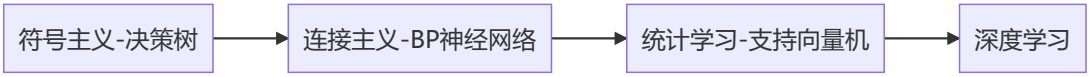
学习笔记—机器学习（西瓜书）

绪论

1. 根据训练数据是否拥有标记信息，学习任务可大致划分为两大类"**监督学习**"和 "**无监督学习**" ，分类和回归是前者的代表，而聚类是后者的分类。
2. **泛化能力**：学得模型适用于 新样本的能力。
3. **"没有免费的午餐"定理** (No Free Lunch Theorem，简称 NFL定理）：无论学习算法是聪明还是笨拙，期望性能相同。
4. **机器学习发展史**：



5. **过拟合**：复杂的模型将抽样误差也进行了很好的拟合。（与之相对的是，欠拟合）
6. **大数据时代关键三大技术**：机器学习、云计算、众包，机器学习提供数据分析能力，云计算提供数据处理能力，众包提供数据标记能力。
7. 验证集：模型评估与选择中用于评估测试的数据集。
8. 番外：

（1）众包：一种分布式的问题解决和生产组织形式，采用某种机制使群体共同参与某件事情，达到某个目标。

（2）NP问题：一个复杂问题不能确定是否在多项式时间内找到答案，但是可以在多项式内验证答案是否正确。

（3）估计误差：估计量的期望值和估计参数的真值之差。误差为零的估计量或决策规划称为无偏的。

模型评估与选择

训练集和测试集的选择

1. 当数量足够，**留出法**简单省时，在牺牲很小的准确度的情况下，换取计算的简便。缺点：会损失一定的样本信息，需要大量样本
2. 当数据量小，选择**交叉验证法**（当数据量特别少时可以考虑留一法）。缺点：计算复杂度过高，空间存储占用过大
3. 当数据集较小、难以有效划分训练/测试集时，选择**自助法**，而且自助法能从初始数据集中产生多个不同的训练集，适合集成学习。缺点：改变了初始数据集的分布，会引入估计偏差。

性能度量

1. 结果性能评估：（部分实现见最后面的ARMA实现）

- MSE——均方误差
- MAE——平均绝对误差
- MAPE——平均误差百分比
- RMSE——均方根误差

2. **查准率P**——准确率

   查全率R——召回率 （矛盾的度量）

3. **P-R曲线**：查准率-查全率曲线

（1）if 学习器A 全包住 学习器B：

学习器A的性能 比 学习器B的性能 好

（2）平衡点BER：查准率 = 查全率

（3）F1度量 :$F1 = \frac{2*P*R}{P+R}$

4. **ROC 曲线**全称是"受试者工作特征" (Receiver Operating Characteristic)：真正准率-假正例率

（1）if 学习器A 全包住 学习器B：

学习器A的性能 比 学习器B的性能 好

（2）AUC（Area Under ROC Curve）：ROC曲线下的面积

5. **代价敏感错误率**：考虑到不同错误的不同代价
6. 常用检验：

- 比较检验
- 假设检验
- 交叉验证t检验
- McNemar检验
- Friedman检验 和 Nemenyi后续检验

偏差与方差

泛化误差 = 偏差 + 方差 + 噪声（学习问题本身的难度）

多项式拟合回归

原理

最小二乘法；泰勒展开的思想

实现

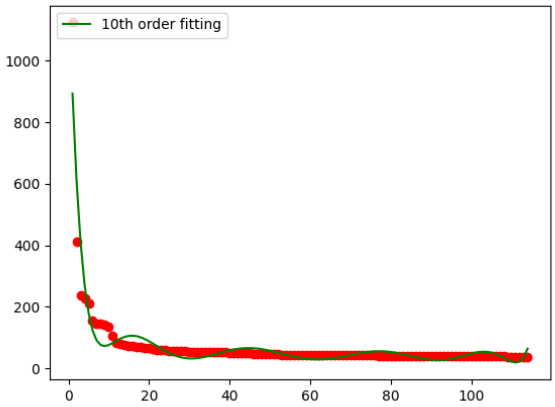
多项式回归（调库版）

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# prepare data
excel = pd.read_excel(r"C:/Users/华为/Desktop/test.xlsx")
x = np.array(excel['a']).astype(np.float)
y = np.array(excel['b']).astype(np.float)
    # print(x)
    # print(y)
x = np.array(x)
y = np.array(y)

coef1 = np.polyfit(x,y, 10)
poly_fit1 = np.poly1d(coef1)
plt.plot(x, poly_fit1(x), 'g',label="10th order fitting")
print(poly_fit1)
plt.scatter(x, y, color='red')
plt.legend(loc=2)
plt.show()
```

数据来源：化学反应的产量

结果：(最高可以拟合到74阶，单纯觉得好玩，但应该用不到这么高，computers are really powerful)

```
           10             9             8             7             6
4.628e-14 x  - 2.801e-11 x + 7.298e-09 x - 1.071e-06 x + 9.714e-05 x
             5          4         3         2
 - 0.005634 x + 0.2088 x - 4.8 x + 64.18 x - 443.1 x + 1277
```



多项式回归（原理版）

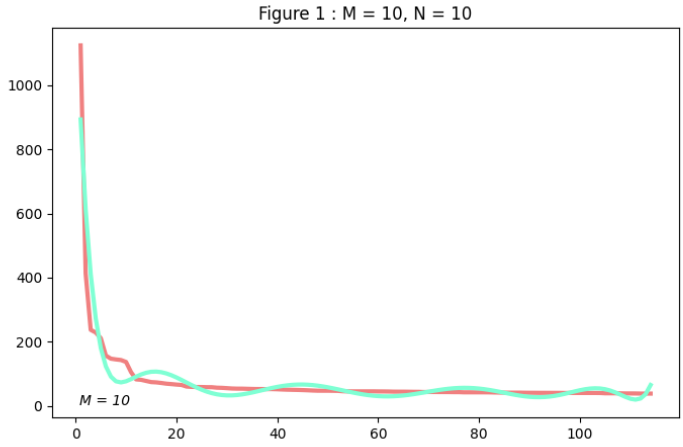
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

def coefficient(M, N, x, y,lamda=0):    # M is the order number
    print("  M=%d  ,  N=%d  "%(M,N))
    order = np.arange(M+1)    #np.arange->list
    order = order[:, np.newaxis] #Increased dimension becomes a column
matrix
    e = np.tile(order, [1,N])  #Copy N times along the y-axis
    XT = np.power(x, e)  #Build a matrix that looks like a Van der
Monde determinant
    X = np.transpose(XT)
    m = np.matmul(XT, X) + lamda*np.identity(M+1)  #XT * X
    n = np.matmul(XT, y) #XT * y
    w = np.linalg.solve(m,n) #mW = n -> W = (XT * X)^-1 * XT * y
    print("W:")   #Coefficient matrix
    print(w)
    return w

def fit(M,w):
    order = np.arange(M+1)    #np.arange->list
    order = order[:, np.newaxis] #Increased dimension becomes a column
matrix
    e2 = np.tile(order, [1,x.shape[0]])
    XT2 = np.power(x, e2)
    p = np.matmul(w, XT2)
    print("p:")
    print(p)
    return p

# prepare data
excel = pd.read_excel(r"C:/Users/华为/Desktop/程序/多项式回归-test.xlsx")
x = np.array(excel['a']).astype(np.float)
y = np.array(excel['b']).astype(np.float)
```

```
x = np.array(x)
y = np.array(y)
N = len(x)   # Number of data

#Get the coefficient matrix
w = coefficient(10, N, x, y)  #10 order fitting
p = fit(10,w)

#drawing picture
plt.figure(1, figsize=(8,5))
plt.plot(x, y, 'lightcoral', x, p, 'aquamarine',linewidth=3)
#plt.scatter(x, y,  marker='o', edgecolors='b', s=100, linewidth=3)
plt.text(0.8, 0.9,'M = 10', style = 'italic')
plt.title('Figure 1 : M = 10, N = 10')
plt.savefig('1.png', dpi=400)
plt.show()
```

结果：（用10阶多项式拟合）

系数矩阵：W = (XT * X)^-1 * XT * y

```
W:（系数矩阵）
[ 1.27779390e+03 -4.43841509e+02  6.43387111e+01 -4.81418651e+00
  2.09539454e-01 -5.65681680e-03  9.75626449e-05 -1.07576974e-06
  7.33472900e-09 -2.81580361e-11  4.65325200e-14]
p:（多项式函数）
[893.68089416 612.12995525 411.0016443  271.97128938 180.00860766
 122.91265949  90.89768277  76.22561303  72.88134119  76.28699867
  83.05178785  90.75409541  97.75283662 103.02518085 106.02800204
 106.58058334 104.76628228 100.85103207  95.21671632  88.30760817
  80.58821147  72.51098086  64.49252957  56.8970592   50.02586402
  44.11187424  39.31830815  35.74060222  33.41088134  32.30431861
  32.34681552  33.42350925  35.3876843   38.0697309   41.28585276
  44.84628219  48.56281081  52.25549043  55.75840006  58.92441231
  61.62892602  63.77256104  65.28283712  66.11488074  66.2512227
  65.70076478  64.49700623  62.69563069  60.37156087  57.61559299
  54.5307247   51.22829044  47.82401565  44.43409757  41.17141421
  38.14195635  35.44156824  33.15307351  31.34385139  30.06391717
  29.34454829  29.19748454  29.61471792  30.56887418  32.0141747
  33.88795433  36.11269796  38.59854592  41.24620724  43.95020841
  46.60239576  49.09560096  51.32737165  53.2036637   54.64238734
  55.57669723  55.95791633  55.75798548  54.97133494  53.61608077
  51.73445909  49.39242304  46.67834329  43.70077132  40.58524688
  37.47015663  34.50168047  31.82789472  29.59213958  27.92579845
  26.94068384  26.72127412  27.31710115  28.73564807  30.93618283
  33.825022    37.25279634  41.01437019  44.85215397  48.46364237
  51.51410915  53.6554955   54.55264025  53.91811826  51.55707884
  47.42360791  41.69027602  34.83268172  27.7309528   21.7903286
  19.08311831  22.51450267  36.01483723  64.7613033 ]
```



收获

1. 学习了利用pyhton建立一个类似范德蒙行列式的矩阵，学习了np库的一些复制增维等函数。
2. 有趣的事情发生了，调库和原理两个版本可拟合的最高阶数居然一样，都是74阶！而且系数矩阵基本类似，有点好奇调库背后的实现，或许是因为原理不那么复杂，所以实现方式比较类似。
3. 用plt调色，colorful，顺便收藏一下**plt的色库**：

4. 多项式拟合预测的缺点：在x->∞时，y->∞，适合给定数据范围内的预测，不过也可以考虑当数据少的时候用多项式函数生成数据，或者用来插值也不错。

番外

1. 线性判别分析（LDA）：给出投影方向，欲使同类样例投影点的协方差尽可能小，即当两类数据同先验、满足高斯分布且协方差相等时，LDA可达到最优分类。
2. 多分类学习（我觉得支持向量机就是最经典的例子）：OvO（N个类别两两配对）、OvR（一个类作为正例，其他类作为反例）、MvM（若干个作为正例，若干个作为反例）

- 对比：OvO的存储开销和测试时间开销比OvR大，但类别多的时候，它的训练时间开销小，至于性能两者差不多，更多取决于数据分布。
- MvM技术——纠错输出码（ECOC）：
  - 编码：对N个类别做M次划分，每次划分将一部分类别划为正类，一部 分划为反类，从而形成一个二分类训练集；这样一共产生M个训练集，可训练出M个分类器。
  - 解码：M 个分类器分别对测试样本进行预测，这些预测标记组成一个编码。将这个预测编码与每个类别各自的编码进行比较，返回其中距离最小的类别作为最终预测结果。

3. 类别不平衡问题——指分类任务中不同类别的训练样例数目差别很大的情况。

- 基本策略：再缩放（代价敏感学习的基础）：欠采样、过采样、基于原始训练集进行学习

决策树

原理

一个指标一个指标的往下分，更像是一种决策的思路，为了判断最优划分，利用信息熵来度量集合纯度，值越小纯度越高。

实现

决策树（调库版）

```
from sklearn import tree
import graphviz
import pandas as pd
import numpy as np

file_path = r"C:/Users/华为/Desktop/程序/svm-data.xlsx"
data = pd.DataFrame(pd.read_excel(file_path))
print(r'''Info:
1. month: 1~12
2. nation: 汉族
3. sex：1(男), 0(女)
4. Ispoorpot：0(Flase), 1(True)
```

```
5. poor_level: 0(非贫困生), 1(突发事件特殊困难), 2(家庭经济困难), 3(家庭经济特别困难)
6. dormi: 0(丁香), 1(海棠), 2(竹园)''')
x, y = np.split(data.values, indices_or_sections=(6,), axis=1)
#绘制树模型
clf = tree.DecisionTreeClassifier()
clf = clf.fit(x, y)
print(clf.predict(x))
tree.export_graphviz(clf)
dot_data = tree.export_graphviz(clf, out_file=None)
graphviz.Source(dot_data)
#利用render方法生成图形
graph = graphviz.Source(dot_data)
graph.render("tree")
```

数据来源：学生分类（数据239955行，效率soso）

数据指标：ID、month、ispoorpot、cost_average、dormitory、poor_level

结果：very very big tree

- [ ] **决策树（原理版）**（代码学习来源于**MLia**，看懂了大部分并尝试自己实现，但create tree还没太理解，**设置为未完成**）

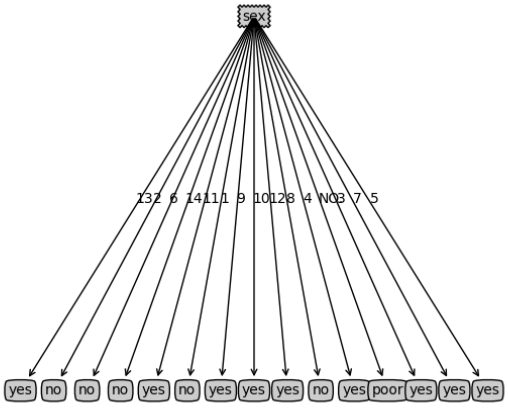
```
#!/usr/bin/env python
from math import log
import operator
import treePlotter
# import imp
# import sys
# imp.reload(sys)
# sys.setdefaultencoding('utf8')

#信息熵#
def Ent(dataset):
    num = len(dataset)
    data_dict = {}
    for i in dataset:
        last = i[-1]  #Take the data in the last column
        if last not in data_dict.keys():data_dict[last] = 0
        data_dict[last] += 1
    Entt = 0.0
    for key in data_dict:
        p_k = float(data_dict[key]) / num
        Entt -= p_k * log(p_k,2)
    return Entt

#划分数据#
```

```
def splitDataSet(dataSet, axis, value): #Axis is the column number of
the dataSet, and value is a characteristic value of the column
    DataSet = []
    for i in dataSet: #The data set is traversed and the data set of
the current value feature is obtained
        if i[axis] == value:
            reducedFeatVec = i[:axis]     #chop out axis used for
splitting
            reducedFeatVec.extend(i[axis+1:])
            DataSet.append(reducedFeatVec)
    return DataSet

#划分数据集的最优特征
def Bestdataset(dataset):
    numfinal = len(dataset[0]) - 1
    data_Ent = Ent(dataset)  #Calculate Ent
    bestInfoGain = 0.0; bestFeature = -1
    for i in range(numfinal):
        list = [example[i] for example in dataset]
        unique = set(list)  #Get eigenvalues
        newEntropy = 0.0
        for value in unique:
            #Calculate the Ent for each partition
            subdataset = splitDataSet(dataset, i, value)
            p_k = len(subdataset) / float(len(dataset))
            newEntropy += p_k * Ent(subdataset)
        gain = data_Ent - newEntropy
        if(gain > bestInfoGain):
            bestInfoGain = gain
            bestFeature = i
    return bestFeature

def majorityCnt(classList):
    #The most frequently occurring category name
    classCount={}
    for vote in classList:
        if vote not in classCount.keys(): classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.iteritems(),
key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]

def createTree(dataset,labels):
    List = [example[-1] for example in dataset]
    if List.count(List[0]) == len(List):
        return List[0]  #Exactly the same category
    if len(dataset[0]) == 1:
        return majorityCnt(List)
        #By default, the category returned the most times
```

```
    bestFeat = Bestdataset(dataset)
    bestFeatLabel = labels[bestFeat] #Get feature names
    # Dictionary variables to store tree information
    myTree = {bestFeatLabel:{}} #Pick the best features
    del(labels[bestFeat]) #Delete features that are already being
selected
    featValues = [example[bestFeat] for example in dataset]
    unique = set(featValues)
    for value in unique:
        subLabels = labels[:]       #copy all of labels, so trees don't
mess up existing labels
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataset,
bestFeat, value),subLabels)
    return myTree

def classify(inputTree,featLabels,testVec):
    firstStr = inputTree.keys()[0]
    secondDict = inputTree[firstStr]
    featIndex = featLabels.index(firstStr)
    key = testVec[featIndex]
    valueOfFeat = secondDict[key]
    if isinstance(valueOfFeat, dict):
        classLabel = classify(valueOfFeat, featLabels, testVec)
    else: classLabel = valueOfFeat
    return classLabel

def storeTree(inputTree,filename):
    import pickle
    fw = open(filename,'w')
    pickle.dump(inputTree,fw)
    fw.close()

def grabTree(filename):
    import pickle
    fr = open(filename)
    return pickle.load(fr)


if __name__ == '__main__':
    fr = open(r'C:/Users/华为/Desktop/tree-data.txt')
    lenses =[inst.strip().split(' ') for inst in fr.readlines()]
    lensesLabels = ['sex','month','cost','poor_pot']
    lensesTree =createTree(lenses,lensesLabels)
    treePlotter.createPlot(lensesTree)
```

数据来源：学生贫困分类（只选取了很少的部分，单纯为了调代码）

结果：（这个决策树有点奇怪，emmm，可能有点小问题，还没想好）



收获

1. 缺点：可能会出现过拟合，得到的决策树过于庞大，for example用全部数据集(至于剪枝暂时还没有考虑)
2. 学会了制作python包，并调用自写模块（其实也不算是自写（是别人的自写模块），因为只不过是需要自己封装一下，**treePlotter**——画树的一些函数，也算是收藏一下，刚好学完数据结构，对树的函数也比较了解）

```
import matplotlib.pyplot as plt


"""绘决策树的函数"""
decisionNode = dict(boxstyle="sawtooth", fc="0.8")  # 定义分支点的样式
leafNode = dict(boxstyle="round4", fc="0.8")  # 定义叶节点的样式
arrow_args = dict(arrowstyle="<-")  # 定义箭头标识样式


# 计算树的叶子节点数量
def getNumLeafs(myTree):
    numLeafs = 0
    firstStr = list(myTree.keys())[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        if type(secondDict[key]).__name__ == 'dict':
            numLeafs += getNumLeafs(secondDict[key])
        else:
            numLeafs += 1
    return numLeafs


# 计算树的最大深度
def getTreeDepth(myTree):
    maxDepth = 0
    firstStr = list(myTree.keys())[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
```

```
        if type(secondDict[key]).__name__ == 'dict':
            thisDepth = 1 + getTreeDepth(secondDict[key])
        else:
            thisDepth = 1
        if thisDepth > maxDepth:
            maxDepth = thisDepth
    return maxDepth


# 画出节点
def plotNode(nodeTxt, centerPt, parentPt, nodeType):
    createPlot.ax1.annotate(nodeTxt, xy=parentPt, xycoords='axes
fraction', \
                            xytext=centerPt, textcoords='axes fraction',
va="center", ha="center", \
                            bbox=nodeType, arrowprops=arrow_args)


# 标箭头上的文字
def plotMidText(cntrPt, parentPt, txtString):
    lens = len(txtString)
    xMid = (parentPt[0] + cntrPt[0]) / 2.0 - lens * 0.002
    yMid = (parentPt[1] + cntrPt[1]) / 2.0
    createPlot.ax1.text(xMid, yMid, txtString)


def plotTree(myTree, parentPt, nodeTxt):
    numLeafs = getNumLeafs(myTree)
    depth = getTreeDepth(myTree)
    firstStr = list(myTree.keys())[0]
    cntrPt = (plotTree.xOff + \
              (1.0 + float(numLeafs)) / 2.0 / plotTree.totalW,
plotTree.yOff)
    plotMidText(cntrPt, parentPt, nodeTxt)
    plotNode(firstStr, cntrPt, parentPt, decisionNode)
    secondDict = myTree[firstStr]
    plotTree.yOff = plotTree.yOff - 1.0 / plotTree.totalD
    for key in secondDict.keys():
        if type(secondDict[key]).__name__ == 'dict':
            plotTree(secondDict[key], cntrPt, str(key))
        else:
            plotTree.xOff = plotTree.xOff + 1.0 / plotTree.totalW
            plotNode(secondDict[key], \
                     (plotTree.xOff, plotTree.yOff), cntrPt, leafNode)
            plotMidText((plotTree.xOff, plotTree.yOff) \
                        , cntrPt, str(key))
    plotTree.yOff = plotTree.yOff + 1.0 / plotTree.totalD
```

```
def createPlot(inTree):
    fig = plt.figure(1, facecolor='white')
    fig.clf()
    axprops = dict(xticks=[], yticks=[])
    createPlot.ax1 = plt.subplot(111, frameon=False, **axprops)
    plotTree.totalW = float(getNumLeafs(inTree))
    plotTree.totalD = float(getTreeDepth(inTree))
    plotTree.xOff = -0.5 / plotTree.totalW
    plotTree.yOff = 1.0
    plotTree(inTree, (0.5, 1.0), '')
    plt.show()

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

3. 关于决策树，更加深入了解了**信息熵**的概念和一些列表的操作，但后面的构造决策树理解的不是很透彻

支持向量机（svm）

原理

1. 根据分类指标确定了点的空间位置，找到一个超平面
2. 转化成的问题：所有点到超平面的距离最大，即有一个限制条件的单目标优化、二次规划（**拉格朗日乘数法**），但此处用到了对偶问题（现成的优化计算包求解也可以，但对偶问题更高效），确定参数，但一个最大距离可能存在多组解
3. 解决方法：**SMO**，固定其余参数，仅优化两个，求出极值，重复循环直至收敛（必须满足**KKT条件**）
4. 确定偏移项b：（方程求解），但更鲁棒的做法：所有支持向量求解的平均值

实现

支持向量机（调库版-计算准确率）（来自数模中的应用实现）

```
from sklearn import svm
import pandas as pd
import numpy as np
import sklearn

if __name__ == '__main__':
    file_path = r"C:/Users/华为/Desktop/程序/svm-data.xlsx"
    data = pd.DataFrame(pd.read_excel(file_path))
    print(r'''Info:
    1. month: 1~12
    2. nation: 汉族
```

```
    3. sex：1(男)，0(女)
    4. Ispoorpot：0(Flase)，1(True)
    5. poor_level：0(非贫困生)，1(家庭经济困难、家庭经济特别困难、突发事件特殊困难)
    6. dormi：0(丁香)，1(海棠)，2(竹园)''')
    x, y = np.split(data.values, indices_or_sections=(6,), axis=1)
    x = x[:, 0:6]
    train_data, test_data, train_label, test_label = sklearn.model_selection.train_test_split(
        x, y, random_state=1)
    classifier = svm.SVC(C=1, kernel='rbf', gamma=10,
                         decision_function_shape='ovo')  # ovr:一对多策略 rbf:Gaussian kernel function
    classifier.fit(train_data, train_label.ravel())
    print("训练集：", classifier.score(train_data, train_label))
    print("测试集：", classifier.score(test_data, test_label))
    a = classifier.decision_function(x)
    b = classifier.predict(x)
    print('train_decision_function:', a)
    print('predict_result:', b)
```

数据来源：和决策树一样

准确率（似乎还可以，不可以也没办法）：

- 训练集： 0.8324516853183379
- 测试集： 0.7761756321992366

结果：预测分类结果、每个空间点到超平面的距离（用这个可以刻画程度）

支持向量机（调库版-画图）（为了可以画图，就生成随机数，只分两类，用三种核函数，进行对比）

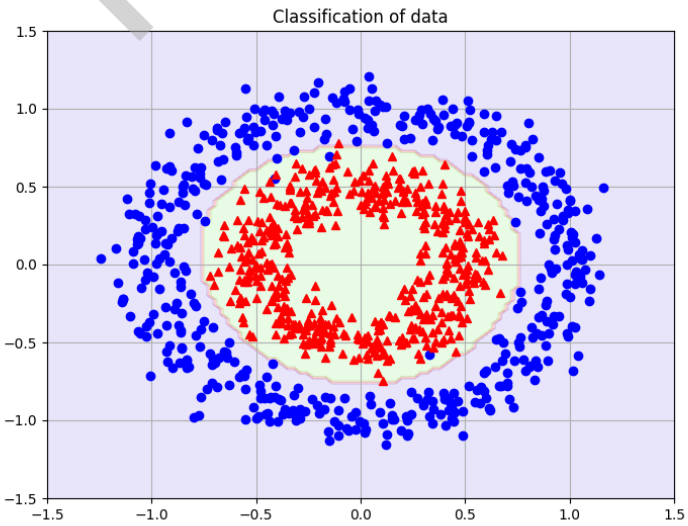
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.datasets import make_circles

#Data preparation
X, y = make_circles(n_samples=1000,factor=0.5,noise=0.1)

#Gridding diagram
x0s = np.linspace(-1.5, 1.5, 100)
x1s = np.linspace(-1.5, 1.5, 100)
x0, x1 = np.meshgrid(x0s, x1s)
Xtest = np.c_[x0.ravel(), x1.ravel()]

# SVC-poly
```

```
modelSVM1 = SVC(kernel='poly', degree=3, coef0=0.2)  # 'poly' 多项式核函数
modelSVM1.fit(X, y)
yPred1 = modelSVM1.predict(Xtest).reshape(x0.shape)

# SVC-rbf
modelSVM2 = SVC(kernel='rbf', gamma='scale')  #'rbf' 高斯核函数
modelSVM2.fit(X, y)
yPred2 = modelSVM2.predict(Xtest).reshape(x0.shape)

#SVC-sigmoid
modelSVM3 = SVC(kernel='sigmoid', gamma='scale')  #'sigmoid' S型核函数
modelSVM3.fit(X, y)
yPred3 = modelSVM3.predict(Xtest).reshape(x0.shape)

#Classification result rendering
fig, ax = plt.subplots(figsize=(8, 6))
ax.contourf(x0, x1, yPred1, cmap=plt.cm.brg, alpha=0.1)
ax.contourf(x0, x1, yPred2, cmap='PuBuGn_r', alpha=0.1)
ax.contourf(x0, x1, yPred3, cmap='CMRmap', alpha=0.1)

ax.plot(X[:,0][y==0], X[:,1][y==0], "bo")
ax.plot(X[:,0][y==1], X[:,1][y==1], "r^")
ax.grid(True, which='both')
ax.set_title("Classification of data")
plt.show()
```

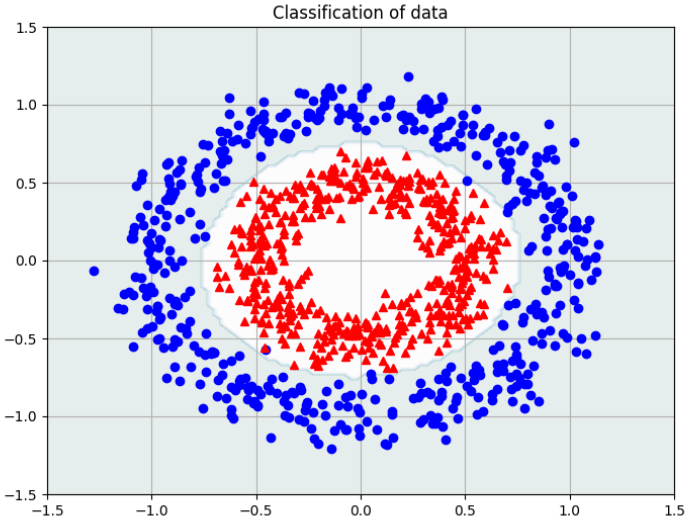
数据来源：库里的make_circle

结果：

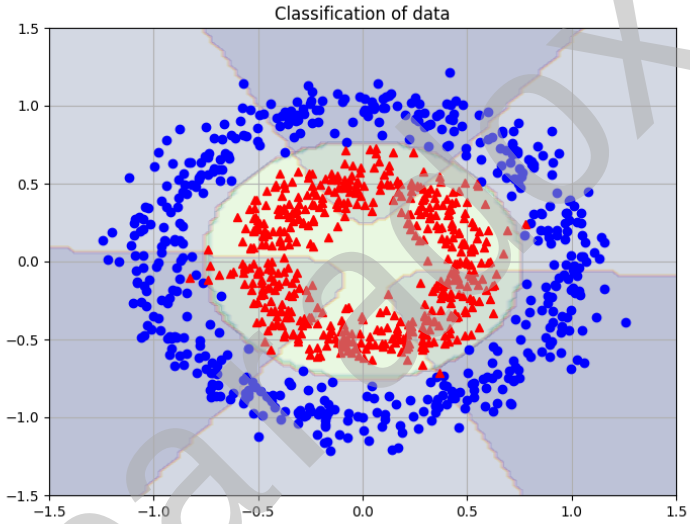
1. poly



2. rbf：



3. poly-rbf-sigmoid：



分析：很明显sigmoid非常不适合这个数据分类，poly和rbf的分类结果基本类似，准确率都非常高

- [ ] 支持向量机（原理版-暂时还没有实现，设置为未完成的任务）

收获

1. 感受到了支持向量机的强大魅力，百闻不如一见，针对大部分问题分类效果确实很好
2. 核函数选取的重要性，如果选择不恰当效果很不好
3. 学习了优化的方法

番外

1. **KKT条件**：解决最优化问题的方法（和拉格朗日乘子均是求解最优解的方法），是必要条件，针对凸函数也是充分条件

- KKT与SMO：如果不满足，目标函数迭代后会增大。SMO采用了一个启发式：使选取的两变量所对应样本直接的间隔最大（直观解释：我觉得和灵敏度分析的思路有点像）

2. **sigmoid函数**(Logistic函数）：用于**隐层神经元**输出，取值范围为(0,1)，它可以将一个实数映射到(0,1)的区间，可以用来做二分类。在特征相差比较复杂或是相差不是特别大时效果比较好。（Logistic函数：非常熟悉的**人口增长模型**，好像也是传染病的一个模型，比如西安疫情就可以用这个分析）

$$S(x) = \frac{e^x}{1+e^x}$$

3. **硬间隔**：所有样本划分必须正确；**软间隔**：允许一些样本不满足约束条件

损失函数：hinge函数、指数损失函数、对率损失函数

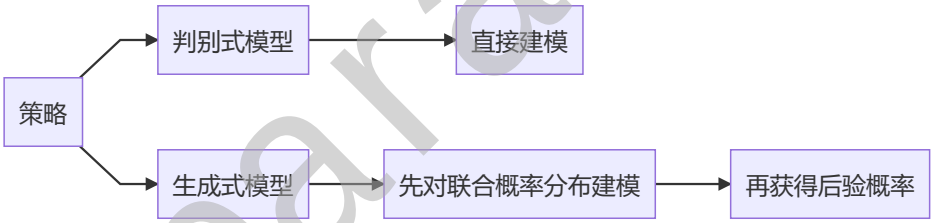
贝叶斯分类器

贝叶斯决策论

1.条件风险：

$$R(c_i \mid \boldsymbol{x}) = \sum_{j=1}^{N} \lambda_{ij} P(c_j \mid \boldsymbol{x})$$

2.机器学习所实现：基于有限的训练样本集尽可能准确的估计出后验概率。



极大似然估计（参数估计）-MLE

似然函数：分布函数的连乘 --> 取对数 -> 求偏导等于0，即可得到最大似然估计量

But这种参数化的方法虽能使类条件概率估计变得相对简单，但估计结果的准确性严重依赖于**所假设的概率分布形式**是否符合潜在的真实数据分布。

```
from sklearn import datasets
iris = datasets.load_iris()
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf = clf.fit(iris.data, iris.target)
y_pred=clf.predict(iris.data)
accuracy = (iris.target != y_pred).sum() / iris.data.shape[0]
print("MLE-Number of samples %d wrong-sample : %d accuracy: %f" %
(iris.data.shape[0],(iris.target != y_pred).sum(), accuracy))
```

结果：MLE-Number of samples 150 wrong-sample : 6 accuracy: 0.040000

朴素贝叶斯分类器

——属性条件独立性假设

多项式朴素贝叶斯分类器

```
from sklearn import datasets
iris = datasets.load_iris()
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf = clf.fit(iris.data, iris.target)
y_pred=clf.predict(iris.data)
accuracy = (iris.target != y_pred).sum() / iris.data.shape[0]
print("Polynomial naive Bayes-Number of samples %d wrong-sample : %d
accuracy: %f" % (iris.data.shape[0],(iris.target != y_pred).sum(),
accuracy))
```

结果：Polynomial naive Bayes-Number of samples 150 wrong-sample : 7 accuracy: 0.046667

伯努利朴素贝叶斯分类器

```
from sklearn import datasets
iris = datasets.load_iris()
from sklearn.naive_bayes import BernoulliNB
clf = BernoulliNB()
clf = clf.fit(iris.data, iris.target)
y_pred=clf.predict(iris.data)
accuracy = (iris.target != y_pred).sum() / iris.data.shape[0]
print("Bernoulli naive Bayes--Number of samples %d wrong-sample : %d
accuracy: %f" % (iris.data.shape[0],(iris.target != y_pred).sum(),
accuracy))
```

结果：Bernoulli naive Bayes--Number of samples 150 wrong-sample : 100 accuracy: 0.666667

半朴素贝叶斯分类器

1. 半朴素贝叶斯分类器的基本想法是适当考虑一部分属性问的相互依赖信息。
2. 独依赖估计 (ODE)是半朴素贝叶斯分类器最常用的一种策略。
3. SPODE方法

TAN (Tree Augmented naïve Bayes) [Friedman et al., 1997] 则是在最大带权生成树(maximum weighted spanning tree)算法 [Chow and Liu, 1968] 的基础上, 通过以下步骤将属性间依赖关系约简为如图 7.1(c) 所示的树形结构:

(1) 计算任意两个属性之间的条件互信息(conditional mutual information)

$$I(x_i, x_j \mid y) = \sum_{x_i, x_j;\, c \in \mathcal{Y}} P(x_i, x_j \mid c) \log \frac{P(x_i, x_j \mid c)}{P(x_i \mid c) P(x_j \mid c)}; \tag{7.22}$$

(2) 以属性为结点构建完全图, 任意两个结点之间边的权重设为 $I(x_i, x_j \mid y)$;

(3) 构建此完全图的最大带权生成树, 挑选根变量, 将边置为有向;

(4) 加入类别结点 y, 增加从 y 到每个属性的有向边.

贝叶斯网

借助无环图来刻画属性之间的依赖关系，并使用条件概率表（CPT）来描述属性的联合概率分布。

1. 分析有向图中变量的条件独立性：

- 找出有向图的所有V型结构，在V型结构的两个父结点之间加上一条无向边
- 将所有有向边改为无向边

无向图——道德图

2. 常用评分函数通常基于**信息论准则**，此类准则将学习问题看作一个**数据压缩任务**，学习的目标是找到一个能以**最短编码长度**描述训练数据的模型，此时编码的长度包括了描述模型自身所需的**字节长度**和使用该模型描述数据所需的字节长度。对贝叶斯网学习而言，模型就是一个贝叶斯网，同时，每个贝叶斯网描述了一个在训练数据上的**概率分布**，自有一套编码机制能使那些经常出现的样本有更短的编码。于是，我们应选择那个**综合编码长度**（包括描述网络和编码数据）最短的贝叶斯网，这就是**最小描述长度"**(MDL)准则。

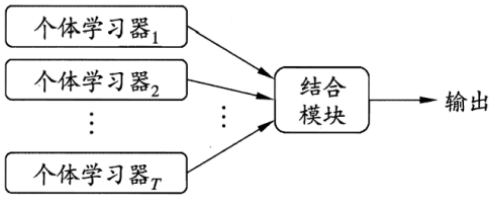
EM算法

若参数θ已知，则可根据训练数据推断出最优隐变量Z的值 (E 步)；反之，若Z的值已知，则可方便地对参数Θ做极大似然估计 (M 步) 。两步交替进行，直至收敛。

集成学习

个体与集成

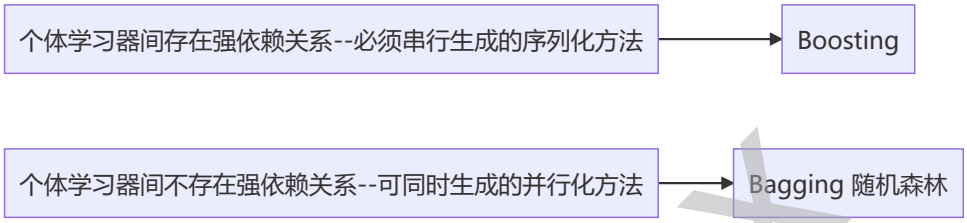
1.集成学习（多分类器系统、基于委员会的学习）：通过**构建**并**结合**多个学习器来完成学习任务。（感觉好像集成电路）



优点：可获得比单一学习器显著优越的泛化性能。

2.假设集成通过简单投票法结合T个基分类器，若有超过半数的基分类器正确， 则集成分类就正确，随着集成中个体分类器数目T的增大，集成的错误率将指数级下降，最终趋向于零。（关键假设：**基学习器的误差相互独立**）

3.两大类：



Boosting

1.一族将弱学习器提升为强学习器的算法。

2.**AdaBoost**

输入： 训练集 $D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_m,y_m)\}$;
基学习算法 $\mathfrak{L}$;
训练轮数 T.

过程：

1: $\mathcal{D}_1(\boldsymbol{x})=1/m$.
2: **for** $t=1,2,\ldots,T$ **do**
3: $h_t=\mathfrak{L}(D,\mathcal{D}_t)$;
4: $\epsilon_t=P_{\boldsymbol{x}\sim\mathcal{D}_t}(h_t(\boldsymbol{x})\neq f(\boldsymbol{x}))$;
5: **if** $\epsilon_t>0.5$ **then break**
6: $\alpha_t=\frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$;
7: $\mathcal{D}_{t+1}(\boldsymbol{x})=\frac{\mathcal{D}_t(\boldsymbol{x})}{Z_t}\times\begin{cases}\exp(-\alpha_t), & \text{if } h_t(\boldsymbol{x})=f(\boldsymbol{x})\\ \exp(\alpha_t), & \text{if } h_t(\boldsymbol{x})\neq f(\boldsymbol{x})\end{cases}$
$=\frac{\mathcal{D}_t(\boldsymbol{x})\exp(-\alpha_t f(\boldsymbol{x})h_t(\boldsymbol{x}))}{Z_t}$
8: **end for**

输出： $H(\boldsymbol{x})=\operatorname{sign}\left(\sum_{t=1}^{T}\alpha_t h_t(\boldsymbol{x})\right)$

图 8.3 AdaBoost算法

```
from sklearn.model_selection import cross_val_score
from sklearn import datasets
# prepare data
iris = datasets.load_iris()
X, y = iris.data[:, 1:3], iris.target
# AdaBoosting
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier(n_estimators=100)  #Control the number of
learners
scores = cross_val_score(clf, X, y)
print('AdaBoost-accuracy:',scores.mean())
```

结果：AdaBoost-accuracy: 0.72

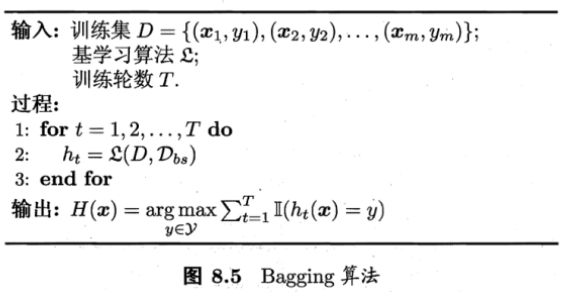
3.**Gradient Tree Boosting**

```
from sklearn.model_selection import cross_val_score
from sklearn import datasets
# prepare data
iris = datasets.load_iris()
X, y = iris.data[:, 1:3], iris.target
# Gradient Tree Boosting
from sklearn.ensemble import GradientBoostingClassifier
clf = GradientBoostingClassifier(n_estimators=100,
learning_rate=1.0,max_depth=1, random_state=0)
scores = cross_val_score(clf, X, y)
print('GDBT-accuracy:',scores.mean())
```

结果：GDBT-accuracy: 0.9066666666666666

Bagging

自助采样法（优点：剩余样本可以用作验证集对泛化性能进行“包外估计”）——采样出T个含m个训练样本的采样集——每个采样集训练出一个基学习器



输入： 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
基学习算法 $\mathfrak{L}$;
训练轮数 T.

过程：

1: **for** $t = 1, 2, \ldots, T$ **do**
2: $h_t = \mathfrak{L}(D, \mathcal{D}_{bs})$
3: **end for**

输出： $H(\boldsymbol{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbb{I}(h_t(\boldsymbol{x}) = y)$

图 8.5 Bagging 算法

```
from sklearn.model_selection import cross_val_score
from sklearn import datasets
# prepare data
iris = datasets.load_iris()
X, y = iris.data[:, 1:3], iris.target
# Bagging
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
bagging = BaggingClassifier(KNeighborsClassifier(),max_samples=0.5,
max_features=0.5)
scores = cross_val_score(bagging, X, y)
print('Bagging-accuracy:',scores.mean())
```

结果：Bagging-accuracy: 0.9400000000000001

随机森林

简单、开销小，而且效果好

```
from sklearn.model_selection import cross_val_score
from sklearn import datasets
# prepare data
iris = datasets.load_iris()
X, y = iris.data[:, 1:3], iris.target
# Random forest
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=10,max_features=2)
clf = clf.fit(X, y)
scores = cross_val_score(clf, X, y)
print('Random forest-accuracy:',scores.mean())
```

结果：Random forest-accuracy: 0.9266666666666667

结合策略

优点：

1. 若使用单学习器可能因误选而导致泛化性能不佳，结合多个学习器会减小风险；
2. 降低陷入糟糕**局部极小点**的风险
3. 由于相应的假设空间有所扩大，有可能学得更好的近似。

常见方法：平均法（简单和加权）、投票法（绝对多数、相对多数、加权）、学习法（Stacking）

多样性

——误差分歧分解：个体学习器准确性越高、多样性越大，则集成越好。

1. 多样性度量：

1.多样性度量：

- 不合度量：$dis_{ij}=\frac{b+c}{m}$
- 相关系数：$\rho_{ij}=\frac{ad-bc}{\sqrt{(a+b)(a+c)(c+d)(b+d)}}$
- Q统计量：$Q_{ij}=\frac{ad-bc}{ad+bc}$
- k统计量：$\kappa=\frac{p_1-p_2}{1-p_2}$

2.多样性增强：

- 数据样本扰动
- 输入属性扰动
- 输出表示扰动
- 算法参数扰动

聚类分析

模型介绍

原理

完全是基于数据的**空间分布结构**，属于**无监督学习**；聚类分析的目标就是在相似的基础上收集数据来分类，得到较高的簇内相似度和较低的簇间相似度，使得簇间的距离尽可能大，簇内样本与簇中心的距离尽可能小。

- 聚类中心是一个簇中所有样本点的均值(质心)
- 簇大小表示簇中所含样本的数量
- 簇密度表示簇中样本点的紧密程度
- 簇描述是簇中样本的业务特征

k-means 聚类（基于划分）

1. 选择 K 个初始质心(K需要用户指定），初始质心随机选择即可，每一个质心为一个类；
2. 对剩余的每个样本点，计算它们到各个质心的欧式距离，并将其归入到相互间距离最小的质心所在的簇。计算各个新簇的质心；
3. 在所有样本点都划分完毕后，根据划分情况重新计算各个簇的质心所在位置，然后迭代计算各个样本点到各簇质心的距离，对所有样本点重新进行划分；
4. 重复2. 和 3.，直到质心不在发生变化时或者到达最大迭代次数时。

BIRCH（基于层次）

BIRCH利用层次方法的平衡迭代规约和聚类）主要是在数据量很大的时候使用，而且数据类型是numerical。首先利用树的结构对对象集进行划分，然后再利用其它聚类方法对这些聚类进行优化；

谱聚类

谱聚类是一种基于数据相似度矩阵的聚类方法，它定义了子图划分的优化目标函数，并作出改进（RatioCut和NCut），引入指示变量，将划分问题转化为求解最优的指示变量矩阵HH。然后利用瑞利熵的性质，将该问题进一步转化为求解拉普拉斯矩阵的kk个最小特征值，最后将 HH 作为样本的某种表达，使用传统的聚类方法进行聚类。（很复杂的样子）

高斯混合模型

高斯混合模型可以看作是由 K 个单高斯模型组合而成的模型，这 K 个子模型是混合模型的隐变量。

实现

聚类分析（调库版）

```
import pandas as pd
import numpy as np
import sklearn.cluster as sc
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import Birch
from sklearn.cluster import MiniBatchKMeans
from sklearn.cluster import SpectralClustering
from sklearn.mixture import GaussianMixture
excel = pd.read_excel(r"C:\Users\华为\Desktop\聚类分析.xls")
x = np.array(excel)

rel = pd.DataFrame()
table = None
color_map = ['k', 'pink', 'r', 'y', 'g']

def draw(titile, table):
    fig = plt.figure()  # 建立一个空间
    ax = fig.add_subplot(111, projection='3d')  # 3D坐标
    for i in range(114):
        ax.scatter(x[i][0],x[i][1],x[i][2], color=color_map[table[i]])
    plt.show()

model = sc.KMeans(n_clusters=5)# k-means 聚类
yhat = model.fit_predict(x)
table = model.predict(x)
print(table)
rel['KMeans'] = table
draw("KM", table)

model = AgglomerativeClustering(n_clusters=5) #层次聚类
yhat = model.fit_predict(x)
# table = model.predict(x)
```

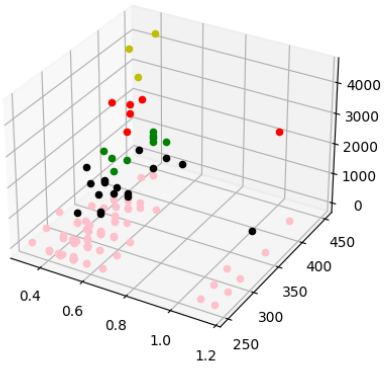
```
# print(table)
# rel['AgglomerativeClustering'] = table
draw("AC", table)

model = Birch(threshold=0.01, n_clusters=5) #Birch（平衡迭代削减聚类法）
yhat = model.fit_predict(x)
table = model.predict(x)
print(table)
rel['Birch'] = table
draw("Birch", table)

model = MiniBatchKMeans(n_clusters=5) # Mini-Batch K-均值（K-均值修改版）
yhat = model.fit_predict(x)
table = model.predict(x)
print(table)
rel['MiniBatchKMean'] = table
draw("MiniBatchKMean", table)

model = SpectralClustering(n_clusters=5) # 光谱聚类
# yhat = model.fit_predict(x)
# table = model.predict(x)
# print(table)
# rel['SpectralClustering'] = table
draw("SpectralClustering", table)

model = GaussianMixture(n_components=5) # 高斯混合模型
yhat = model.fit_predict(x)
table = model.predict(x)
print(table)
rel['GaussianMixture'] = table
draw("GaussianMixture", table)
```


数据来源：化学反应的加权数、温度、产量构成的三维图（但这好像不重要，只是单纯想学习分个层而已，至于为什么用了6种方法去实现，因为有趣，看图差别好像不是很大）

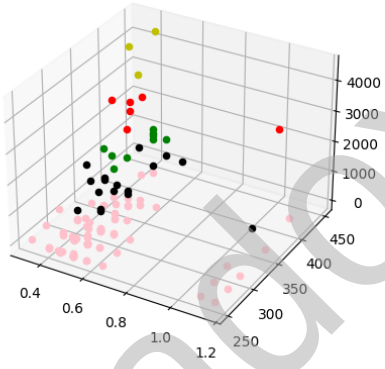
结果：（看图感觉没啥区别）

K-means

```
[1 1 4 4 2 1 1 4 2 0 1 1 4 4 2 3 3 1 1 1 4 2 3 1 1 1 1 4 0 1 1 1 4 0 1
1 1
 4 0 1 1 1 4 0 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 4 1 1 1
1 4
 1 1 1 4 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 4 1 1 1 1 1 4 1 1 1 1 4 2 1
1 1
 1 4 0]
```

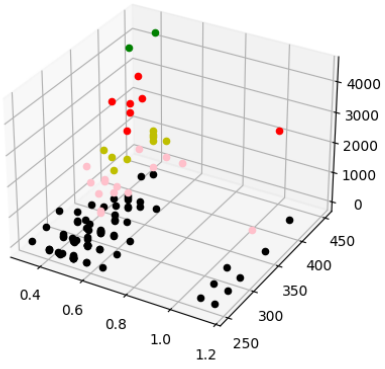


层次聚类



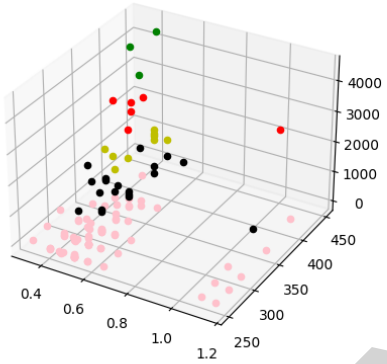
Birch

```
[0 0 1 1 3 0 0 1 3 2 0 0 0 1 3 4 4 0 0 0 1 3 2 0 0 0 0 1 2 0 0 0 1 2 0
0 0
 1 2 0 0 0 1 2 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 1 0 0 0
0 1
 0 0 0 0 3 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 3 0
0 0
 0 1 2]
```

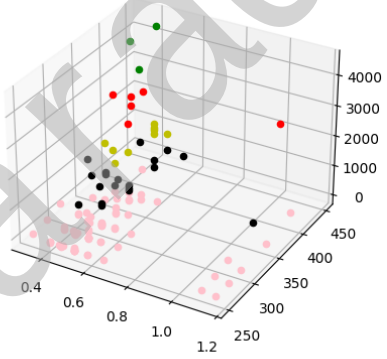


~~K-均值~~

```
[1 1 0 0 4 1 1 0 4 3 1 1 0 0 4 2 2 1 1 1 0 4 2 1 1 1 1 0 3 1 1 1 0 3 1
1 1
 0 3 1 1 1 0 3 1 1 1 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 4 1 1 1 1 0 1 1 1
1 0
 1 1 1 0 4 1 1 1 1 4 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 0 4 1
1 1
 1 0 3]
```

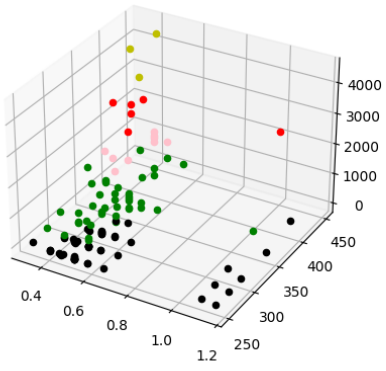


光谱聚类



高斯混合模型

```
[0 3 3 3 1 0 3 3 1 4 0 3 3 3 1 2 2 0 0 3 3 1 2 0 0 3 3 3 4 0 0 0 3 4 0
3 3
 3 4 0 0 3 3 4 0 0 0 3 1 0 0 0 0 0 0 0 3 3 0 0 0 3 1 0 0 0 3 3 0 0 0
3 3
 0 0 0 3 1 0 0 0 3 1 0 0 0 3 3 3 0 0 0 0 3 3 0 0 0 0 3 3 0 0 0 3 3 1 0
0 3
 3 3 4]
```



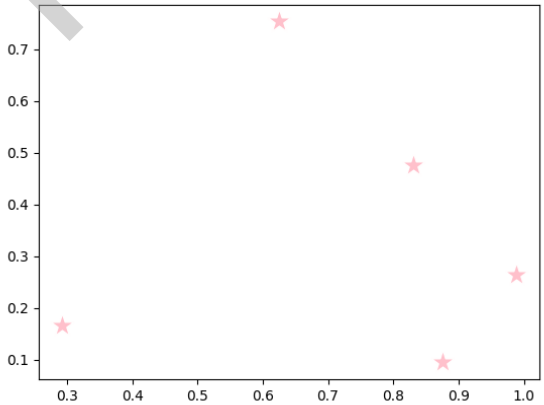
K-means（原理版）

```
import numpy as np
import matplotlib.pyplot as plt
class Kmeans:
    def __init__(self,data,k):
        self.data=data  #定义实例属性
        self.k = k

    def kmeans(self):
        dataset = np.random.random(size=self.k)
        dataset = np.floor(dataset * len(self.data))
        dataset = dataset.astype(int)
        print('Random index', dataset)  #k points in the data set were
randomly selected as the initial center points
        center = data[dataset]
        cls = np.zeros([len(self.data)], np.int)
        print('Init-center=\n', center)
        run = 1
        time = 0
        n = len(self.data)
        while run:
            time += 1
            for i in range(n):
                tmp = data[i] - center  #delta
                tmp = np.square(tmp)  #square
                tmp = np.sum(tmp, axis=1)  #Sum by row
                cls[i] = np.argmin(tmp)
            run = 0
            # Compute the center point of each class
            for i in range(self.k):
                club = data[cls == i] #Find all the points in this
class
                new = np.mean(club, axis=0)
                instance = np.abs(center[i] - new)
                if np.sum(instance, axis=0) > 1e-4:   #If the distance
of the new center is small, it can be regarded as the same class
```

```
                    center[i] = new
                    run = 1
            print('new center=\n', center)
        print('The number of iterations:', time)
        # get different kinds of graphs
        for i in range(self.k):
            club = data[cls == i]
            self.show(club)
        #final center
        self.show(center)
        print('cluster labels:')
        print(cls)

    def show(self,data):  #draw graph
        x = data.T[0]
        y = data.T[1]
        plt.scatter(x, y, s=300, c='pink', edgecolors='w', marker='*')
        plt.show()

if __name__ == '__main__':
    data = np.random.rand(20,2)
    K = 5
    model = Kmeans(data,K)
    model.kmeans()
```

结果:

```
Random index [4 9 6 0 5]
The number of iterations: 3
cluster labels:[2 0 0 4 1 0 0 0 1 2 1 0 1 1 2 1 2 1 3 4]
finalcenter-graph:
```



收获

1.简单了解K-means以外的其他聚类（听起来都很高级的样子）

2.学习了用类实现二维K-means的算法，以及画散点图（再一次觉得plt库好强大，真的好奇背后的实现，居然还有边缘颜色，和点的形状）

降维学习

主成分分析（PCA）

代码学习from《Python深度学习：基于TensorFlow》

原理

通过正交变换将一组可能存在相关性的变量转换为一组线性不相关的变量，转换后的这组变量叫主成分。（思路是比较简单的，但也是神奇的）

转化后的问题：设在n维空间中有m个样本点：{x1,x2,..., xm}，假设m比较大，需要对这些点进行压缩（重构数据），使其投影到k维空间中，其中k<n，同时使损失的信息最小（带约束的单目标优化，拉格朗日乘数法）。

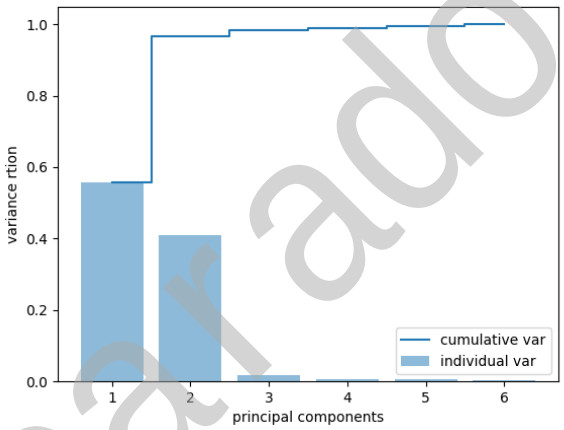
实现

（好像没有找到合适的库，所以只有原理版）

```
import numpy as np
import pandas as pd
from numpy.linalg import eig
import seaborn as sns
import matplotlib.pyplot as plt

def pca(X,k):
    X = X - X.mean(axis = 0) #向量X去中心化
    X_cov = np.cov(X.T, ddof = 0) #计算向量X的协方差矩阵，自由度可以选择0或1
    eigenvalues,eigenvectors = eig(X_cov) #计算协方差矩阵的特征值和特征向量
    klarge_index = eigenvalues.argsort()[-k:][::-1] #选取最大的K个特征值及其特征向量
    k_eigenvectors = eigenvectors[klarge_index] #用X与特征向量相乘
    return np.dot(X, k_eigenvectors.T)

excel = pd.read_excel(r"C:\Users\华为\Desktop\程序\主成分分析.xlsx")
X = np.array(excel)
k = 2 #降维后的维度
X_pca = pca(X, k)
print(X_pca)

X = X - X.mean(axis = 0)

#计算协方差矩阵
X_cov = np.cov(X.T, ddof = 0)
```

```
#计算协方差矩阵的特征值和特征向量
eigenvalues,eigenvectors = eig(X_cov)

tot = sum(eigenvalues)
var_exp = [(i/tot) for i in sorted(eigenvalues, reverse = True)]
cum_var_exp = np.cumsum(var_exp)

plt.bar(range(1,7), var_exp, alpha = 0.5, align = 'center', label =
'individual var')
plt.step(range(1,7), cum_var_exp, where = 'mid', label = 'cumulative
var')
plt.ylabel('variance rtion')
plt.xlabel('principal components')
plt.legend(loc = 'best')
plt.show()
```

结果：主成分分析-各特征值的贡献率：前两个加起来大概96%，k=2，比较合理



学习后的疑问：（希望后面可以得到答案）

？：似乎得不到主成分是哪几个（还没想好）

？：尝试过若某一个向量的值非常大，进行主成分分析，会出现某一个特征值贡献率基本等于

（是否是值大的那个向量待探究）

番外

1. 懒惰学习著名代表——**k近邻学习（kNN）**：给定测试样本，基于某种距离度量找出训练集中与其最靠近的k个训练样本，然后基于这k个"邻居"的信息来进行预测。（与之相反的是**急切学习**）

分类任务——投票法 回归任务——平均法 基于距离加权（有点像指数平滑法）

总结：虽然算法思路简单，但它的泛化错误比较低

2. 低维嵌入：（背景：维数灾难）

3. **核化线性降维**：线性降维方法假设从高维空间到低维空间的函数映射是线性的，但线性可能会失真，故对线性降维进行核化。

4. **流形学习**：拓扑流形概念的降维方法。低维流形嵌入到高维空间中，仍具有欧氏空间的性质，所以容易在局部建立降维映射关系，然后设法将局部映射关系推广到全局。

- **等度量映射**：在高维空间中直线距离具有误导性，低维流形嵌入两点的距离是"测地线"（蚂蚁爬立方体问题），所以将其转变为计算近邻连接图两点之间的**最短路径问题。**（Dijkstra/Floyd Isomap）
- 局部线性嵌入（LLE）：保持邻域内样本之间的**线性关系**

5. **度量学习**：（降维的目的：对高维数据降维希望找到一个合适的低维空间，在此空间中进行学习能比原始空间性能更好）"学习"出一个合适的距离度量。

特征学习与稀疏学习

子集搜索与评价

1. **特征选择**的reason：

- 解决维数灾难问题，和降维的动机类似，且与降维为处理高维数据的两大主流技术。
- 去除不相关特征往往会降低学习任务的难度，只留下key。

2. **选择特征子集**：

- 子集搜索问题：前向搜索——增加相关特征，后向搜索——逐渐减少特征，双向搜索——前向加后向搜索（避免不了的bug：均为局部最优，而不一定是全局最优）

- 子集评价问题——计算信息增益——越大，意味着特征子集包含的有助于分类里的信息越多

3. 常见的特征选择方法：过滤式(filter)、包裹式(wrapper)、嵌入式(embedding)

explainment:

（1）**过滤式**——先对数据集进行特征选择，然后再训练学习器。**特征选择过程与后续学习器无关**，这相当于先对初始特征进行"过滤"，再用过滤后的特征训练模型。

过滤式选择的方法有：

- 移除低方差的特征；
- 相关系数排序，分别计算每个特征与输出值之间的相关系数，设定一个阈值，选择相关系数大于阈值的部分特征；
- 利用假设检验得到特征与输出值之间的相关性，方法有比如卡方检验、t检验、F检验等。
- 互信息，利用互信息从信息熵的角度分析相关性。

e.g. Relief（to 二分类）—有距离度量的意味（我个人jio得可能还要再看几遍）

（2）**包裹式**——从初始特征集合中不断的选择特征子集，训练学习器，**根据学习器的性能来对子集进行评价**，直到选择出最佳的子集。

优点：针对学习器进行优化，从最终学习器的性能来看，包裹式比过滤式更好；

缺点：由于特征选择过程中需要多次训练学习器，因此包裹式特征选择的计算开销通常比过滤式特征选择要大得多。

e.g.LVW：在**拉斯维加斯方法**框架下使用随机策略进行自己搜索，并以最终分类器的误差为特征子集评价准则。

（3）**嵌入式选择与L1正则化**：嵌入式选择——将特征选择过程与学习器训练过程融为一体，两者在同一个优化过程中完成，即在学习器训练过程中自动地进行了特征选择。

L1范数正则化——更易获得稀疏解，即d个特征中仅有对应着w的非零分量的特征才会出现在最终模型中。

L1正则化问题的求解——近端梯度下降（PGD）

4. 番外：

（1）拉斯维加斯方法——基于随机数进行求解，和蒙特卡洛一样，是一种**思想**，但它在生成随机值的环节中，会不断的进行尝试，直到生成的随机值令自己满意。但可能会存在停不下来的情况，需要设置停止条件控制参数

（2）岭回归——共线性数据分析的有偏估计回归的正则化方法，实质上是一种改良的最小二乘估计法，通过放弃最小二乘法的无偏性，以损失部分信息、降低精度为代价获得回归系数更为符合实际、更可靠的回归方法，对病态数据的拟合要强于最小二乘法 。

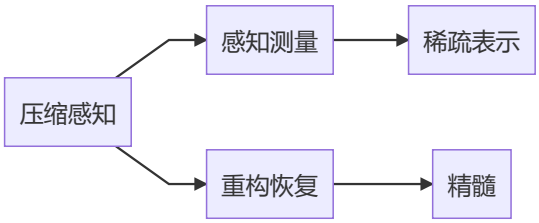
（3）LASSO回归：将岭回归的L2范数正则化变成L1范数正则化

稀疏表示与字典学习

字典学习（稀疏编码）——为普通稠密表达的样本找到合适的字典，将样本转化为合适的稀疏表示形式，从而使学习更加简便，模型复杂度降低。（只简单了解了一下）

压缩感知

1.背景：利用奈奎斯特采样定理将模拟信号转换为数字信号，但数字信号压缩后会损失一些信息，且无法得到原信号的精确数值解，但如果信号稀疏，未知因素的影响大为减少。



2.番外：限定等距性（RIP）：

Definition 1.1 (Restricted Isometry Constants): Let F be the matrix with the finite collection of vectors $(v_j)_{j\in J} \in \boldsymbol{R}^p$ as columns. For every integer $1 \leq S \leq |J|$, we define the S-restricted isometry constants δ_S to be the smallest quantity such that F_T obeys

$$(1-\delta_S)\|c\|^2 \leq \|F_T c\|^2 \leq (1+\delta_S)\|c\|^2 \qquad (1.7)$$

for all subsets $T \subset J$ of cardinality at most S, and all real coefficients $(c_j)_{j\in T}$.

计算学习理论

研究经验误差与泛化误差之间的逼近程度

PAC学习（概率近似正确学习理论）

1. 学习算法的**可分**：一个学习算法会有一个对应的目标概念集合，而在学习算法之前会有假设空间，如果假设空间包括了目标概念集合，则可将"真实的"目标学习概念和非目标学习概念集合分开，即可分。

2. **PAC辨识**：

$$P(E(h) \leq \epsilon) \geq 1-\zeta$$

3. **PAC可学习**：若学习算法能从假设空间中PAC辨识概念类，则称概念类对假设空间而言PAC可学习。（关键因素：假设空间的复杂度）

有限假设空间

1.可分情况

训练学习思路："真实的"假设空间是由目标概念决定，虽然无法判断正确的是否正确（这话说的有点奇怪），但是在训练集上出现标记错误的假设肯定不是目标概念，所以把标记错误的剔除，重复下去。收敛速度：O（$\frac{1}{m}$）

2.不可分情况

不可知PAC可学习：m为分布D中独立同分布采样的样例数目，若存在学习算法 £ 和多项式函数 poly，使得任何对于任何m>=poly，若 £ 能从假设空间H中输出满足下式：

$$P\big(E(h) - \min_{h'\in\mathcal{H}} E(h') \leqslant \epsilon\big) \geqslant 1-\delta$$

VC维

范围：无限假设空间

1.**增长函数**：每个假设对示例集进行标记，实例的样本越多，标记个数可能会越多，即假设空间的表示能力越强，也对学习任务的适应能力越强，反映出假设空间的复杂度。

2..对二分类问题来说，H中的假设对D中示例赋予标记的每种可能结果称为对D的一种**对分**。

3.若假设空间H能实现示例集D上的所有对分，即 增长函数 = 2^m,则称示例集D能被假设空间H**打散**。

4.假设空间H的**VC 维**是能被H打散的最大示例集的大小

$$\mathrm{VC}(\mathcal{H}) = \max\{m : \Pi_{\mathcal{H}}(m) = 2^m\}$$

由于与数据分布D无关，因此，在数据分布未知时依旧可以计算出假设空间H的VC维。

任何VC维有限地假设空间H都是（不可知）PAC可学习的。

Rademacher复杂度

（基于VC维的泛化误差界是分布无关、数据独立的）

Rademacher 复杂度是另一种刻画假设空间复杂度的途径，并且一定程度上考虑并依赖数据分布，所以泛化误差界更紧一些。（考虑随机误差）

稳定性

1. VC维和Rademacher 复杂度都与学习算法无关，此时提出**稳定性**，即**灵敏度分析**。
2. 输入发生变化时，研究输出的变化。
3. 若学习算法 £ 所输出的假设满足经验损失最小化，则称算法 £ 满足**经验风险最小化（ERM）**。
4. 若学习算法 £ 是ERM 且稳定的，则假设空间H可学习。

半监督学习

未标记样本

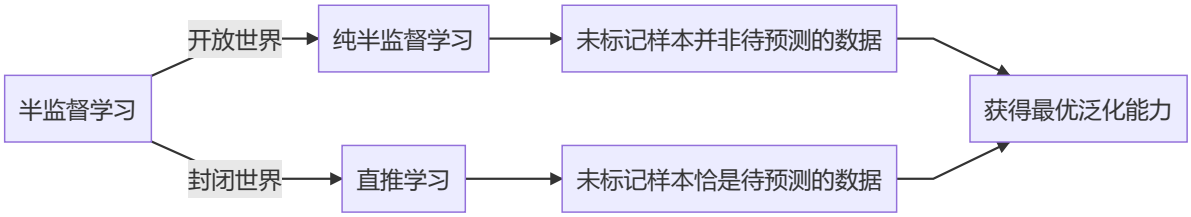
（在传统模型里，只能训练有标记的样本，而未标记的样本会被浪费掉）

1. **主动学习**：借用标记的样本训练模型，判断一个未标记的，判断之后将其归入标记样本，再训练模型。
2. **半监督学习**：让学习器不依赖于外界交互、自动地利用未标记样本来提升学习性能。
3. 常见假设：（基本假设：相似的样本拥有相似的输出）

聚类假设：（适用范围更广）假设数据存在簇结构，同一个簇的样本属于同一个类别，如果未标记样本和正例在一个簇，则为正例的可能性更大。

流形假设：假设数据分布在一个流形结构上，邻近的样本拥有相似的输出值（对输出值没有限制）

4.



5. 主动学习、纯半监督学习、直推学习的区别：

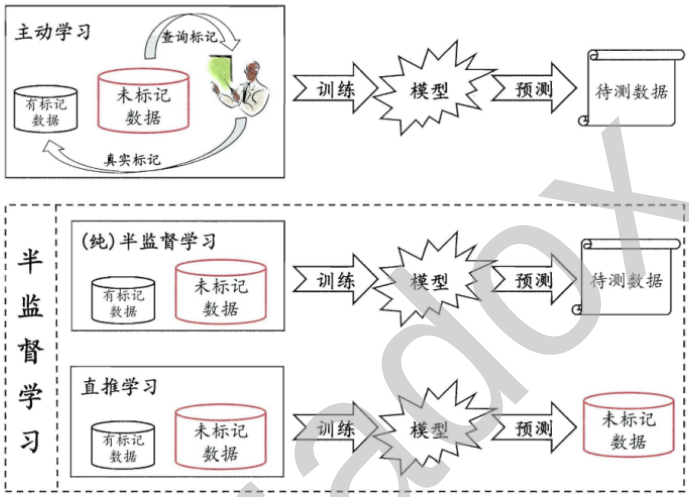


图 13.2 主动学习、(纯)半监督学习、直推学习

生成式方法

1. 生成式方法：假设所有数据（无论是否标记）都是由潜在的模型“生成”的。（想到了随机数和随机种子）未标记数据 is seen as 模型缺失参数，可进行参数估计（比如极大似然估计）
2. key：模型假设必须准确，即假设的生成式模型必须与真实数据分布吻合。(but it is so difficult)

半监督SVM（S3VM)

1. 在不考虑未标记样本时，支持向量机找到最大间隔划分超平面，再考虑标记样本,S3VM try to find 能将两类有标记样本分开且穿过数据低密度区域的划分超平面。（基本假设：低密度分隔）
2. TSVM

输入： 有标记样本集 $D_l = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_l, y_l)\}$;
未标记样本集 $D_u = \{\boldsymbol{x}_{l+1}, \boldsymbol{x}_{l+2}, \ldots, \boldsymbol{x}_{l+u}\}$;
折中参数 C_l, C_u.

过程：

1: 用 D_l 训练一个 SVM_l;
2: 用 SVM_l 对 D_u 中样本进行预测, 得到 $\hat{\boldsymbol{y}} = (\hat{y}_{l+1}, \hat{y}_{l+2}, \ldots, \hat{y}_{l+u})$;
3: 初始化 $C_u \ll C_l$;
4: **while** $C_u < C_l$ **do**
5: 基于 $D_l, D_u, \hat{\boldsymbol{y}}, C_l, C_u$ 求解式(13.9), 得到 $(\boldsymbol{w}, b)$, $\boldsymbol{\xi}$;
6: **while** $\exists \{i, j \mid (\hat{y}_i \hat{y}_j < 0) \wedge (\xi_i > 0) \wedge (\xi_j > 0) \wedge (\xi_i + \xi_j > 2)\}$ **do**
7: $\hat{y}_i = -\hat{y}_i$;
8: $\hat{y}_j = -\hat{y}_j$;
9: 基于 $D_l, D_u, \hat{\boldsymbol{y}}, C_l, C_u$ 重新求解式(13.9), 得到 $(\boldsymbol{w}, b)$, $\boldsymbol{\xi}$
10: **end while**
11: $C_u = \min\{2C_u, C_l\}$
12: **end while**

输出： 未标记样本的预测结果: $\hat{\boldsymbol{y}} = (\hat{y}_{l+1}, \hat{y}_{l+2}, \ldots, \hat{y}_{l+u})$

图 13.4 TSVM 算法

3. 涉及巨大计算开销的大规模优化问题。

图半监督学习

1. 数据集映射成图，每个边的强度正比于样本之间的相似度，标记的视为染色，未标记的未染色，半监督学习就是"颜色"在图上扩散或传播的过程。

2. 拉普拉斯矩阵：L = D-A ,D:度矩阵 A：邻接矩阵

3. 多分类问题的标记传播方法

输入： 有标记样本集 $D_l = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_l, y_l)\}$;
未标记样本集 $D_u = \{\boldsymbol{x}_{l+1}, \boldsymbol{x}_{l+2}, \ldots, \boldsymbol{x}_{l+u}\}$;
构图参数 σ;
折中参数 α.

过程：

1: 基于式(13.11)和参数 σ 得到 $\mathbf{W}$;
2: 基于 $\mathbf{W}$ 构造标记传播矩阵 $\mathbf{S} = \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}$;
3: 根据式(13.18)初始化 $\mathbf{F}(0)$;
4: $t = 0$;
5: **repeat**
6: $\mathbf{F}(t+1) = \alpha\mathbf{S}\mathbf{F}(t) + (1-\alpha)\mathbf{Y}$;
7: $t = t + 1$
8: **until** 迭代收敛至 $\mathbf{F}^*$
9: **for** $i = l+1, l+2, \ldots, l+u$ **do**
10: $y_i = \arg\max_{1 \leqslant j \leqslant |\mathcal{Y}|} (\mathbf{F}^*)_{ij}$
11: **end for**

输出： 未标记样本的预测结果: $\hat{\boldsymbol{y}} = (\hat{y}_{l+1}, \hat{y}_{l+2}, \ldots, \hat{y}_{l+u})$

图 13.5 迭代式标记传播算法

4. 缺陷：

- 存储开销大，很难处理大数据
- 仅能考虑训练样本集，难以判知新样本在图中的位置。

基于分歧的方法

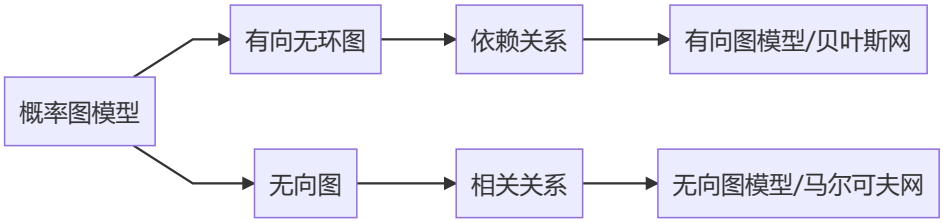
1.协同训练，被看作"多视图学习"：基于标记样本训练分类，每个分类器挑选可能准确率最高的未标记样本赋予伪标记，并将伪标记提供给另一个分类器作为新增的有标记样本用于训练更新……

2.基于分歧的方法只需采用合适的基学习器，就能骄傲少受到模型假设、损失函数非凸性和数据规模的影响。

概率图模型

隐马尔可夫模型

1. **概率图模型**：用图来表达变量相关关系的概率模型。



2. **隐马尔可夫模型（HMM）**：结构最简单的动态贝叶斯网，著名的有向图模型，应用于时序建模、语音识别、自然语言处理。

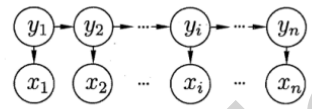
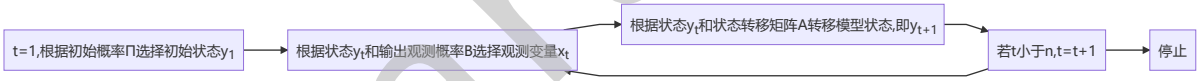


图 14.1 隐马尔可夫模型的图结构

马尔可夫链：系统下一时刻的状态仅由当前状态决定，不依赖于以往的任何状态。

隐马尔可夫模型（图画的有点奇怪）



3.带解决的问题（关注的问题）：

- 评估模型与观测序列之间的匹配程度（依据x_1,……,x_{n-1}推测当前时刻最有可能的观测值x_n）
- 根据观测序列推测出隐藏的模型状态（语音识别，依据语音序号推断最有可能的状态序列-文字）
- 最好的描述观测数据（训练参数，不知道能不能用到BP神经网络上的参数估计？）

马尔可夫随机场(MRF)

1. 势函数（亦称因子）：定义在变量子集上的非负实函数，用于定义概率分布函数
2. 团：任意两结点都有边相连（强连通）；极大团：一个团中加入另外任何一个结点都不再形成团（想到了等价类和最大等价类）

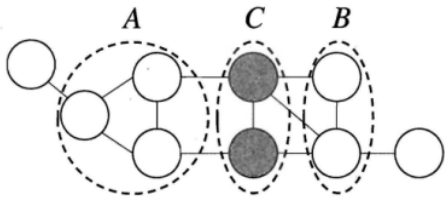
3. 

图 14.3 结点集 A 和 B 被结点集 C 分离

(有点像三部图的定义，但又不一样)

4. 全局马尔可夫性：给定两个子集的分离集，则这两个变量子集条件独立。

- 局部马尔可夫性：给定某变量的邻接变量，则该变量条件独立于其他变量。
- 成对马尔可夫性：给定所有其他变量，两个非邻接变量条件独立

▢ 条件随机场（CRF）

1. 是一种判别式无向图模型（**生成式**模型直接对联合分布进行建模，而**判别式**模型则对条件分布进行建模）
2. 线性条件随机场是只考虑概率图中相邻变量是否满足特征函数的模型。（好像没有特别明白，我再想想）

学习与推断

解决的问题：高效的计算边际分布 （1）计算出精确值，但开销过大；（2）在较低的时间复杂度下计算出近似值

精确值

1. **变量消去法**把多个变量的积的求和问题，转化为对部分变量交替进行求积与求和的问题。（缺点：若需计算多个边际分布，重复使用变量消去法将会造成大量的冗余计算）
2. **信念传播**：将变量消去法中的求和操作看作一个消息传递过程，较好地解决了求解多个边际分布时的重复计算问题。

- 指定一个根结点，从所有叶结点开始向根结点传递消息，直到根结点收到所有邻接结点的消息
- 从根结点开始向叶结点传递消息，直到所有叶结点均收到消息

近似值

1. **MCMC采样**（马尔可夫链蒙特卡罗）：先设法构造一条马尔可夫链，使其收敛至平稳分布恰为待估计参数的后验分布，然后通过这条马尔可夫链来产生符合后验分布的样本，并基于这些样本来进行估计。

MH算法:

输入：先验概率 $Q(\mathbf{x}^* \mid \mathbf{x}^{t-1})$.
过程：
1: 初始化 $\mathbf{x}^0$;
2: **for** $t = 1, 2, \ldots$ **do**
3: 根据 $Q(\mathbf{x}^* \mid \mathbf{x}^{t-1})$ 采样出候选样本 $\mathbf{x}^*$;
4: 根据均匀分布从 $(0, 1)$ 范围内采样出阈值 u;
5: **if** $u \leqslant A(\mathbf{x}^* \mid \mathbf{x}^{t-1})$ **then**
6: $\mathbf{x}^t = \mathbf{x}^*$
7: **else**
8: $\mathbf{x}^t = \mathbf{x}^{t-1}$
9: **end if**
10: **end for**
11: **return** $\mathbf{x}^1, \mathbf{x}^2, \ldots$
输出：采样出的一个样本序列 $\mathbf{x}^1, \mathbf{x}^2, \ldots$

图 14.9 Metropolis-Hastings 算法

2. **变分判断**：使用已知简单分布来逼近需推断的复杂分布，并通过限制近似分布的类型，从而得到一种局部最优、但具有确定解的近似后验分布。

规则学习

基本概念

1. 规则学习（命题和逻辑）的特点和优点：

- 与神经网络、支持向量机这样的“黑箱模型”相比，规则学习具有更好的**可解释性**
- 规则学习能更自然的学习过程中引入领域知识

2. **冲突**——当同一个示例被判别结果不同的多条规则覆盖时，

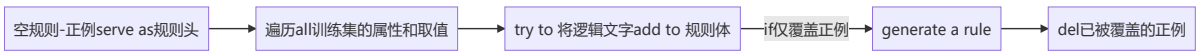
解决冲突——**冲突消散**——投票法、排序法、元规则法等

- **投票法**是将判别相同的规则数最多的结果作为最终结果
- **排序法**是在规则集合上定义一个顺序，在发生冲突时使用排序最前的规则，相应的规则学习过程称为"带序规则" (ordered rule) 学习或"优先级规则" (priority rule) 学习
- **元规则法**是根据领域知识事先设定一些"元规则" (meta-rule)，即关于规则的规则

序列覆盖（分治策略）

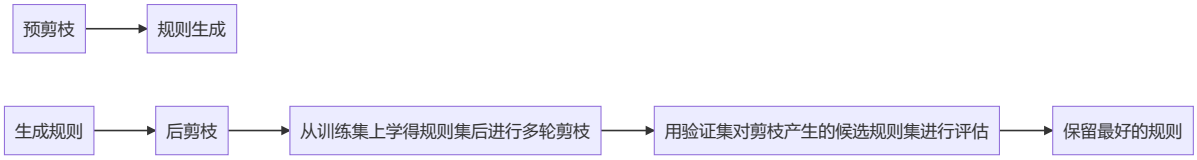
在训练集上每学到一条规则，就将该规则覆盖的训练样例去除，然后以剩下的训练样例组成训练集重复上述过程.

```
while(n--)
```



在现实中，有两种策略产生规则：**生成-测试法**——逐渐特化，自顶而下、**数据驱动法**——逐渐泛化，自底而上，由于前者对噪声的鲁棒性均比后者强，所以前者应用更多。

剪枝优化



RIPPER 的优化策略：:最初生成规则集的时候，规则是**按序**生成的，每条规则都没有对其后产生的规则加以考虑，这样的贪心算法本质常导致算法**陷入局部最优**；RIPPER 的后处理优化过程将R中的**所有规则**放在一起重新加以优化，恰是通过**全局的考虑**来缓解贪心算法的局部性，从而往往能得到更好的效果。

一阶规则学习

e.g. FOIL (Fírst-Order Inductive Learner) 遵循序贯覆盖框架且采用**自顶向下**的规则归纳策略，使用 "FOIL 增益" (FOIL gain)(仅考虑正例的信息量，并且用新规则覆盖的正例数作为权重) 来选择文字，最终生成合适的单条规则加入规则集，之后FOIL 使用后剪枝对规则集进行优化。

归纳逻辑程序设计

归纳逻辑程序设计（ILP） = 一阶规则 + 函数和逻辑表达式

优点（可解决的问题）

- 使机器学习系统具备了强大的表达能力
- 看作用机器学习技术来解决基于背景知识的**逻辑程序归纳**

最小一般泛化（LGG）

归纳逻辑程序设计采用**自底向上**的规则生成策略，直接将一个或多个正例所对应的具体事实作为初始规则，再对规则逐步进行泛化以增加其对样例的覆盖率。（看得有点迷惑，暂时没太理解）

逆归结

1.在逻辑学中，演绎和归纳是两种基本方式，机器学习属于归纳的范畴

2.逆归结：（分子上的 蕴含 分母上的）

吸收(absorption)：

$$\frac{p \leftarrow A \wedge B \quad q \leftarrow A}{p \leftarrow q \wedge B \quad q \leftarrow A}.$$

辨识(identification)：

$$\frac{p \leftarrow A \wedge B \quad p \leftarrow A \wedge q}{q \leftarrow B \quad p \leftarrow A \wedge q}.$$

内构(intra-construction)：

$$\frac{p \leftarrow A \wedge B \quad p \leftarrow A \wedge C}{q \leftarrow B \quad p \leftarrow A \wedge q \quad q \leftarrow C}.$$

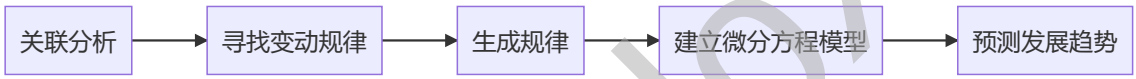
互构(inter-construction)：

$$\frac{p \leftarrow A \wedge B \quad q \leftarrow A \wedge C}{p \leftarrow r \wedge B \quad r \leftarrow A \quad q \leftarrow r \wedge C}.$$

时间序列预测（非神经网络的传统方法）

灰色预测模型

算法思路：灰色预测是一种对含有不确定因素的系统进行预测的方法。其用**等时距**观测到的反映预测对象特征的一系列数量值构造**灰色预测模型**，预测未来某一时刻的特征量，或达到某一特征量的时间。



实现（原理版，好像没有合适的库）：

```
# 灰色模型预测

import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt

path = "C:/Users/华为/Desktop/data1/train.csv"
# 使用pandas读入
data = pd.read_csv(path) #读取文件中所有数据
data = np.array(data)#改成数组

lens = len(data)  # 数据量
for i in range(lens):
    data[i] = (data[i]-0.0002)*1e9  #只考虑后面几位（数据前几位变化不大），这个只和数据有关

# 数据检验
## 计算级比
lambds = []
for i in range(1, lens):
    lambds.append(data[i-1]/data[i])
## 计算区间
x_min = np.e**(-2/(lens+1))
```

```
x_max = np.e**(2/(lens+1))
## 检验
flag = True
for lambd in lambds:
    if (lambd < x_min or lambd > x_max):
        flag = False
if (flag == False):
    print('该数据未通过检验')
else:
    print('该数据通过检验')

# 构建灰色模型GM(1,1)
## 累加数列
data_1 = []
data_1.append(data[0])
for i in range(1, lens):
    data_1.append(data_1[i-1]+data[i])
## 灰导数及临值生成数列
d = []
z = []
for i in range(1, lens):
    d.append(data[i])
    z.append(-1/2*(data_1[i-1]+data_1[i]))
## 求a、b
B = np.array(z).reshape(lens-1,1)
one = np.ones(lens-1)
B = np.c_[B, one]  # 加上一列1
Y = np.array(d).reshape(lens-1,1)
a, b = np.dot(np.dot(np.linalg.inv(np.dot(B.T, B)), B.T), Y)
print('a='+str(a))
print('b='+str(b))

# 预测
def func(k):
    c = b/a
    return (data[0]-c)*(np.e**(-a*k))+c
data_1_hat = []  # 累加预测值
data_0_hat = []  # 原始预测值
data_1_hat.append(func(0))
data_0_hat.append(data_1_hat[0])
for i in range(1, lens+3):  # 多预测3次
    data_1_hat.append(func(i))
    data_0_hat.append(data_1_hat[i]-data_1_hat[i-1])
print('预测值为：')
for i in data_0_hat:
    print(i)

# 模型检验
## 预测结果方差
```

```
data_h = np.array(data_0_hat[0:7]).T
sum_h = data_h.sum()
mean_h = sum_h/lens
S1 = np.sum((data_h-mean_h)**2)/lens
## 残差方差
e = data - data_h
sum_e = e.sum()
mean_e = sum_e/lens
S2 = np.sum((e-mean_e)**2)/lens
## 后验差比
C = S2/S1
## 结果
if (C <= 0.35):
    print('1级，效果好')
elif (C <= 0.5 and C >= 0.35):
    print('2级，效果合格')
elif (C <= 0.65 and C >= 0.5):
    print('3级，效果勉强')
else:
    print('4级，效果不合格')
```

结果：短期的效果不错，适合数据量小的时候，至于具体适合哪一种暂时没有想法，因为有的效果很差

指数平滑法

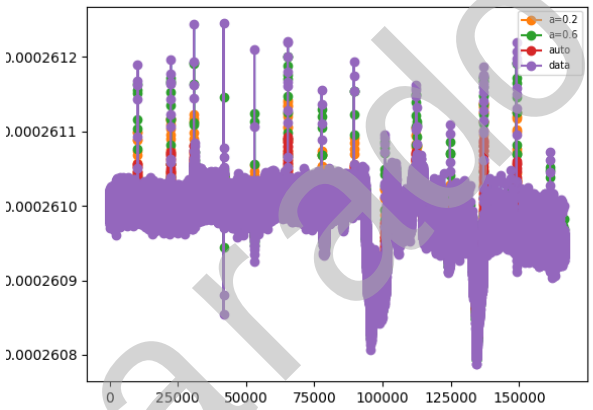
原理：任一期的指数平滑值都是本期实际观察值与前一期指数平滑值的加权平均。

实现：（简单指数平滑）

```
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing,
SimpleExpSmoothing, Holt
import numpy as np
import pandas as pd
path = "C:/Users/华为/Desktop/data1/train -处理版.csv"
# 使用pandas读入
data = pd.read_csv(path) #读取文件中所有数据
data = np.array(data)#改成数组
plt.plot(data)
# Simple Exponential Smoothing
fit1 =
SimpleExpSmoothing(data).fit(smoothing_level=0.2,optimized=False)
# plot
l1, = plt.plot(list(fit1.fittedvalues) + list(fit1.forecast(5)),
marker='o')
```

```
fit2 =
SimpleExpSmoothing(data).fit(smoothing_level=0.6,optimized=False)
# plot
l2, = plt.plot(list(fit2.fittedvalues) + list(fit2.forecast(5)),
marker='o')

fit3 = SimpleExpSmoothing(data).fit() #自动优化，允许statsmodels自动为我们找到优化值
# plot
l3, = plt.plot(list(fit3.fittedvalues) + list(fit3.forecast(5)),
marker='o')

l4, = plt.plot(data, marker='o')
plt.legend(handles = [l1, l2, l3, l4], labels = ['a=0.2', 'a=0.6',
'auto', 'data'], loc = 'best', prop={'size': 7})
plt.show()
```

结果：(emmmm，其实数据是处理过的，但看起来效果不是很好，暂时归结为数据的问题)



算法特点：

1. 指数平滑法进一步加强了观察期近期观察值对预测值的作用，对不同时间的观察值所赋予的**权数不等**，从而加大了近期观察值的权数，使预测值能够迅速反映市场实际的变化。
2. 指数平滑法对于观察值所赋予的权数有**伸缩性**，可以取不同的a 值以改变权数的变化速率。（a越小，则权数变化越快）

ARMA

（实现来自一次数模练习题）

算法前提：**平稳性分析**，如果不平稳要进行差分分析，依据自相关图和偏自相关图判断**拖尾截尾**（python有函数可以直接算出来，SPSS有专家建模器也可以得出，但都无法得到最理想的值）

实现:

```
from statsmodels.graphics.tsaplots import plot_acf, pacf, plot_acf,
plot_pacf
from sktime.forecasting.base import ForecastingHorizon
from sktime.forecasting.arima import ARIMA, AutoARIMA
from statsmodels.tsa.stattools import adfuller
import matplotlib.pyplot as plt
import statsmodels.api as sm
import pandas as pd
import numpy as np
import time
import os

unflat = []

def adf_test(ts):
    # 检测数据平稳性
    adftest = adfuller(ts)
    adf_res = pd.Series(adftest[0:4], index=[
                        'Test Statistic', 'p-value', 'Lags Used',
'Number of Observations Used'])
    for key, value in adftest[4].items():
        adf_res['Critical Value (%s)' % key] = value
    print(adf_res)
    if adf_res['p-value'] <= 0.05:
        print("The DATA is flat.")
        return 1
    else:
        print("The DATA isn't flat, It's need Differencing.")
        return 0

def get_pdq(ts, di):
    # 画自相关、偏自相关图与数据
    r, rac, Q = sm.tsa.acf(ts, qstat=True)
    prac = pacf(ts, method='ywmle')
    table_data = np.c_[range(1, len(r)), r[1:], rac,
prac[1:len(rac)+1], Q]
    table = pd.DataFrame(table_data, columns=[
                        'lag', "AC", "Q", "PAC", "Prob(>Q)"])

    print(table)

    plot_acf(ts)
    # plt.savefig(f"{di}_AC")
    plot_pacf(ts)
    plt.show()
    # plt.savefig(f"{di}_PAC)
```

```
    return

if __name__ == '__main__':
    # pq_file = open("d:/数模/homework/2/pq.txt", "r+")
    # chi = 'f'
    for di in range(1, 1):
        #print(f"Now, It's running the {di}_th data: \nStart!!!")
        start = time.time()

        file = f"C:/Users/华为/Desktop/data1/train筛选版.csv"
        data = pd.read_csv(file)
        x, y = np.split(data.values, indices_or_sections=(1,), axis=1)
        x = [i[0] for i in x]
        a = [i[0] for i in y]
        b = [i[1] for i in y]

        time_series = pd.Series(a)
        time_series.index = pd.date_range(
            x[0], periods=40000, freq="S")
        time_series.plot()
        plt.savefig(f"{di}_t_pa")
        plt.close()

        choi = input("Do you know p and q? <y/n/p>")
        if choi == "n":
            if adf_test(time_series):
                pass
            else:
                unflat.append(di)
                continue
            get_pdq(time_series, di)
            chi = input("Can you figure out p and q?<y/n>")
        elif choi=='y':
            chi = 'y'
            pass
        else:
            continue

        file = f"C:/Users/华为/Desktop/data1/train筛选版.csv"
        data = pd.read_csv(file)
        x, y = np.split(data.values, indices_or_sections=(1,), axis=1)
        x = [i[0] for i in x]
        fh = ForecastingHorizon(
            pd.date_range(x[0], periods=4000, freq="S"),
is_relative=False
        )

        if chi == 'y':
            p = int(input("P: "))
```

```
            q = int(input("Q: "))
            forecaster = ARIMA(order=(p, 0, q), suppress_warnings=True)
            forecaster.fit(time_series)
        elif chi == 'n':
            forecaster = AutoARIMA(start_p=0, max_p=40,
                                   strat_q=0, max_q=40,
suppress_warnings=True)
            forecaster.fit(time_series)
        else:
            #p_q = pq_file.readline()

            p_q = p_q.split()
            p = int(p_q[0])
            q = int(p_q[1])
            print(f"p_q has been read:\n    p: {p}\n    q: {q}")
            forecaster = ARIMA(order=(p, 0, q), suppress_warnings=True)
            forecaster.fit(time_series)

        rel = forecaster.predict(fh)

        f = open(f"{di}_rel.txt", "w")
        for i in rel:
            f.write(str(i)+"\n")
        f.close()

        elapsed = (time.time() - start)
        print("Time used:", elapsed)
        print(f"The {di}th data is over!!!\n\n")

    os.system("python 2.py")
    if len(unflat) != 0:
        print(unflat)
```

下面是**指标评估**的实现：

```
import pandas as pd
import numpy as np

if __name__ == "__main__":
    for di in range(1, 14):
        file = f"d:/数模/homework/2/Data/Datasets2/test12hour_{di}.csv"
        data = pd.read_csv(file)
        x, y = np.split(data.values, indices_or_sections=(1,), axis=1)
        x = [i[0] for i in x]
        a = [i[0] for i in y]

        f = open(f"d:/数模/homework/2/{di}_rel.txt", "r+")
        my_a = f.readlines()
```

```
        my_a = [np.float64(i.strip("\n")) for i in my_a]

        sigma = []
        for i in range(len(a)):
            sigma.append(abs(a[i]-my_a[i])/a[i])

        mape = np.sum(sigma)/len(a)

        mae = np.sum([abs(a[i] - my_a[i]) for i in
range(len(a))])/len(a)

        rel = []
        for i in range(len(a)):
            rel.append(a[i]-my_a[i])
        rms = np.var(rel)

        rel = [abs(i) for i in rel]
        rel.sort()
        rang = abs(rel[0] - rel[-1])

        f = open(f"d:/数模/homework/2/{di}_data.txt", "w")
        f.write(f'''The data of the {di}_th sample:
        MAE: {mae}
        MAPE: {mape}
        RMS: {rms}
        Range: {rang}
        σ: \n''')
        for i in sigma:
            f.write("          "+str(i)+"\n")
```