的常有指标有信息熵，信息增益率，基尼指数等等。信息熵描述了当前样本集合的纯度。而信息增益则为划分前后信息熵的差值，基尼指数与之类似。一般而言，信息熵越低则纯度越高，信息增益率越高则划分效率最高，基尼指数越低划分效率越高。

2.2.1 C4.5 算法

C4.5 算法根据信息熵的增益率来选择最优划分属性，具体实现如下：

代码：

```python
from math import *
import csv
from collections import Counter
from random import *

class C45:
    def __init__(self,filename):
        parameter "self" of method "__init__"
        self.dataset = [] #数据集
        self.file_reader(file_name=filename)
        self.myTree = None

    def file_reader(self,file_name):#读取文件提取数据集
        self.dataset = [] #数据集dadaset
        with open(file_name, newline='') as csvfile:
            data = csv.reader(csvfile)
            for line in data:
                row = [x.strip() for x in line]
                if row != [] or row != [""]:
                    self.dataset.append(row)
        # print(self.dataset)

    def calcShannonEnt(self,dataset):
        """计算当前样本集合的信息熵"""
        numSam = len(dataset)  # 统计样本的数量
        labelCounts = {}  # 创建一个字典存储样本的的分类机器对应的数量
        for sample in dataset:
            currentLabel = sample[-1]  # 数据集中通常最后一位为标签属性
            if currentLabel not in labelCounts.keys():  # 将不存在的类别添字典置为0
                labelCounts[currentLabel] = 0
            labelCounts[currentLabel] += 1 #统计在当前样本集中各个类的种数
        shannonEnt = 0.0
        for key in labelCounts:
            prob = float(labelCounts[key]) / numSam #统计该类所占百分比
            shannonEnt -= prob * log(prob, 2)
        return shannonEnt
```

```python
    def splitDataSet(self,dataset,axis,value):
        """提取样本集某属性为特定值的样本"""
        restDataSet = []
        for sample in dataset:
            sam1 = []
            if sample[axis] == value:
                sam1 = sample[:]
                sam1.pop(axis)
                restDataSet.append(sam1)
        return restDataSet #返回数据集某个属性为特定值value的数据集

    def chooseBestAttr(self,dataset):
        """选择最好的数据集划分属性,并返回该属性的下标"""
        baseEnt = self.calcShannonEnt(dataset) #样本划分前的信息熵
        bestGain = 0.0
        bestAttrIdx = 0  #最佳划分属性默认为第一个属性
        for i in range(len(dataset[0])-1): #对样本中的非标签属性一一进行判别,取最大信息增益的属性
            attrsList = [sample[i] for sample in dataset]
            uniqueAttrsL = set(attrsList) #得到属性的所有属性值域
            newEnt = 0.0
            splitInf = 0.0
            for value in uniqueAttrsL:
                subDataSet = self.splitDataSet(dataset,i,value) #将样本集按照属性下不同的取值进行圈划
                prob = len(subDataSet)/(float)(len(dataset)) #对统计新划得的集合的信息（信息熵
                newEnt += prob*self.calcShannonEnt(subDataSet) #按照所占权重统计划分后的信息熵）
                splitInf -= prob*log(prob,2) #属性value的固有值
            infoGain = (baseEnt - newEnt) / splitInf #求得当下划分属性的增益熵/固有值
            if(infoGain > bestGain): #选出增益率最大的划分属性
                bestGain = infoGain
                bestAttrIdx = i #并标记该属性对应样本的下标索引
        return bestAttrIdx
```

```
    def majorLabel(self,classList):
        """函数用于统计某个样本集下类别数最多的类别"""
        statistical = Counter(classList) #使用Counter类来统计比例最高的类
        majorLabel = statistical.most_common(1)
        return majorLabel[0][0]

    def creatTree(self,dataset,labels):
        """创建决策树"""
        attrs = labels[:]
        classList = [sample[-1] for sample in dataset] #提取样本集的分类信息（标签）
        if classList.count(classList[0]) == len(classList):
            return classList[0] #当所有样本标签相同时结束迭代，返回标签
        if (len(dataset[0]) == 1): #若划分的节点的属性为零即数据集只有标签值
            return self.majorLabel(classList)  #返回当前节点样本集中占据多数的标签（类）

        bestAttrIdx = self.chooseBestAttr(dataset) #选出当前节点的最佳划分属性并返回下标
        bestAttrLabel = labels[bestAttrIdx] #由labels列表提取对应划分属性的名称（储存为节点数据）
        myTree = {bestAttrLabel:{}} #以最佳分类属性为节点建立一个空树{}为字典{}内储存节点信息（节点的测试属性和节点下的样本集）
        del (attrs[bestAttrIdx]) #从labels列表中删除已经被选出来当节点的属性
        attrValues = [sample[bestAttrIdx] for sample in dataset]
        uniqueVals = set(attrValues) #找出该属性所有不重复的属性值
        myTree[bestAttrLabel]['majorLabel'] = self.majorLabel(classList) #键majorLabel表示该节点下占最多的类
        for value in uniqueVals:
            subLabels = labels[:] #切边
            myTree[bestAttrLabel][value] = self.creatTree\
                    (self.splitDataSet(dataset,bestAttrIdx,value),subLabels)
        self.myTree = myTree
        return myTree
```

```
    def classify(self,inputTree,attrs,testSample):
        """根据决策树对传递过来的样本进行决策"""
        test = False #test用作是否能在决策树下找到所有的属性值
        _a = list(inputTree.keys())
        firstAttr = _a[0] #得到决策树第一个节点的测试属性

        splitSet = inputTree[firstAttr] #该测试属性下的字典样本集
        AttrIdx = attrs.index(firstAttr)
        for key in splitSet.keys(): #key 为firstAttr属性下的一个唯一取值
            if str(testSample[AttrIdx]) == key: #若样本firAttr属性值存在于决策树下
                test = True
                if type(splitSet[key]).__name__ == 'dict': #为字典表明在该属性下还有其他属性的子树划分
                    samLabel = self.classify(splitSet[key], attrs, testSample) #继续迭代
                else:
                    samLabel = splitSet[key] #表明触底为叶节点，直接得到分类名
        if not test:
            samLabel = splitSet['majorLabel'] #若在当前属性节点下无对应的属性值匹配样本，则返回该节点下最多的类名

        return samLabel

    def a_train(self,labels):
        """一次测试"""
        testSet = []  #测试集
        trainSet = [] #训练集
        numTestSet = int(len(self.dataset) / 3) #测试集占样本总数的1/3
        for i in range(numTestSet):
            x = randint(0,len(self.dataset) - 1) #每次从样本集中随机取样
            testSet.append(self.dataset[x])
        for sample in self.dataset: #将不出现在测试集中的所有样本添加至训练集中，即测试集与训练集无交集
            if sample not in testSet:
                trainSet.append(sample)
        mytree = self.creatTree(trainSet,labels)
        count = 0 #统计分类正确率
        for testVc in testSet:
            if self.classify(mytree,self.labels,testVc) == testVc[-1]:
                count += 1
        print("训练集数目",len(trainSet),"测试集数目",numTestSet,"正确率",'{:.2%}'.format(count/numTestSet))
```

```
    def train_for_n_times(self,datasetname,labels,n):
        '''进行n次测试'''
        self.labels = labels[:]
        print("数据集：", datasetname,"样本属性数量：",len(self.labels),"样本总数:",len(self.dataset),"评估方法:","自助法")
        for i in range(n):
            print("测试",i+1,'',end='')
            self.a_train(self.labels)
```

数据集一：Iris

main.py × iris.csv ×

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
```

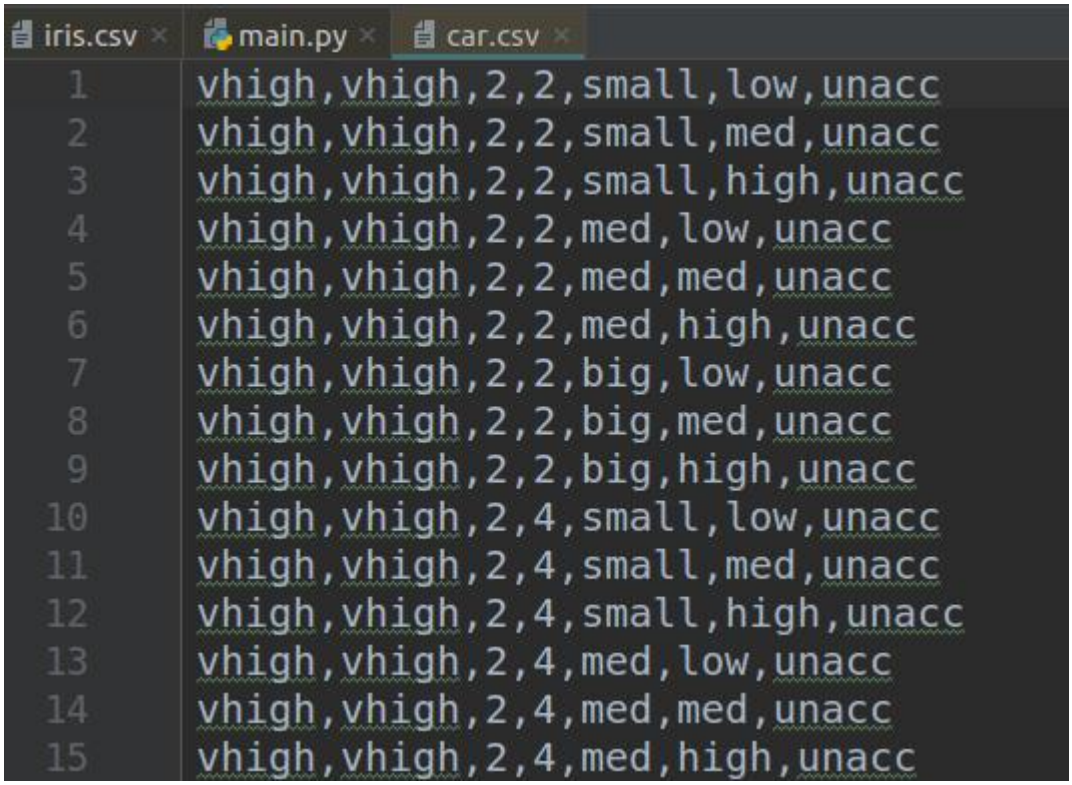
测试代码：

```
143
144 if __name__ == '__main__':
145     cTree = C45("/home/ubuntu/PycharmProjects/C4.5/venv/Data/iris.csv")
146     labels = ['outlook','temperature', 'humidity', 'windy']
147     cTree.train_for_n_times("Iris",labels,n=10)
148
```

测试结果：

```
/home/ubuntu/PycharmProjects/C4.5/venv/bin/python /home/ubun
数据集： Iris 样本属性数量： 4 样本总数: 150 评估方法: 自助法
测试 1 训练集数目 103 测试集数目 50 正确率 82.00%
测试 2 训练集数目 103 测试集数目 50 正确率 92.00%
测试 3 训练集数目 103 测试集数目 50 正确率 96.00%
测试 4 训练集数目 104 测试集数目 50 正确率 88.00%
测试 5 训练集数目 109 测试集数目 50 正确率 94.00%
测试 6 训练集数目 106 测试集数目 50 正确率 96.00%
测试 7 训练集数目 102 测试集数目 50 正确率 92.00%
测试 8 训练集数目 105 测试集数目 50 正确率 96.00%
测试 9 训练集数目 109 测试集数目 50 正确率 100.00%
测试 10 训练集数目 104 测试集数目 50 正确率 94.00%
```

数据集二：Car



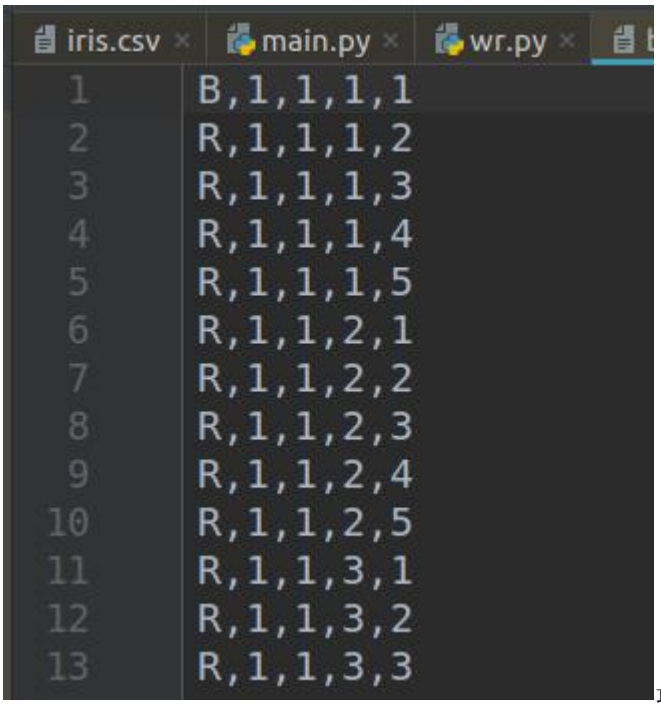
测试代码：

```
44
45  if __name__ == '__main__':
46      cTree = C45("/home/ubuntu/PycharmProjects/C4.5/venv/Data/car.csv")
47      labels = ['buying','maint', 'doors', 'peasons','lug_boot','safety']
48      cTree.train_for_n_times("Car",labels,n=12)
49
```

测试结果：

```
/home/ubuntu/PycharmProjects/C4.5/venv/bin/python /home/ubu
数据集： Car 样本属性数量： 6 样本总数: 1728 评估方法: 自助法
测试 1 训练集数目 1240 测试集数目 576 正确率 91.49%
测试 2 训练集数目 1246 测试集数目 576 正确率 91.15%
测试 3 训练集数目 1225 测试集数目 576 正确率 90.97%
测试 4 训练集数目 1239 测试集数目 576 正确率 91.15%
测试 5 训练集数目 1232 测试集数目 576 正确率 93.40%
测试 6 训练集数目 1233 测试集数目 576 正确率 92.19%
测试 7 训练集数目 1237 测试集数目 576 正确率 91.49%
测试 8 训练集数目 1235 测试集数目 576 正确率 92.71%
测试 9 训练集数目 1242 测试集数目 576 正确率 93.06%
测试 10 训练集数目 1231 测试集数目 576 正确率 92.71%
测试 11 训练集数目 1245 测试集数目 576 正确率 92.36%
测试 12 训练集数目 1228 测试集数目 576 正确率 91.84%
```

数据集三：Balance Scale



最近邻居法

第一列为分类，但在读取 csv 文件时，将第一列与最后一列进行了交换处理。

测试代码：

```
150
151 ▶ if __name__ == '__main__':
152       cTree = C45("/home/ubuntu/PycharmProjects/C4.5/venv/Data/balance-scale.csv")
153       labels = ['Right-Distance','Right-Weight', 'Left-Distance', 'Left-Weight']
154       cTree.train_for_n_times("Balance Scale",labels,n=10)
155
```

测试结果：

```
/home/ubuntu/PycharmProjects/C4.5/venv/bin/python
数据集： Balance Scale 样本属性数量： 4 样本总数: 625
测试 1 训练集数目 441 测试集数目 208 正确率 67.31%
测试 2 训练集数目 446 测试集数目 208 正确率 68.75%
测试 3 训练集数目 447 测试集数目 208 正确率 65.38%
测试 4 训练集数目 451 测试集数目 208 正确率 72.12%
测试 5 训练集数目 444 测试集数目 208 正确率 71.63%
测试 6 训练集数目 448 测试集数目 208 正确率 67.79%
测试 7 训练集数目 452 测试集数目 208 正确率 70.19%
测试 8 训练集数目 443 测试集数目 208 正确率 70.67%
测试 9 训练集数目 441 测试集数目 208 正确率 64.90%
测试 10 训练集数目 445 测试集数目 208 正确率 73.08%
```

2.2.2 CART 算法

CART 决策树通过基尼指数来选择划分属性。数据集的基尼指数越低，集合纯度越高。因此在候选属性中，选择划分后基尼指数最小的属性作为最优划分属性。

代码：

```python
from  math import *
import csv
from collections import Counter
from random import *
class CART:
    def __init__(self,filename):
        self.dataset = [] #数据集
        self.file_reader(file_name=filename)
        self.myTree = None

    def file_reader(self,file_name):#读取文件提取数据集
        self.dataset = [] #数据集dadaset
        with open(file_name, newline='') as csvfile:
            data = csv.reader(csvfile)
            for line in data:
                row = [x.strip() for x in line]
                # a = row[0]
                # row[0] = row[-1]
                # row[-1] = a
                if row != [] and row != [""] and '?' not in row:
                    self.dataset.append(row)

    def calGini(self,dataset):
        """计算当前样本集合的基尼值"""
        numSam = len(dataset)  # 统计样本的数量
        labelCounts = {}  # 创建一个字典存储样本的的分类机器对应的数量
        for sample in dataset:
            currentLabel = sample[-1]  # 数据集中通常最后一位为标签属性
            if currentLabel not in labelCounts.keys():  # 将不存在的类别添字典置为0
                labelCounts[currentLabel] = 0
            labelCounts[currentLabel] += 1  # 统计在当前样本集中各个类的种数
            Gini = 1.0
        for key in labelCounts:
            prob = float(labelCounts[key]) / numSam #统计该类所占百分比
            Gini -= prob ** 2
        return Gini #返回该集合的基尼值

```

```python
    def chooseBestAttr(self,dataset):
        """选择最好的数据集划分属性,并返回该属性的下标"""
        baseGini = self.calGini(dataset)  # 样本划分前的基尼值
        Ginis = [] #Ginis包含按不同属性分类得到的基尼值
        for i in range(len(dataset[0]) - 1):  # 对样本中的非标签属性一一进行判别,取最小基尼数的属性
            attrsList = [sample[i] for sample in dataset]
            uniqueAttrsL = set(attrsList)  # 得到属性的所有属性值域
            newGini = 0.0

            for value in uniqueAttrsL:
                subDataSet = self.splitDataSet(dataset, i, value)  # 将样本集按照属性下不同的取值进行圈划
                prob = len(subDataSet) / (float)(len(dataset))  # 对统计新划得的集合的信息
                newGini += prob * self.calGini(subDataSet)  # 迭代得到该属性下的基尼值
            Ginis.append(newGini)
        return Ginis.index(min(Ginis))#返回最小基尼值对应的属性的下标

    def splitDataSet(self,dataset,axis,value):
        """提取样本集某属性为特定值的样本"""
        restDataSet = []
        for sample in dataset:
            sam1 = []
            if sample[axis] == value:
                sam1 = sample[:]
                sam1.pop(axis)
                restDataSet.append(sam1)
        return restDataSet #返回数据集某个属性为特定值value的数据集

    def majorLabel(self,classList):
        """函数用于统计某个样本集下类别数最多的类别"""
        statistical = Counter(classList) #使用Counter类来统计比例最高的类
        majorLabel = statistical.most_common(1)
        return majorLabel[0][0]

```

```
    def creatTree(self,dataset,labels):
        """创建决策树"""
        attrs = labels[:]
        # classList = [sample[-1] for sample in dataset]
        classList = []
        for sample in dataset:
            classList.append(sample[-1])
        if classList.count(classList[0]) == len(classList):
            return classList[0] #当所有样本标签相同时结束迭代，返回标签
        if (len(dataset[0]) == 1): #若划分的节点的属性为零即数据集只有标签值
            return self.majorLabel(classList)  #返回当前节点样本集中占据多数的标签（类）

        bestAttrIdx = self.chooseBestAttr(dataset) #选出当前节点的最佳划分属性并返回下标
        bestAttrLabel = attrs[bestAttrIdx] #由labels列表提取对应划分属性的名称（储存为节点数据）
        myTree = {bestAttrLabel:{}} #以最佳分类属性为节点建立一个空树{}为字典{}内储存节点信息（节点的测试属性和节点下的样本集）
        del (attrs[bestAttrIdx]) #从labels列表中删除已经被选出来当节点的属性
        attrValues = [sample[bestAttrIdx] for sample in dataset]
        uniqueVals = set(attrValues) #找出该属性所有不重复的属性值
        myTree[bestAttrLabel]['majorLabel'] = self.majorLabel(classList) #键majorLabel表示该节点下占最多的类
        for value in uniqueVals:
            subLabels = attrs[:] #切边
            myTree[bestAttrLabel][value] = self.creatTree\
                    (self.splitDataSet(dataset,bestAttrIdx,value),subLabels)
        self.myTree = myTree
        return myTree
```

```
    def classify(self,inputTree,attrs,testSample):
        """根据决策树对传递过来的样本进行决策"""
        test = False #test用作是否能在决策树下找到所有的属性值
        _a = list(inputTree.keys())
        firstAttr = _a[0] #得到决策树第一个节点的测试属性

        splitSet = inputTree[firstAttr] #该测试属性下的字典样本集
        AttrIdx = attrs.index(firstAttr)
        for key in splitSet.keys(): #key 为firstAttr属性下的一个唯一取值
            if str(testSample[AttrIdx]) == key: #若样本firAttr属性值存在于决策树下
                test = True
                if type(splitSet[key]).__name__ == 'dict': #为字典表明在该属性下还有其他属性的子树划分
                    samLabel = self.classify(splitSet[key], attrs, testSample) #继续迭代
                else:
                    samLabel = splitSet[key] #表明触底为叶节点，直接得到分类名
        if not test:
            samLabel = splitSet['majorLabel'] #若在当前属性节点下无对应的属性值匹配样本，则返回该节点下最多的类名
        return samLabel

    def a_train(self,labels):
        """一次测试"""
        testSet = []  #测试集
        trainSet = [] #训练集
        numTestSet = int(len(self.dataset) / 3) #测试集占样本总数的1/3
        for i in range(numTestSet):
            x = randint(0,len(self.dataset) - 1) #每次从样本集中随机取样
            testSet.append(self.dataset[x])
        for sample in self.dataset: #将不出现在测试集中的所有样本添加至训练集中，即测试集与训练集无交集
            if sample not in testSet:
                trainSet.append(sample)
        mytree = self.creatTree(trainSet,labels)
        count = 0 #统计分类正确率
        for testVc in testSet:
            if self.classify(mytree,self.labels,testVc) == testVc[-1]:
                count += 1
        print("训练集数目",len(trainSet),"测试集数目",len(testSet),"正确率",'{:.2%}'.format(count/numTestSet))
```

```
    def train_for_n_times(self,datasetname,labels,n):
        '''进行n次测试'''
        self.labels = labels[:]
        print("数据集：", datasetname,"样本属性数量：",len(self.labels),"样本总数:",len(self.dataset),"评估方法:","自助法")
        for i in range(n):
            self.labels = labels[:]
            print("测试",i+1,'',end='')
            self.a_train(self.labels)
```

PEP 8: missing whitespace after ','

数据集一：Iris

main.py × iris.csv ×

```
1   5.1,3.5,1.4,0.2,Iris-setosa
2   4.9,3.0,1.4,0.2,Iris-setosa
3   4.7,3.2,1.3,0.2,Iris-setosa
4   4.6,3.1,1.5,0.2,Iris-setosa
5   5.0,3.6,1.4,0.2,Iris-setosa
6   5.4,3.9,1.7,0.4,Iris-setosa
7   4.6,3.4,1.4,0.3,Iris-setosa
8   5.0,3.4,1.5,0.2,Iris-setosa
9   4.4,2.9,1.4,0.2,Iris-setosa
10  4.9,3.1,1.5,0.1,Iris-setosa
11  5.4,3.7,1.5,0.2,Iris-setosa
12  4.8,3.4,1.6,0.2,Iris-setosa
```

测试代码：

```
if __name__ == '__main__':
    cTree = CART("/home/ubuntu/PycharmProjects/C4.5/venv/Data/iris.csv")
    labels = ['outlook', 'temperature', 'humidity', 'windy']
    cTree.train_for_n_times("Iris", labels, n=15)
```

测试结果：

```
数据集： Iris 样本属性数量： 4 样本标签种类： 3 样本总数: 149 评估方法: 自助法
测试 1 训练集数目 102 测试集数目 49 正确率 87.76%
测试 2 训练集数目 108 测试集数目 49 正确率 87.76%
测试 3 训练集数目 106 测试集数目 49 正确率 85.71%
测试 4 训练集数目 105 测试集数目 49 正确率 89.80%
测试 5 训练集数目 105 测试集数目 49 正确率 79.59%
测试 6 训练集数目 107 测试集数目 49 正确率 83.67%
测试 7 训练集数目 103 测试集数目 49 正确率 89.80%
测试 8 训练集数目 109 测试集数目 49 正确率 85.71%
测试 9 训练集数目 105 测试集数目 49 正确率 85.71%
测试 10 训练集数目 104 测试集数目 49 正确率 93.88%
测试 11 训练集数目 105 测试集数目 49 正确率 85.71%
测试 12 训练集数目 108 测试集数目 49 正确率 87.76%
```

数据集二：Car

iris.csv × main.py × car.csv ×

```
vhigh,vhigh,2,2,small,low,unacc
vhigh,vhigh,2,2,small,med,unacc
vhigh,vhigh,2,2,small,high,unacc
vhigh,vhigh,2,2,med,low,unacc
vhigh,vhigh,2,2,med,med,unacc
vhigh,vhigh,2,2,med,high,unacc
vhigh,vhigh,2,2,big,low,unacc
vhigh,vhigh,2,2,big,med,unacc
vhigh,vhigh,2,2,big,high,unacc
vhigh,vhigh,2,4,small,low,unacc
vhigh,vhigh,2,4,small,med,unacc
vhigh,vhigh,2,4,small,high,unacc
vhigh,vhigh,2,4,med,low,unacc
vhigh,vhigh,2,4,med,med,unacc
vhigh,vhigh,2,4,med,high,unacc
```

测试代码：

```
if __name__ == '__main__':
    cTree = CART("/home/ubuntu/PycharmProjects/C4.5/venv/Data/iris.csv")
    labels = ['outlook', 'temperature', 'humidity', 'windy']
    cTree.train_for_n_times("Cars", labels, n=10)
```

测试结果：

```
/home/ubuntu/PycharmProjects/C4.5/venv/bin/python /home/ubuntu/PycharmProjects/C4.5/venv/bin/CART.py
数据集： Car 样本属性数量： 6 样本标签种类： 4 样本总数: 1727 评估方法: 自助法
测试 1 训练集数目 1225 测试集数目 575 正确率 89.57%
测试 2 训练集数目 1242 测试集数目 575 正确率 90.61%
测试 3 训练集数目 1239 测试集数目 575 正确率 92.00%
测试 4 训练集数目 1247 测试集数目 575 正确率 91.65%
测试 5 训练集数目 1244 测试集数目 575 正确率 94.26%
测试 6 训练集数目 1232 测试集数目 575 正确率 93.04%
测试 7 训练集数目 1245 测试集数目 575 正确率 90.26%
测试 8 训练集数目 1246 测试集数目 575 正确率 89.39%
测试 9 训练集数目 1242 测试集数目 575 正确率 91.83%
测试 10 训练集数目 1244 测试集数目 575 正确率 93.39%
测试 11 训练集数目 1234 测试集数目 575 正确率 91.30%
测试 12 训练集数目 1231 测试集数目 575 正确率 91.13%
```

数据集三：absenteeism_at_work

```
ID,Reason for absence,Month of absence,Day of the week,Seasons,Transportation expense,Dis
11,26,7,3,1,289,36,13,33,239.554,97,0,1,2,1,0,1,90,172,30,4
36,0,7,3,1,118,13,18,50,239.554,97,1,1,1,1,0,0,98,178,31,0
3,23,7,4,1,179,51,18,38,239.554,97,0,1,0,1,0,0,89,170,31,2
7,7,7,5,1,279,5,14,39,239.554,97,0,1,2,1,1,0,68,168,24,4
11,23,7,5,1,289,36,13,33,239.554,97,0,1,2,1,0,1,90,172,30,2
3,23,7,6,1,179,51,18,38,239.554,97,0,1,0,1,0,0,89,170,31,2
10,22,7,6,1,361,52,3,28,239.554,97,0,1,1,1,0,4,80,172,27,8
20,23,7,6,1,260,50,11,36,239.554,97,0,1,4,1,0,0,65,168,23,4
14,19,7,2,1,155,12,14,34,239.554,97,0,1,2,1,0,0,95,196,25,40
1,22,7,2,1,235,11,14,37,239.554,97,0,3,1,0,0,1,88,172,29,8
20,1,7,2,1,260,50,11,36,239.554,97,0,1,4,1,0,0,65,168,23,8
20,1,7,3,1,260,50,11,36,239.554,97,0,1,4,1,0,0,65,168,23,8
20,11,7,4,1,260,50,11,36,239.554,97,0,1,4,1,0,0,65,168,23,8
3,11,7,4,1,179,51,18,38,239.554,97,0,1,0,1,0,0,89,170,31,1
```

（数据提取时过滤了第一行的属性名）
测试代码：

```
147 ▶ if __name__ == '__main__':
148       cTree = CART("/home/ubuntu/PycharmProjects/C4.5/venv/Data/absenteeism_at_work.csv")
149       # labels = ['outlook', 'temperature', 'humidity', 'windy']
150       cTree.train_for_n_times("absenteeism_at_work.csv", cTree.Line, n=18)
151
```

测试结果：

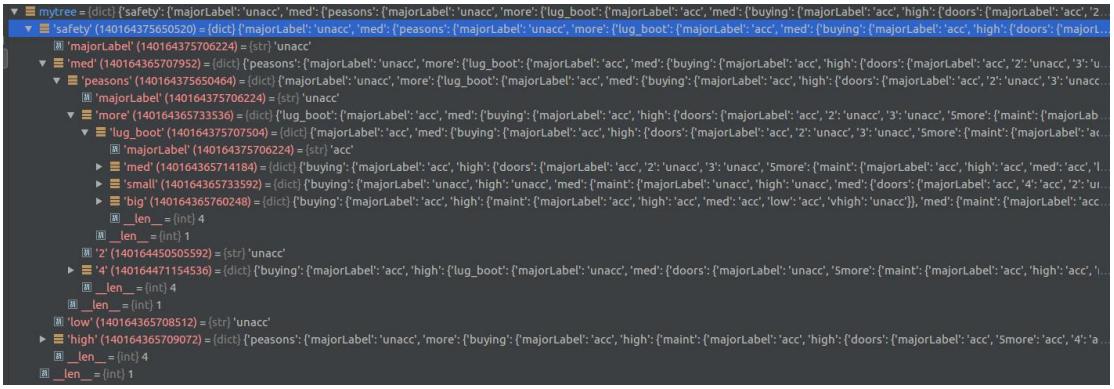
```
数据集： absenteeism_at_work.csv 样本属性数量： 20 样本标签种类： 19 样本总数: 740 评估方法: 自助法
测试 1 训练集数目 508 测试集数目 246 正确率 41.87%
测试 2 训练集数目 513 测试集数目 246 正确率 44.31%
测试 3 训练集数目 518 测试集数目 246 正确率 43.09%
测试 4 训练集数目 521 测试集数目 246 正确率 38.62%
测试 5 训练集数目 508 测试集数目 246 正确率 38.21%
测试 6 训练集数目 523 测试集数目 246 正确率 41.46%
测试 7 训练集数目 521 测试集数目 246 正确率 41.46%
测试 8 训练集数目 513 测试集数目 246 正确率 41.06%
测试 9 训练集数目 516 测试集数目 246 正确率 39.84%
测试 10 训练集数目 508 测试集数目 246 正确率 36.18%
测试 11 训练集数目 505 测试集数目 246 正确率 44.72%
测试 12 训练集数目 524 测试集数目 246 正确率 44.72%
测试 13 训练集数目 508 测试集数目 246 正确率 38.62%
测试 14 训练集数目 515 测试集数目 246 正确率 41.06%
测试 15 训练集数目 516 测试集数目 246 正确率 35.77%
测试 16 训练集数目 507 测试集数目 246 正确率 37.80%
测试 17 训练集数目 521 测试集数目 246 正确率 40.65%
测试 18 训练集数目 521 测试集数目 246 正确率 50.00%
```

对于第三个实验，通过 20 个属性来划分决策得到 CART 决策树，再通过决策树对测试集进行验证，预测旷工时间。实验发现正确率并不高。分析主要有一下原因：1。与之前 Seeds(3)和 Cars(4)数据集相比，Absenteeism_at_work(19)的分类标签过多，达到 19 种之多，显然过多的种类对于决策树的分类正确率影响极大。2.在对分类进行评估预测时采用的时自助法，自助法使得测试集与训练集无交集，一定程度上降低了正确率但也提升了分类模型的泛化性能。此外，由于自助法产生的数据集改变了初始数据集的分布(体现在测试集某些样本可能多次从原始数据集中采样)，这带来了一定的估计偏差。因此在数据集足够大时往往采用交叉验证与留出法来验证模型，而在数据集较小时才使用自助法。

总结：与聚类相似，决策树也是一种分类算法，不同的是决策树算法可以用于，也更适合监督学习中。决策树的原理很简单，他通过某种度量对数据集进行层层分类，这个度量一般指的是集合的纯度，即整个集合中数据之间的相似度，根据纯度不同标准有信息熵和基尼指数等等，再根据根据这些标准可得不同的算法(C4.5,CART 等等),但考虑到决策树分类的目标即是提高集合的纯度，这些算法又其实很相似的，只不过实现过程中分类的判断标准不同罢了。

在整个报告所给的算法中，决策树的算法应该是最高的，在代码中遇到了出现了非常多的问题，我试着梳理一下：决策树的原理如上并不复杂，不管是求数据集的信息熵还是基尼指数很简单，难就难在，在求的某个节点最佳分类属性的预处理中，由于不知道最佳属性，只能暴力迭代，对节点剩余的不同属性的不同取值一一进行分类，在根据不同的算法比较得到最佳分类属性，其中预处理时还不能破坏原有节点集合。此外，对于树的存储形式我一直很纠结，到底时用列表，集合还是字典，亦或是两两嵌套，如何定位树的节点（是否为叶节点，分支又如何展开）由于想不出好的方式这个算法一度搁浅很久。坦诚说，决策树的算法要我独立完成真的无法做到，期间在网上查看了不少代码，都很难啃，难以理解。最后在网上看到了某段代码对决策树的存储的处理，一番冥思苦想后，豁然开朗，叹为观止。他仅仅通过字典的嵌套存储节点信息，每个字典的键对应时该节点的分类属性，字典的值为一个一个字典，这个字典内的键对应的是在这个分类属性下一个唯一取值，这个字典的值又为一个

字典，这个字典指向下一个节点。由此层层嵌套直到最后节点的属性只剩下标签名。这个实现手段可见下图：



Car 数据集生成的一个决策树

通过这种巧妙的存储设计我很快完成了创建生成树的代码，但是又有了新的问题，这种决策手段生成的树只能验证分类已存在的数据，例如对于一个 Iris 样本 [4.3,3.0,1.1,8.1,Iris-setosa]，其不存在于训练集中，先对齐分类，他就会分类到每个节点下由于找不到与之属性对应的属性值集合而无法得到分类结果。这样的话决策树只能完成自内的分类检测，不能对未有的样本进行预测，这种训练也就毫无意义了。为此我在每个节点中添加了一个 majorLabel 属性，他保存了当下分类节点的占多数的标签。当样本在分类中，于某个节点中找不对应的属性集，他就会自动采用这个节点的 majorLabel，将分类结果标记为此，这种方式虽然并不科学，很有可能某个属性节点下的集合们之间差异较大，通过这种粗暴的多数原则对他们一概而论时不合理的，或许对这个样本 majorLabel 结合聚类中的中类似距离的概念综合考虑会更加好，但是最近事情很多，在决策树熵也花费了大量的时间，也就没有实现这种想法，剪枝那块也没有完成，不过，通过实验我这种粗暴的方式正确率也不低，索性便暂时放弃了。总之，完成决策树确实耗费了相当精力，但是完成这个算法，以及在 UCI 数据集决策验证成功也给予了我不少成就感。

近邻学习

2.2.3 KNN 算法

KNN(K 近邻学习)是一种常用的监督学习算法。对于给定的测试样本，其根据某种距离度量在训练集中找出最靠近的 K 个训练样本，基于这 K 个样本的信息进行预测。在分类任务中，通常使用投票法，即选择这 K 个样本中占据多数的类别标签作为预测结果，当然在这 K 个样本的距离里上加权投票，距离越近的权重越大。在回归类任务中通常采用取平均值作为预测结果，同样，这个平均值也可以根据 K 个样本进行加权得到加权平均值。

代码：

```python
from collections import Counter
import csv
import heapq
import random
import math
class KNN:
    def __init__(self, file_name, k):
        self.k = k  #
        self.file_reader(file_name)

    def file_reader(self, file_name):  # 读取文件提取数据集
        """读取文件信息同时通过留出法划分好训练集和测试集"""
        self.Dataset = []  # 原始数据集
        with open(file_name, newline='') as csvfile:
            # _ = csvfile.readline()
            data = csv.reader(csvfile)
            for row in data:
                for i in range(len(row)):
                    try:
                        row[i] = float(row[i])
                    except ValueError:
                        row[i] = str(row[i])
                    else:
                        pass
                self.Dataset.append(row)
        self.testSet = random.sample(self.Dataset,10) #sample函数从原始数据集中随机取得10个样本作为测试集
        self.D = []
        for sample in self.Dataset: #将原始数据集中的其他示例添加到训练集中
            if sample not in self.testSet:
                self.D.append(sample)
```

```python

    def dist(self, a, b):
        """计算两个样本a,b之间的欧式距离"""
        i = s = 0
        while i < len(a):
            if type(a[i]) is float or type(a[i]) is int:
                s += (a[i] - b[i]) ** 2
                i += 1
            else:
                break
        return math.sqrt(s)

    def majorLabel(self, classList):
        """函数用于统计某个样本集下类别数最多的类别"""
        statistical = Counter(classList)  # 使用Counter类来统计比例最高的类
        majorLabel = statistical.most_common(1)
        return majorLabel[0][0]

    def classifyLabel(self,sam,regression=False):
        '''通过regreession来决定执行什么任务（回归或分类）'''
        distances = []
        for sample in self.D:
            distances.append(self.dist(sam,sample))
        L = heapq.nsmallest(self.k,distances) #heapq堆模块可方便寻得最小的k个距离
        classLabels = [self.D[i][-1] for i in range(len(self.D)) if distances[i] in L ]
        if regression: #回归任务
            avg_label = 0
            for x in set(sorted(classLabels,reverse=True)):
                avg_label += (classLabels.count(x)) / (float)(len(classLabels)) * x #求得加权平均数
                return avg_label
        else:
            return self.majorLabel(classLabels) #返回标签类别列表中的多数标签

```

```python
    def test(self,file_name):
        regression = False #通过示例样本的最后一位属性值的属性确定KNN任务
        print("数据集",file_name,"评估方法：留出法")
        if type(self.testSet[0][-1]) is str:#若为字符，即为回归任务
            regression = False
            print("分类任务:")
        elif type(self.testSet[0][-1]) is float or type(self.testSet[0][-1]) is int:#若为实数型，即为回归任务
            print("回归任务:")
            regression = True
#       求得所有测试集的结果列表
        results = [self.classifyLabel(self.testSet[i],regression) for i in range(len(self.testSet))]
        if regression:
            for i in range(len(self.testSet)):
                print("测试",i+1,self.testSet[i],"测试结果:",results[i], "相对误差:",
                      (self.testSet[i][-1] - results[i]) / self.testSet[i][-1])
        else:
            for i in range(len(self.testSet)):
                print("测试",i+1,self.testSet[i],"测试结果:",results[i],
                      "判断结果:",self.testSet[i][-1] == results[i])

```

数据集一：Iris

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
```

测试代码：

```
if __name__ == "__main__":
    exp = KNN('/home/ubuntu/PycharmProjects/KNN/venv/data/iris.csv',k = 5)
    exp.test("Iris")

```

测试结果:

```
数据集 Iris 评估方法: 留出法
分类任务:
测试 1 [6.1, 2.8, 4.7, 1.2, 'Iris-versicolor'] 测试结果: Iris-versicolor 判断结果: True
测试 2 [5.1, 2.5, 3.0, 1.1, 'Iris-versicolor'] 测试结果: Iris-versicolor 判断结果: True
测试 3 [5.7, 3.0, 4.2, 1.2, 'Iris-versicolor'] 测试结果: Iris-versicolor 判断结果: True
测试 4 [5.1, 3.3, 1.7, 0.5, 'Iris-setosa'] 测试结果: Iris-setosa 判断结果: True
测试 5 [6.6, 3.0, 4.4, 1.4, 'Iris-versicolor'] 测试结果: Iris-versicolor 判断结果: True
测试 6 [7.7, 3.8, 6.7, 2.2, 'Iris-virginica'] 测试结果: Iris-virginica 判断结果: True
测试 7 [4.6, 3.4, 1.4, 0.3, 'Iris-setosa'] 测试结果: Iris-setosa 判断结果: True
测试 8 [6.7, 3.1, 4.4, 1.4, 'Iris-versicolor'] 测试结果: Iris-versicolor 判断结果: True
测试 9 [6.7, 3.3, 5.7, 2.1, 'Iris-virginica'] 测试结果: Iris-virginica 判断结果: True
测试 10 [6.2, 2.9, 4.3, 1.3, 'Iris-versicolor'] 测试结果: Iris-versicolor 判断结果: True
```

数据集二：absenteeism_at_work

```
ID,Reason for absence,Month of absence,Day of the week,Seasons,Transportation expense,Distance from Residence to Work,Ser
11,26,7,3,1,289,36,13,33,239.554,97,0,1,2,1,0,1,90,172,30,4
36,0,7,3,1,118,13,18,50,239.554,97,1,1,1,1,0,0,98,178,31,0
3,23,7,4,1,179,51,18,38,239.554,97,0,1,0,1,0,0,89,170,31,2
7,7,7,5,1,279,5,14,39,239.554,97,0,1,2,1,1,0,68,168,24,4
11,23,7,5,1,289,36,13,33,239.554,97,0,1,2,1,0,1,90,172,30,2
3,23,7,6,1,179,51,18,38,239.554,97,0,1,0,1,0,0,89,170,31,2
10,22,7,6,1,361,52,3,28,239.554,97,0,1,1,1,0,4,80,172,27,8
20,23,7,6,1,260,50,11,36,239.554,97,0,1,4,1,0,0,65,168,23,4
14,19,7,2,1,155,12,14,34,239.554,97,0,1,2,1,0,0,95,196,25,40
```

测试代码：

```
if __name__ == "__main__":
    exp = KNN("/home/ubuntu/PycharmProjects/KNN/venv/data/absenteeism_at_work.csv",k = 5)
    exp.test("absenteeism_at_work")
```

测试结果：

```
数据集 absenteeism_at_work 评估方法: 留出法
回归任务:
测试 1 1.0 测试结果: 0.42857142857142855 相对误差: 0.5714285714285714
测试 2 8.0 测试结果: 3.2 相对误差: 0.6
测试 3 3.0 测试结果: 1.7999999999999998 相对误差: 0.4000000000000001
测试 4 8.0 测试结果: 8.0 相对误差: 0.0
测试 5 8.0 测试结果: 4.0 相对误差: 0.5
测试 6 2.0 测试结果: 3.2 相对误差: -0.6000000000000001
测试 7 8.0 测试结果: 3.2 相对误差: 0.6
测试 8 8.0 测试结果: 3.2 相对误差: 0.6
测试 9 3.0 测试结果: 1.4285714285714286 相对误差: 0.5238095238095238
测试 10 2.0 测试结果: 1.6 相对误差: 0.19999999999999996
```

数据集三：Seeds

```
15.26,14.84,0.871,5.763,3.312,2.221,5.22,1
14.88,14.57,0.8811,5.554,3.333,1.018,4.956,1
14.29,14.09,0.905,5.291,3.337,2.699,4.825,1
13.84,13.94,0.8955,5.324,3.379,2.259,4.805,1
16.14,14.99,0.9034,5.658,3.562,1.355,5.175,1
14.38,14.21,0.8951,5.386,3.312,2.462,4.956,1
14.69,14.49,0.8799,5.563,3.259,3.586,5.219,1
14.11,14.1,0.8911,5.42,3.302,2.7,5,1
16.63,15.46,0.8747,6.053,3.465,2.04,5.877,1
16.44,15.25,0.888,5.884,3.505,1.969,5.533,1
```

虽然 seeds 的类别属性为实型，但是实际考虑(标签数字表示种子种类)还是做了分类任务。

测试代码：

```
if __name__ == "__main__":
    exp = KNN("/home/ubuntu/PycharmProjects/KNN/venv/data/seeds.csv",k = 5)
    exp.test("seeds")
```

测试结果：

```
数据集 seeds 评估方法: 留出法
回归任务:
测试 1 2.0 测试结果: 2.0 判断结果: True
测试 2 3.0 测试结果: 3.0 判断结果: True
测试 3 1.0 测试结果: 1.0 判断结果: True
测试 4 3.0 测试结果: 3.0 判断结果: True
测试 5 1.0 测试结果: 1.0 判断结果: True
测试 6 1.0 测试结果: 1.0 判断结果: True
测试 7 3.0 测试结果: 3.0 判断结果: True
测试 8 2.0 测试结果: 2.0 判断结果: True
测试 9 2.0 测试结果: 2.0 判断结果: True
测试 10 1.0 测试结果: 1.0 判断结果: True
```

总结：KNN 算法虽未监督算法，但实际上他并没有任何显式地训练过程，是懒惰学习的一个典型代表。他仅仅将训练集的样本保存好，训练开销也就为零，在测试时，通过比较取得与测试样本最为接近的 K 个训练样本，通过分析这 K 个样本和测试任务，最终得到结果。KNN 算法可执行回归任务，可以执行任务。然而在前面的试验中，发现分类任务的表现非常好，正确率很高，但是回归任务型的测试结果并不理想，相对误差比较大，方差也很高。分析了一下，可能与我的算法有关。考虑到这种情况下，在寻得 K 个最小距离时，发现有几项并列排在后面，我的做法是将并列的这几项也添加进距离样本中，也就是说实际 K 值时大于预定值的。这种做法在分类时问题不大，但是在回归任务中可能引起较大的波动，尤其是在样本比较小的时候，要解决这个并列的问题可能采取其他的度量单位，比如说采取欧式距离外的一个算法再进行筛选，然而这种并列的问题是无法回避的，我也没能查到有效的办法。再一个引起回归误差的是在求得加权平均数中，我采用了最直接的办法对类别按照比例权重求得平均数，并未考虑到实际距离的远近进行合适的加权(尺寸不好把握)，这种处理方式直接将入选 K 个最小距离的样本一视同仁，这样忽视了这些样本实际上的对测试样本不同的影响，对于那些距离更为亲近的，欧式距离小的样本应该给予更高的权重。另外我这个算法，缺陷比较严重，就是只能处理实数型的属性，对于一些属性为字符的，无法使用欧式距离叠加，直接跳过忽略了，查询了下资料，似乎对于这种离散无序的属性有汉明距离这种度量，但是并不适合于实际的数据集。

2.2.4 朴素贝叶斯

朴素贝叶斯法是基于贝叶斯定理的分类方法。其特殊之处在于对属性条件独立独立性的假设，即朴素贝叶斯假设所有属性之间的分布相互独立，这是种理想化的处理方式。贝叶斯分类器通过求得样本分类选择的最大后验概率 P(c|x)，由此得到样本 x 的最小条件风险。而后验概率 P(c|x)贝叶斯同理 P(c|x)=P(x,c)/P(x)=P(c)P(x|c)/P(x),P(c)即为类先验概率(样本空间中某类的所占比例，在数据足够大时，可用频率表示概率),P(x)为证据因子(其大小与类的标记无关，即使所有的类标记相同)，因此求解的关键便在于对最大 P(x|c)的求解。一般通过极大似然估计求解。在朴素贝叶斯中，由于假设各个属性的分布是独立的，由此 P(x|c)可转化为各个属性 xi 在类别为 c 的数据集上的分布频率的累积，这便是朴素贝叶斯的核心所在。最后综合各个不同的类 ci，计算每一个标记的后验概率 P(ci|x)，选择最大的 P(ci|x)对应的类标签 ci 作为预测值。

总结：朴素贝叶斯分类器的训练过程就是基于训练集 D 来估计类先验概率 P(c)，并为每个属性估计条件概率 P(xi|c)。处理过程中中可能出现测试样本的某个属性值不存在于训练集对应的属性中，使 P(ci|x)=0，导致 P(x|c)结果也为 0。为了避免因为某个属性的原因造成其他有效属性被忽略，通常可使用“拉普拉斯修正”。虽然会造成一定的数据偏差，但是在训练集足够大时，实际影响也在减少。

2.3 统计学习

2.3.1 支持向量机 SVM

支持向量机(support vector macher)，简称 SVM，是回归与分类问题中分析数据的监督学习算法。SVM 算法处理一个类别为二的数据集，他将实例表示为空间中的点，通过线性或非线性的方式将两个不同类别的实例尽可能的间隔开来，由此建立了一个 SVM 模型。在这个模型的基础上，对新的测试实例一一映射到同一空间，根据 SVM 模型落在间隔的哪一侧来预测所属的类别。SVM 的关键在于找出这个划分的直线，平面或超平面。一个标准时我们希望分类界限距离最近的训练数据点越远越好，因为这样可以缩小分类器的泛化误差，

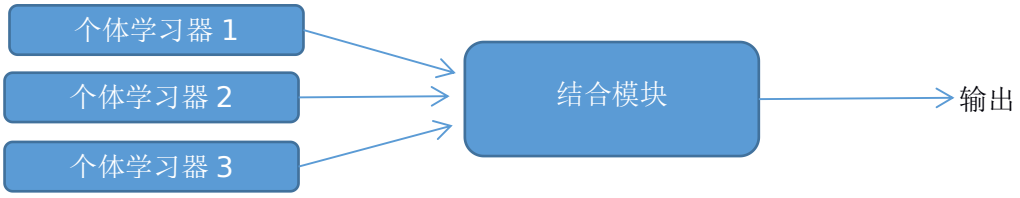
提高泛化能力。通常我们将划分的超平面表示为 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b=0$，其中 $\boldsymbol{w}=(w_1;w_2;\ldots;w_d)$ 表示超平面的法向量，b 表示位移项，即超平面与原点之间的距离。对于一个训练集，

$$D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_m,y_m)\},\ y_i\in\{-1,+1\},$$

假设超平面将训练集正确划分，若对训练集中的任意样本都 yi = +1, wt x + b > 0, yi = -1, 则 wt x + b < 0。我们将满足上式的离超平面最近的几个数据点称为支持向量。两个异类的支持向量的和为 2 / ||w|| ，即“间隔”。当间隔取最大值时，即 min ||w||²/2，划分超平面也就最佳。这便是支持向量机的核心所在。欲求 min ||w||²/2， 通常对齐使用拉格朗日乘子法得其对偶问题解决。对于非线性可分的样本，可能不存在一个划分超平面能将原始样本空间正确划分为两类。对这样的问题，通常将样本从原始空间映射到一个更高的特征空间，使这个样本在这个特征空间内线性可分。然而，更高维的特征空间增加了支持向量机的泛化误差，但是在样本足够多的情况下，算法仍能表现良好。

2.4 集成学习

集成学习通过构建并结结合多个学习器来完成学习任务，也被称为定义分类学习器系统。集成学习本质上不是一个具体的算法，他是利用多个学习器相结合，由此获得比单一学习器更加优越的泛化能力。



……

集成学习通过某种策略将学习器结合起来，通过多个学习器的共同决策，再采用相应的方法（如投票法）等，得出最终分类结果。若集成的学习器中只含有一种类型的个体学习器，如三个 CART 算法生成的学习器，这样的集成为同质的，也称为”基学习器“。例如上面实验中，Iris 数据集实验出现在了 KNN，CART，C4.5 等多中算法中，我们对于相同数据集生成的三个不同的学习器相互结合得到一个自定义的分类学习器，对于同一个测试样本，一一经过三个学习器的预测给出结果，由三者的投票给出最终预测结果。

2.4.1 AdaBoost 算法

Boosting 是一族将弱学习器提升为强学习器的方法。他的工作机制为：先从初始训练集训练出一个基学习器，再根据基学习器的表现对训练样本的分布进行调整，对于基学习器预测失败的样本予以重点关注，然后在调整的训练样本中再度训练出新的基学习器，如此反复，直到基学习器的生成数目达到预定的数目 T，最终对于生成的 T 个基学习器进行加权结合。

AdaBoosting 算法时 Boosting 算法中比较出色的一种，他通过对这 T 个生成基学习器进行线性组合，来最小化指数损失函数。在 AdaBoost 算法中，除去第一次生成的基学习器时通过初始数据集分布而得，之后的基学习器 Ht 和对应的权重αt 都是基于新的样本分布 Dt 产生的。在实际迭代中，关键之处在于每次迭代中的分类器权重更新公式，以及根据预测结果更新的样本分布公式，两者对之后的迭代起着决定性的作用。如上所言，AdaBooting 算法能够在得到上次的样本分布后进行动态调整，在下一轮生成

基学习器的过程中进行纠正。注意对于每次的样本分布，分类的误差应当小于 0.5(如大于 0.5 会退出循环)，这也就是所谓的残差逼近思想。

2.4.2 随机森林

Bagging 是并行式集成学习的一个代表。他的思想与自助法相关(如上诉 Kmean 算法实验中)，他通过采样 T 个随机训练样本采样集，基于每个采样集训练出一个基学习器，再将这 T 个基学习器结合。在分类问题中通过简单投票法得到最终预测结果，在回归问题中，通过简单平均法得到最终预测结果。若在分类中遇上票数相同的，则随机选择一个，也可以进一步考察学习器的投票置信度来确定最终预测结果。

随机森林时(Random Forest)，简称 RF，Bagging 的一个拓展变体。他是在决策树为基学习器的基础上，在进一步的决策树的训练过程中引入随机属性选择。通过之前的决策树学习可知，决策树在划分时会在当前节点下选最佳的划分属性。但随机森林中，他是在该节点的余下属性集合中随机选择一个包含 k 个属性的子集，再从这个子集中选择一个最佳属性用作划分。这个 k 参数某种程度上控制了随机性，一般情况下采用 k = log(2) d ,d 为余下属性集的属性数量。随机森林相比其他集成学习要简单，也容易实现，计算的开销小，但是性能却十分强大，被称为“代表集成学习技术水平的方法。