**作业清单（4/20）**

1. 安装Python 3.X 和 Orange3 软件，是否完成？

是

1. 完成课堂实验（给定教师数据，判断身份的实验），是否完成？

是

1. 复习Numpy的主要功能（按照课堂PPT完成相关实验）
2. 完成参考教材第1章 1.9的第1题和第4题（抄题）

1.1 什么是数据挖掘？在你的回答中，强调以下问题：

（a)它是又一种广告宣传吗？

（b)它是一种从数据库、统计学、机器学习和模式识别发展而来的技术的简单转换或应用吗？

（c)我们提出了一种观点，说数据挖掘是数据库技术进化的结果。你认为数据挖掘也是机器学习研究进化的结果吗？你能基于该学科的发展历史提出这一观点吗？针对统计学和模式识别领域，做相同的事。

（d)当把数据挖掘看做知识发现过程时，描述数据挖掘所涉及的步骤。

|  |
| --- |
| Answer:   1. .不是 2. .不是 3. .数据挖掘是从数据中提取和发现其信息或规律的过程，统计学和机器学习等都是数据挖掘可能使用或借助的方法，数据挖掘过程使用这些方法但是不能与之划等号，机器学习、统计学等领域的发展为数据挖掘提供了更有效地手段，但是数据挖掘并非这些学科的简单加和。 4. 数据清理、数据集成、 数据选择、数据变换、挖掘方法、模式评估、知识表示。 |

1.4 给出一个例子，其中数据挖掘对于工商企业的成功是至关重要的。该工商企业需要什么数据挖掘功能（例如，考虑可以挖掘何种类型的模式）？这种模式能够通过简单的查询处理或统计分析得到吗？

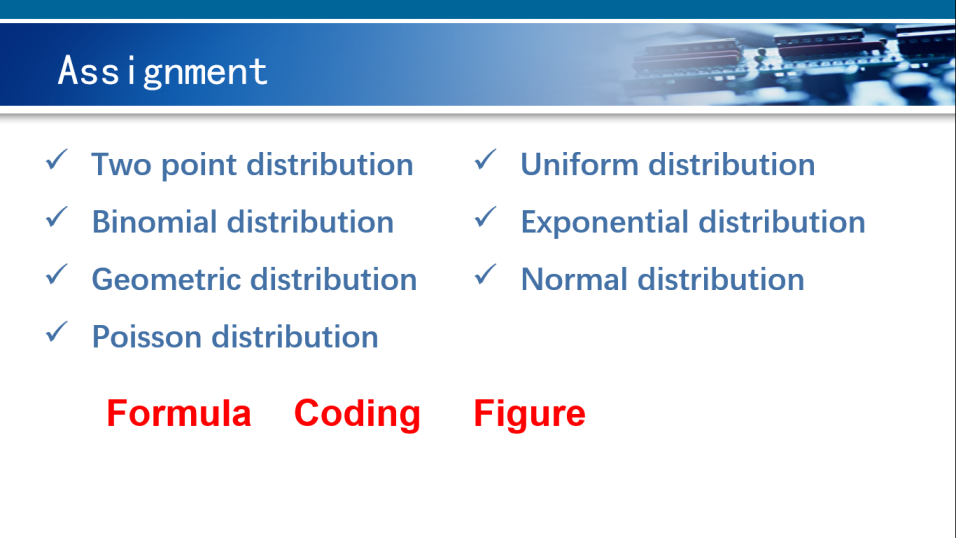
|  |
| --- |
| Answer:  在医学上数据挖掘可以使用一些统计学或机器学习的方法对大量医疗患者的各特征和指标进行建模以便于预测新的病人的病情，而建构这种模型的方法是不能通过简单的查询处理或统计分析得到。 |

**作业清单（4/22）**

1. 继续安装Python 3.X 和 Orange3 软件，是否完成？

是

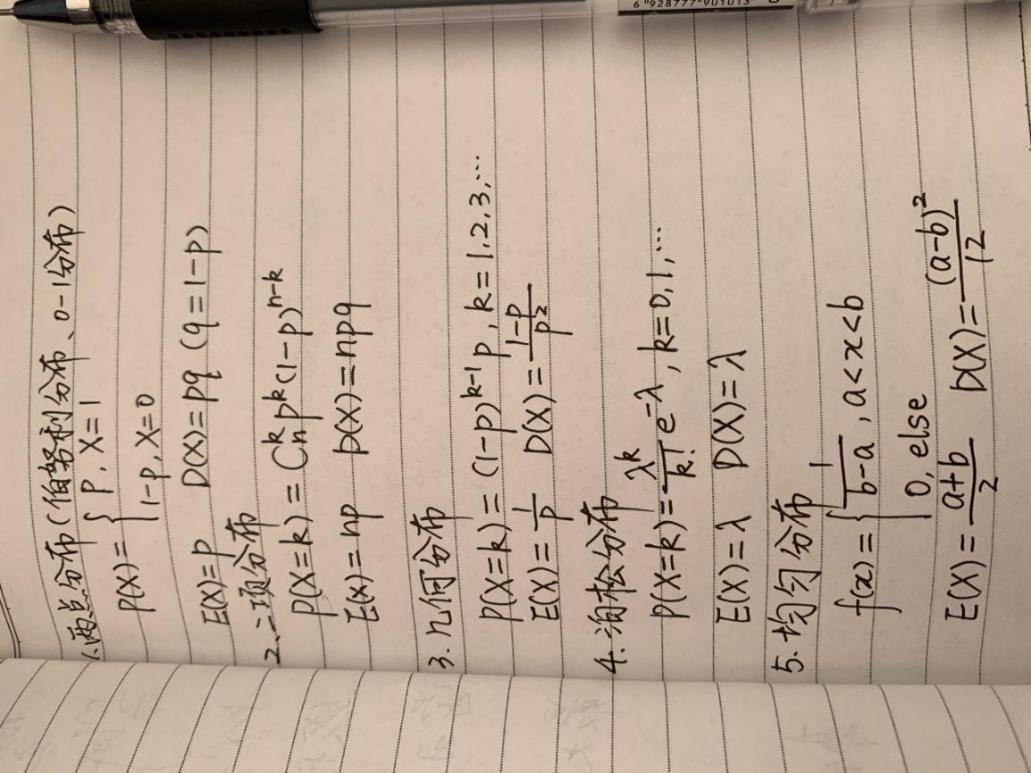
1. 完成常用的概率分布代码，如下图

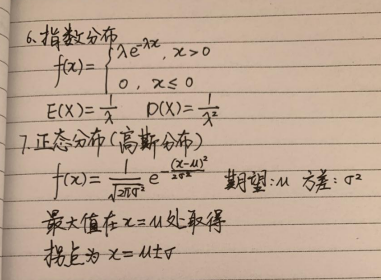


|  |
| --- |
| from scipy.stats import binom *# 导入伯努利分布* import matplotlib.pyplot as plt import numpy as np  *# 这里的伯努利分布指的是n重伯努利实验，即二项分布* n = 10 p = 0.3 x = np.arange(0, n + 1) binomial = binom.pmf(x, n, p)  plt.plot(x, binomial, **'o-'**) plt.title(**'Binomial: n = %i, p=%0.2f'** % (n, p), fontsize=15) plt.xlabel(**'Number of successes'**) plt.ylabel(**'Probability'**, fontsize=15) plt.show() |
| *# -\*- coding:utf-8 -\*-* import numpy as np import matplotlib.pyplot as plt  lambd = 0.5 x = np.arange(0, 15, 0.1) y = lambd \* np.exp(-lambd \* x) plt.plot(x, y) plt.show() |
| import numpy as np import matplotlib.pyplot as plt from scipy.stats import geom  p = 0.5 x = np.arange(1, 11) geometric = geom.pmf(x, p)  plt.plot(x, geometric, **'-o'**) plt.title(**'Geometric distribution'**, fontsize=15) plt.xlabel(**'Number of successes'**) plt.ylabel(**'Probability'**, fontsize=15) plt.show() |
| import numpy as np import matplotlib.pyplot as plt import math  u = 0 sig = math.sqrt(0.2) *# 标准差* x = np.linspace(u - 3 \* sig, u + 3 \* sig, 50) y\_sig = np.exp(-(x - u) \*\* 2 / (2 \* sig \*\* 2)) / (math.sqrt(2 \* math.pi) \* sig)  plt.plot(x, y\_sig, **"r-"**, linewidth=2) plt.grid(True) plt.show() |
| import numpy as np import matplotlib.pyplot as plt  *# Poisson分布* x = np.random.poisson(lam=5, size=10000) *# lam为λ size为k* pillar = 15 a = plt.hist(x, bins=pillar, density=True, range=[0, pillar], color=**'g'**, alpha=0.5) plt.plot(a[1][0:pillar], a[0], **'-o'**) plt.grid() plt.show() |
| import matplotlib.pyplot as plt  *# 随机变量x只能取0,1 我们称X服从以P为参数的（0-1)分布 或两点分布* p = float(1) / 4 x = [0, 1] y = [1 - p, p]  fig = plt.figure() ax = fig.add\_subplot(111) ax.scatter(x, y, label=**'X~B(%s, %s)'** % (1, p))  plt.grid() plt.legend() plt.show() |
| *# 绘图——均匀分布* import numpy as np import scipy.stats as stats import matplotlib.pyplot as plt import matplotlib.style as style  *# PLOTTING CONFIG 绘图配置* style.use(**'fivethirtyeight'**) plt.rcParams[**'figure.figsize'**] = (14, 7) plt.figure(dpi=100)  *# PDF（概率密度函数）* plt.plot(np.linspace(-4, 4, 100), stats.uniform.pdf(np.linspace(-4, 4, 100))) plt.fill\_between(np.linspace(-4, 4, 100), stats.uniform.pdf(np.linspace(-4, 4, 100)), alpha=0.15)  *# CDF（概率累积函数） # plt.plot(np.linspace(-4, 4, 100), stats.uniform.cdf(np.linspace(-4, 4, 100)))  # LEGEND 图例* plt.text(x=-1.5, y=0.7, s=**"pdf(uniform)"**, rotation=65, alpha=0.75, weight=**"bold"**, color=**"#008fd5"**) *# plt.text(x=-0.4, y=0.5, s="cdf", rotation=55, alpha=0.75, weight="bold", color="#fc4f30")* plt.show() |

1. 完成最大似然估计MLE的Python代码

|  |
| --- |
| from scipy.stats import norm import matplotlib.pyplot as plt import numpy as np  **''' norm.cdf 返回对应的累计分布函数值 norm.pdf 返回对应的概率密度函数值 norm.rvs 产生指定参数的随机变量 norm.fit 返回给定数据下，各参数的最大似然估计（MLE）值 '''** x\_norm = norm.rvs(size=200) *# 在这组数据下，正态分布参数的最大似然估计值* x\_mean, x\_std = norm.fit(x\_norm) print(**'mean, '**, x\_mean) print(**'x\_std, '**, x\_std) plt.hist(x\_norm, density=True, bins=15) *# 归一化直方图（用出现频率代替次数），将划分区间变为 20（默认 10）* x = np.linspace(-3, 3, 50) *# 在在(-3,3)之间返回均匀间隔的50个数字。* plt.plot(x, norm.pdf(x), **'r-'**) plt.show() |

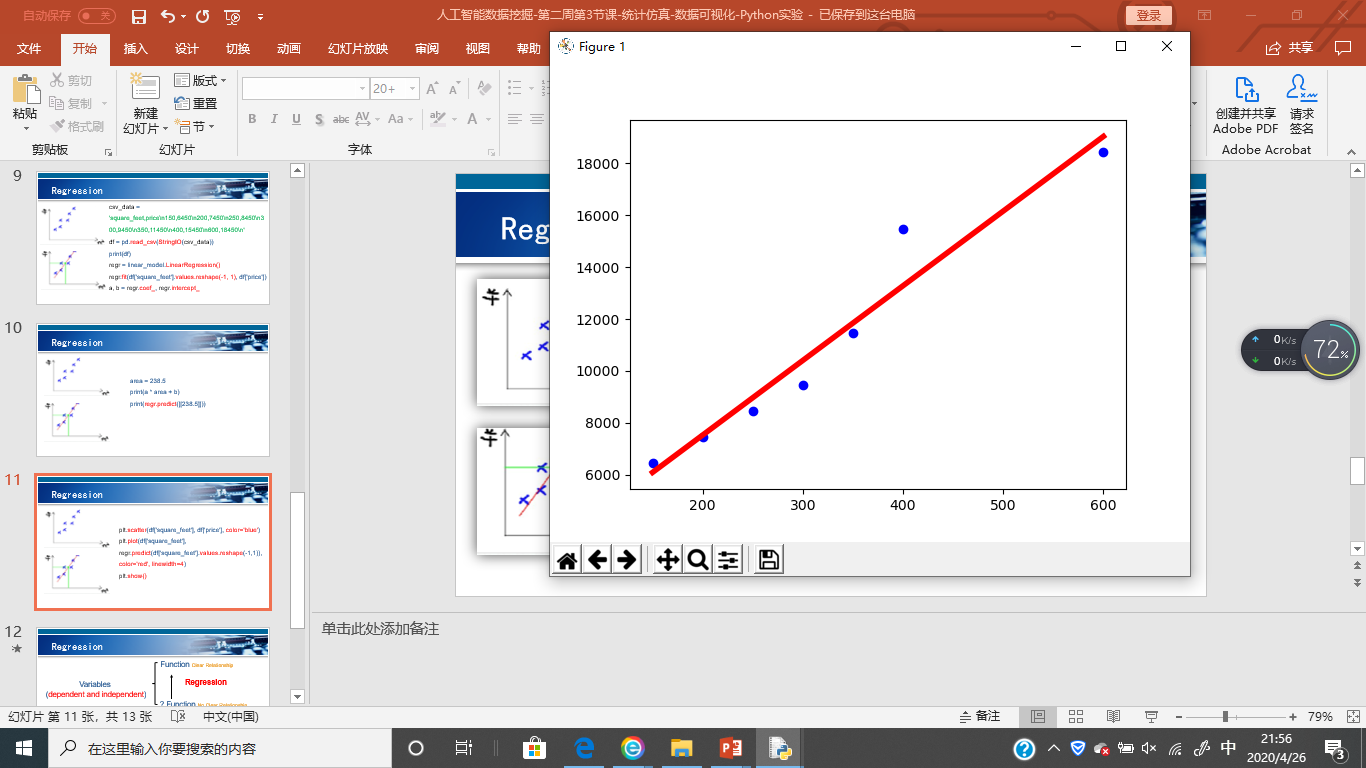




1. DM Lab2 实验结果用orange3挖掘（选做）

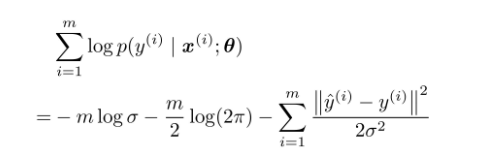
**作业清单（4/27）**

1. 熟悉CSV文件的打开、读取和写入数据。
2. 利用Orange3和Python orange方法完成下图

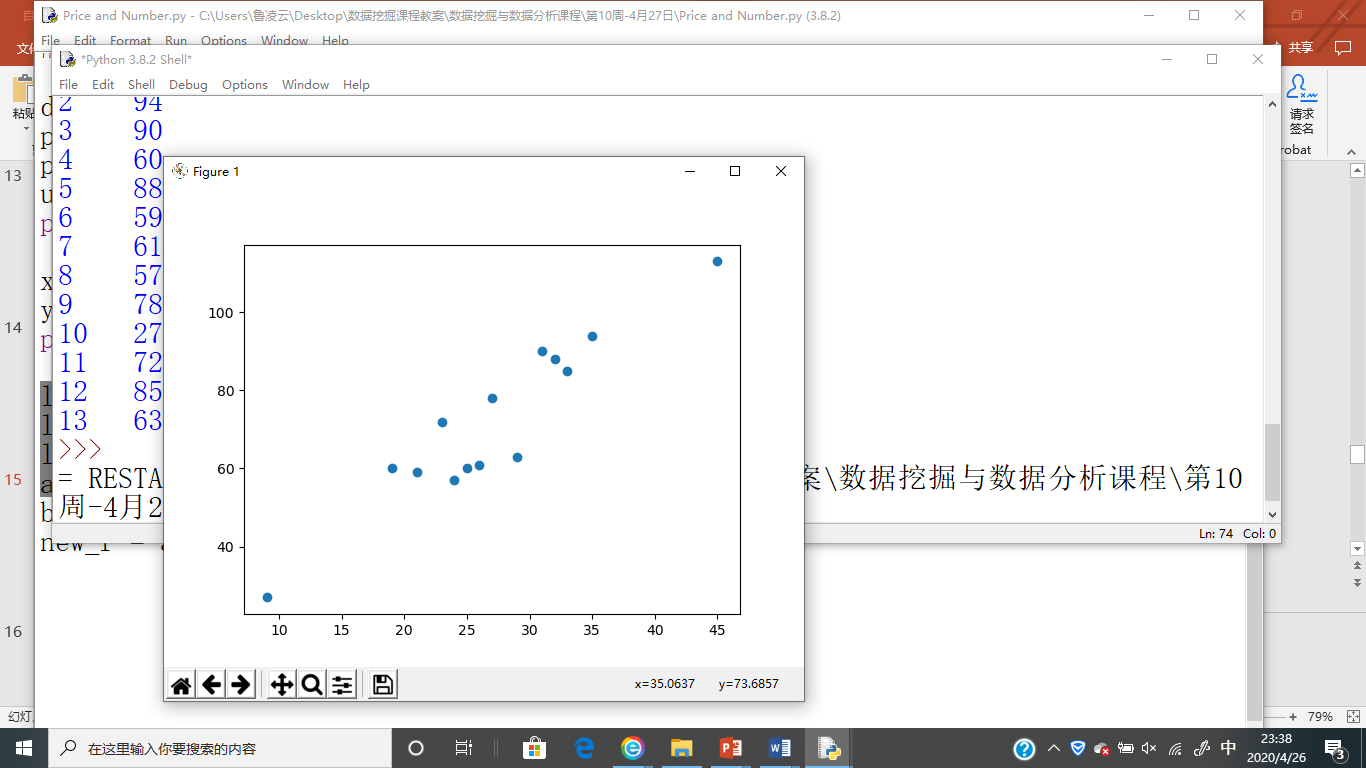


1. 思考最大似然估计MLE和最小二乘之间的关系？

如果将预测值与真实值之间的误差看作服从高斯分布，则最大化对数似然与最小二乘可以看作等价的优化过程，证明如下：



1. 根据DM Lab3数据散点图，画出一元回归线。



|  |
| --- |
| from pandas import read\_csv from matplotlib import pyplot as plt from sklearn import linear\_model  data = read\_csv(**'test1.csv'**)  regr = linear\_model.LinearRegression() regr.fit(data.活动推广费.values.reshape(-1, 1), data.销售额)  plt.scatter(data.活动推广费, data.销售额, color=**'blue'**) plt.plot(data.活动推广费, regr.predict(data.活动推广费.values.reshape(-1, 1)), color=**'red'**, linewidth=4) plt.show() |

1. 根据DM Lab3实验过程，母亲身高167cm，预测孩子身高可能是多少？

|  |
| --- |
| import numpy as np from pandas import read\_csv from matplotlib import pyplot as plt from sklearn import linear\_model  x = np.array([154, 157, 158, 159, 160, 161, 162, 163]) y = np.array([155, 156, 159, 162, 161, 164, 165, 166])  regr = linear\_model.LinearRegression() regr.fit(x.reshape(-1, 1), y) print(regr.predict([[167]]))  Result:171 |

**作业清单（4/29、5/4）**

1. 根据下列数据集（数据表存为csv格式）建立线性回归模型。



1. 预测面积为1000平方英尺的房子价格。

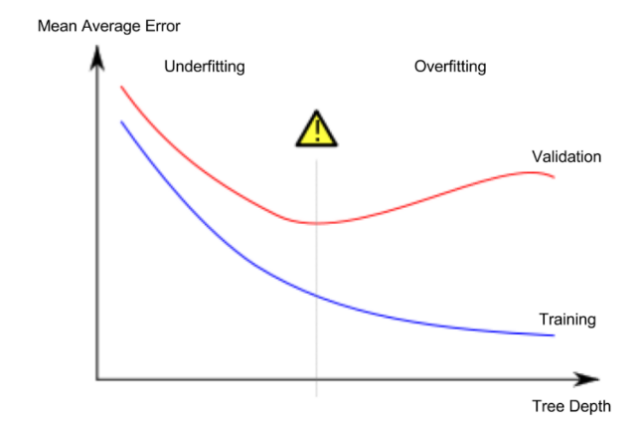
要求: 完成2遍，第1遍可以参考课堂笔记、查阅网络资料等方式完成；第2遍不参考任何辅助方式，限定15分钟内独立编写代码，完成此回归模型。

1. 建立多元回顾模型。至少增加2项房子价格的特征，例如：地段、新旧等因素。
2. 将（1）和（2）整理成实验报告。5月6日上课检查实验报告情况。

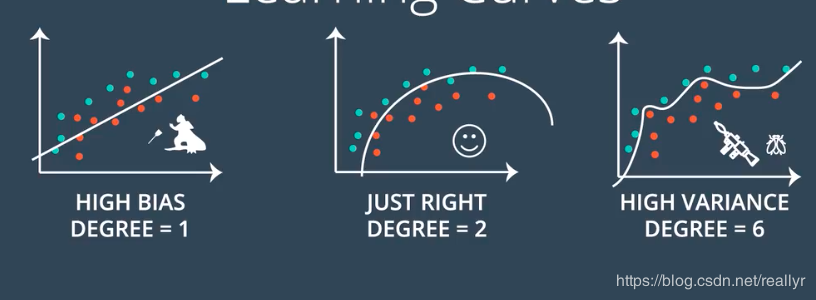
|  |
| --- |
| import pandas as pd *# 读取csv文件* from sklearn import linear\_model *# 线性模型* data = pd.read\_csv(**'PriceData.csv'**) regr = linear\_model.LinearRegression() *# 线性回归模型* regr.fit(data.square\_feet.values.reshape(-1, 1), data.price) print(regr.predict([[1000]])) *# 预测面积为1000时的房价* |
| import pandas as pd *# 读取csv文件* from sklearn import linear\_model *# 线性模型* data = pd.read\_csv(**'PriceData.csv'**) trainData = data.iloc[:, 1:4] *# 取读取数据的2、3、4列作为训练数据，每条训练数据都有三个特征* trainLabel = data.price regr = linear\_model.LinearRegression() regr.fit(trainData, trainLabel) print(regr.predict([[1200, 720, 700]])) *# 预测特征为[[1200, 720, 700]]时的房价* |

实验报告链接：<https://www.jianshu.com/p/2c476af4f2a9>

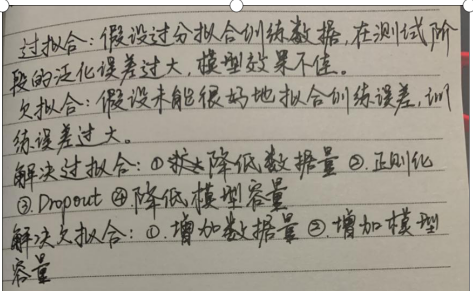
1. 结合下图(a)和(b)解释什么是过拟合和欠拟合？通常用什么方法解决这两个问题？



(a)



(b)



1. 预测鲍鱼的年龄。网上下载“鲍鱼数据集”（见微信群，鲍鱼数据集.csv），建立线性回归模型，指出简单线性回归模型进行预测的问题，思考如何解决？

|  |
| --- |
| import pandas as pd *# 读取csv文件* from sklearn import linear\_model *# 线性模型* from sklearn.model\_selection import train\_test\_split  data = pd.read\_csv(**'abalone.csv'**) trainData = data.iloc[:, :-1] trainLabel = data.rings sexMapping = {  **'F'**: 0.1,  **'M'**: 0.5,  **'I'**: 0.9 } trainData[**'sex'**] = trainData[**'sex'**].map(sexMapping) X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(trainData, trainLabel, train\_size=.80) regr = linear\_model.LinearRegression() regr.fit(X\_train, Y\_train) pre = regr.predict(X\_test).astype(**'int'**) ytest = Y\_test.values  pre\_train = regr.predict(X\_train).astype(**'int'**) ytrain = Y\_train.values  loss\_train\_sum = 0 for i in range(len(pre\_train)):  loss\_train\_sum += pow(pre\_train[i] - ytrain[i], 2) loss\_train\_avg = loss\_train\_sum / len(pre\_train)  loss\_test\_sum = 0 for i in range(len(pre)):  loss\_test\_sum += pow(pre[i] - ytest[i], 2) loss\_test\_avg = loss\_test\_sum / len(pre)  print(**'训练均方误差:'**, loss\_train\_avg) print(**'测试均方误差:'**, loss\_test\_avg)  问题及解决方案：使用线性回归必须使用实数数据作为输入，而鲍鱼数据集中含有字符数据，因此需要将字符映射为实数然后进行线性回归。 |

1. Python实现新型冠状病毒传播模型及预测（选做）

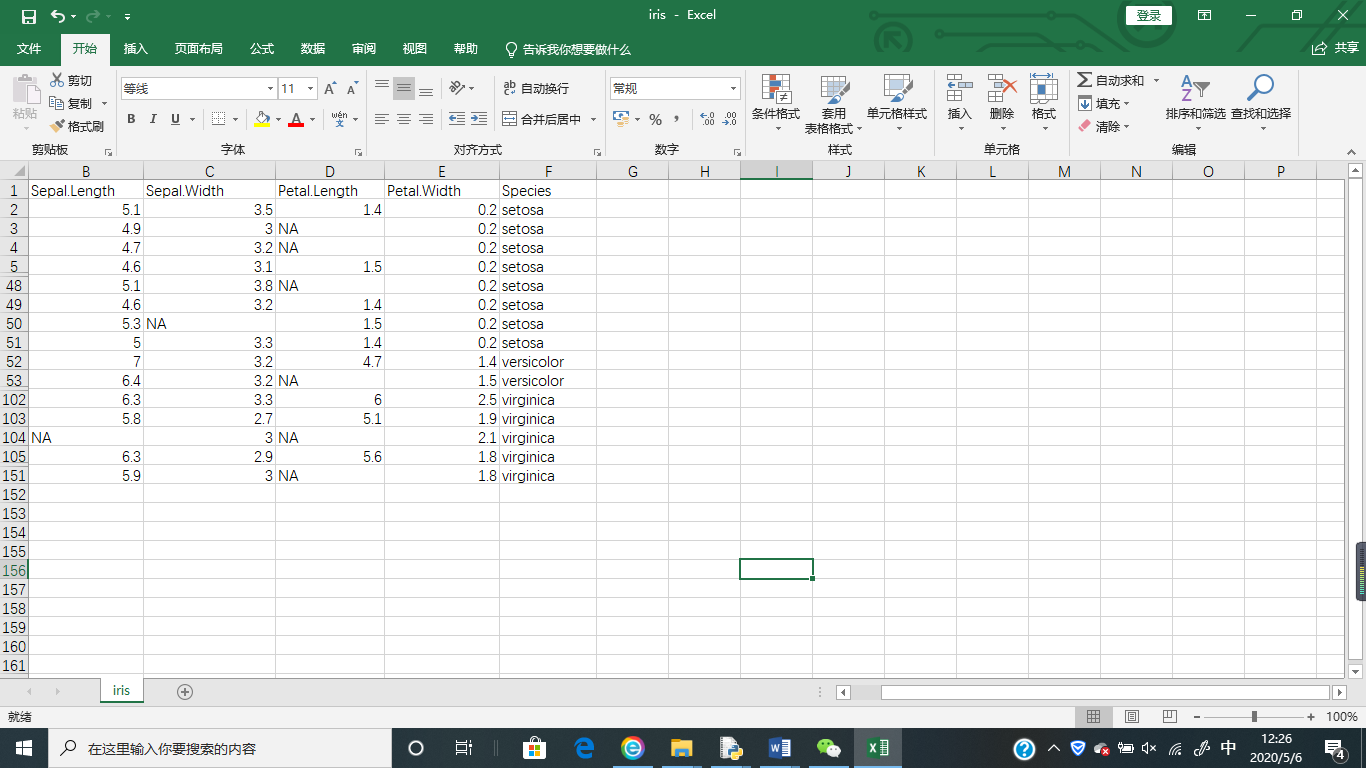
**作业清单（5/6）**

1. 数据清洗是数据挖掘模型建立过程中很重要的一步吗一般，清洗的方法包括什么？

是重要的一步

方法包括：删除缺失值、均值填补法、热卡填补法、决定填补法、回归填补法、多重填补方法、K-最近邻法、有序最近邻法、基于贝叶斯的方法。

1. 对下图的数据采用删除和填补两种方法进行清洗。



|  |
| --- |
| import pandas as pddata = pd.read\_csv(**'1.csv'**)  print(data.isnull().sum()) print(data.dropna(how=**'all'**)) print(data.dropna(axis=1)) print(data.fillna(0)) print(data.fillna(data.mean())) print(data.fillna(data.median())) print(data.fillna(data.mode())) print(data.fillna(method=**'pad'**)) |

1. 对【2】题数据采用回归方法填补。

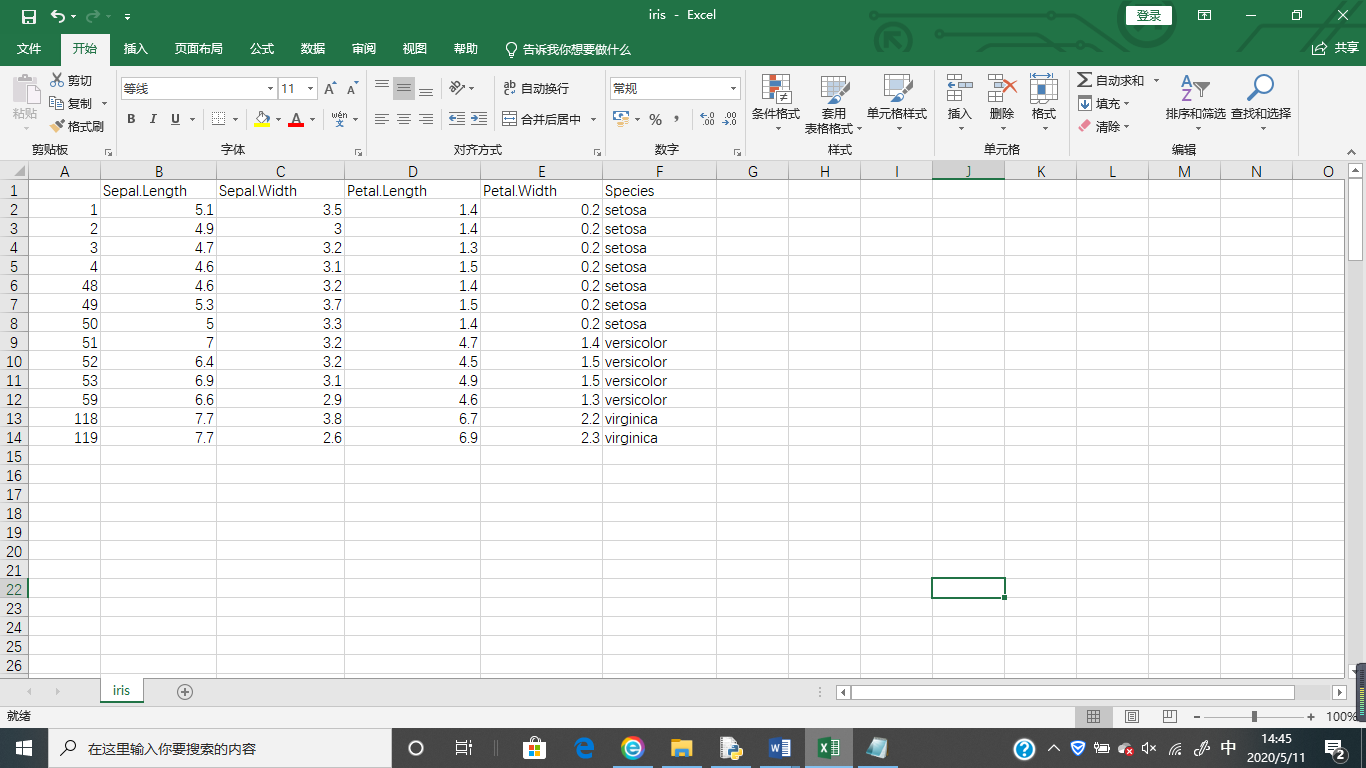
|  |
| --- |
| import pandas as pd from sklearn import linear\_model *# 线性模型* data = pd.read\_csv(**'1.csv'**)newData = data.fillna(data.mean()) trainData = newData.iloc[:, 0:2] *# 取读取数据的2、3、4列作为训练数据，每条训练数据都有三个特征* trainLabel = newData[**"Petal Length"**] regr = linear\_model.LinearRegression() regr.fit(trainData, trainLabel) print(regr.predict([[4.9, 3]])) |

**作业清单（5/11）**

1. Pandas Series是什么? Pandas中的DataFrame是什么？如何将numpy数据转成DataFrame格式的数据？如何将Series数据转成DataFrame格式的数据？如何将DataFrame转换为NumPy数组？如何对DataFrame进行排序？什么是数据聚合？（注：每一小问，举例说明）

|  |
| --- |
| *# Series (Series)是能够保存任何类型的数据(整数，字符串，浮点数，Python对象等)的一维标记数组。轴标签统称为索引。 # DataFrame是一种表格型数据结构，它含有一组有序的列，每列可以是不同的值。DataFrame既有行索引，也有列索引，它可以看作是由Series组成的字典，不过这些Series公用一个索引。* import numpy as np import pandas as pd  *# array --> Series* arr = np.array([4, 2, 3, 1]) ser = pd.Series(arr, index=[**'a'**, **'b'**, **'c'**, **'d'**]) print(ser)  *# Series --> DataFrame* df1 = pd.DataFrame(ser) print(df1)  data = {**'data1'**: ser.values, **'data2'**: [**'aa'**, **'bb'**, **'cc'**, **'dd'**]} df2 = pd.DataFrame(data, index=[**'a'**, **'b'**, **'c'**, **'d'**]) print(df2)  *# DataFrame --> array* arr2 = np.array(df2.values) print(arr2)  *# DataFrame排序* print(df2.sort\_values(by=**'data1'**, ascending=False))  *# 数据聚合指使用基于多组观测结果的总结的统计替换多组观测结果* |

1. 利用iris.csv数据集，建立KNN模型，预测Sepal.Length\Sepal.Width\Petal.Length \Petal.Width分别为(6.3，3.1，4.8，1.4)时，属于鸢尾花的哪个类别？编写KNN源代码。



|  |
| --- |
| from sklearn.neighbors import KNeighborsClassifier import pandas as pd  data = pd.read\_csv(**'1.csv'**)  SpeciesMap = {  **'setosa'**: 1,  **'versicolor'**: 2,  **'virginica'**: 3 } data[**'Species'**] = data[**'Species'**].map(SpeciesMap)  X = data.iloc[:, :4].values y = data.Species.values  print(**'X.shape:'**, X.shape) print(**'y.shape:'**, y.shape)  *# 模型训练和测试* knn\_clf = KNeighborsClassifier(n\_neighbors=2) knn\_clf.fit(X, y)  pre = knn\_clf.predict([[6.3, 3.1, 4.8, 1.4]]) *# 预测* print(**'predict:'**, pre) |

1. 计算X = [1,2,3]和Y = [0,1,2]的曼哈顿距离(Manhattan Distance)，切比雪夫距离 ，闵可夫斯基距离，标准化欧氏距离，马氏距离。给出计算公式，并根据公式计算。利用Python实现上述距离。.

|  |
| --- |
| import numpy as np  X = np.array([1, 2, 3]) Y = np.array([0, 1, 2])  *# 曼哈顿距离:sumi(|xi-yi|)* ManhattanDistance = np.sum(np.abs(X - Y)) print(**'ManhattanDistance:'**, ManhattanDistance)  *# 切比雪夫距离:maxi(|xi-yi|)* ChebyshevDistance = np.max(np.abs(X - Y)) print(**'ChebyshevDistance:'**, ChebyshevDistance)  *# 闵可夫斯基距离:sumi((xi-yi)^p) \*\* (1/p) # 注意闵可夫斯基距离的一些特例: # p = 1:曼哈顿距离 # p = 2:欧式距离 # p = ∞:切比雪夫距离* p = 5 MinkowskiDistance = np.sum((X - Y) \*\* p) \*\* (1 / p) print(**'MinkowskiDistance:'**, MinkowskiDistance)  *# 标准化欧氏距离:sumi(((xi-yi) / si)^2) \*\* 0.5 si为第分量的标准差* mtx = np.vstack([X, Y])  sk = np.var(mtx, axis=0, ddof=1) SED1 = np.sqrt(((X - Y) \*\* 2 / sk).sum()) print(**'Standardized Euclidean distance:'**, SED1, **'(方法一:根据公式求解)'**)  from scipy.spatial.distance import pdist  SED2 = pdist(mtx, **'seuclidean'**) print(**'Standardized Euclidean distance:'**, SED2[0], **'(方法二:根据scipy库求解)'**)  *# 马氏距离 D(Xi,Xj)=sqrt(dot(dot((Xi-Xj).T,SI),(Xi-Xj))) # 马氏距离要求样本数要大于维数，否则无法求协方差矩阵* print() X = np.array([[3, 5, 2, 8],  [4, 6, 2, 4]]) *# 总计10个样本，每个样本2维* XT = X.T  S = np.cov(X) *# 两个维度之间协方差矩阵* SI = np.linalg.inv(S) *# 协方差矩阵的逆矩阵* n = XT.shape[0] MahalanobisDistance1 = [] for i in range(0, n):  for j in range(i + 1, n):  delta = XT[i] - XT[j]  d = np.sqrt(np.dot(np.dot(delta, SI), delta.T))  MahalanobisDistance1.append(d)  print(**'MahalanobisDistance:'**, MahalanobisDistance1, **'(方法一:根据公式求解)'**)  from scipy.spatial.distance import pdist  MahalanobisDistance2 = pdist(XT, **'mahalanobis'**) print(**'MahalanobisDistance:'**, MahalanobisDistance2, **'(方法二:根据scipy库求解)'**) |

**作业清单（5/13）**

【1】选择4名同学A、B、C、D，两次小测成绩，利用Kmeans算法分为“优秀”和“及格”两类。@注意：不能直接调用sklearn第三方库的KMeans函数，根据课堂讲授的分类过程，编写代码。撰写实验报告。

|  |  |  |
| --- | --- | --- |
| 学生姓名 | 小测1 | 小测2 |
| A | 1 | 1 |
| B | 2 | 1 |
| C | 4 | 3 |
| D | 5 | 4 |
| 我写的KMeans算法具有一定的通用性，因此1、2小题使用同样的程序进行求解，因此1、2小题的代码同时在第2小题下面展示。 | | | | |

【2】根据下列成绩单，将5名同学成绩归为A类、B类、C类，利用Kmeans算法实现。@注意：不能直接调用sklearn第三方库的KMeans函数，根据课堂讲授的分类过程，编写代码。撰写实验报告。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 学生姓名 | 小测1 | 小测2 | 小测3 | 期末成绩 | 项目答辩 | 成绩 |
| 张三 | 12 | 15 | 13 | 28 | 24 | ？ |
| 李四 | 7 | 11 | 10 | 19 | 21 | ？ |
| 王五 | 12 | 14 | 11 | 27 | 23 | ？ |
| 赵六 | 6 | 7 | 4 | 13 | 20 | ？ |
| 刘七 | 13 | 14 | 13 | 27 | 25 | ？ |

|  |
| --- |
| import numpy as np import pandas as pd   *# p:样本点维度 # n:样本点个数 # k:聚类中心个数* def final\_classify(train\_data, crowds):  p = train\_data.shape[1]  n = len(train\_data)  k = len(crowds)   new\_crowds = crowds  clsy = np.ndarray((n,))  new\_clsy = np.ndarray((n,))  while (clsy != new\_clsy).any():  clsy = new\_clsy  new\_clsy = classify(train\_data, new\_crowds)  print(**'new\_clsy:'**, new\_clsy)  new\_crowds = []  clusters = [] *# 每一个聚类中的样本点的索引* for i in range(k):  clusters.append([])  for i in range(n):  clusters[new\_clsy[i]].append(i)  for j in range(k):  if len(clusters[j]) == 0:  new\_crowds.append(crowds[j])  else:  sums = np.zeros((p,))  for m in clusters[j]:  sums += train\_data[m]  means = sums / len(clusters[j])  new\_crowds.append(means)   return (new\_crowds, new\_clsy)   *# 将样本点分类到最近的聚类中心，其维度为(n,)* def classify(train\_data, crowds):  all\_distances = get\_distances(train\_data, crowds)  clsy = np.argmin(all\_distances, axis=0)  return clsy   *# 返回所有样本点到所有聚类中心的欧氏距离，其维度为(k,n)* def get\_distances(train\_data, crowds):  all\_distances = [] *# 保存所有样本点到所有聚类中心的欧氏距离，其维度为(k,n)* for i in range(len(crowds)):  distances = [] *# 保存所有样本点到一个聚类中心的欧氏距离，其维度为(n,)* for j in range(len(train\_data)):  distances.append(get\_euclidean\_distance(train\_data[j], crowds[i]))  all\_distances.append(distances)  return all\_distances   *# 返回两点之间的欧氏距离，其中point1、point2为两个点的坐标，其维度为(p,)* def get\_euclidean\_distance(point1, point2):  return (np.sum((point1 - point2) \*\* 2)) \*\* 0.5   *# 返回一个bool值，表示分类结果是否改变* def clsy\_change(new\_clsy, clsy):  changed = False  for i in range(len(clsy)):  if clsy[i] != new\_clsy[i]:  changed = True  break  return changed   print(**'===========Problem1==========='**) crowds1 = np.array([[1, 1], [2, 1]]) dataCsv1 = **'p1.csv'** data1 = pd.read\_csv(dataCsv1)  train\_data1 = data1.iloc[:, 1:].values  result1 = final\_classify(train\_data1, crowds1)  print(**'聚类中心:'**, np.array(result1[0])) print(**'聚类结果:'**, np.array(result1[1])) print()  print(**'===========Problem2==========='**)  *# 初始聚类中心* crowds2 = np.array([[12, 15, 13, 28, 24], [7, 11, 10, 19, 21], [6, 7, 4, 13, 20]]) dataCsv2 = **'p2.csv'** data2 = pd.read\_csv(dataCsv2)  train\_data2 = data2.iloc[:, 1:].values result2 = final\_classify(train\_data2, crowds2)  print(**'聚类中心:'**, np.array(result2[0])) print(**'聚类结果:'**, np.array(result2[1]) |

实验报告链接：<https://www.jianshu.com/p/4def09a719c6>

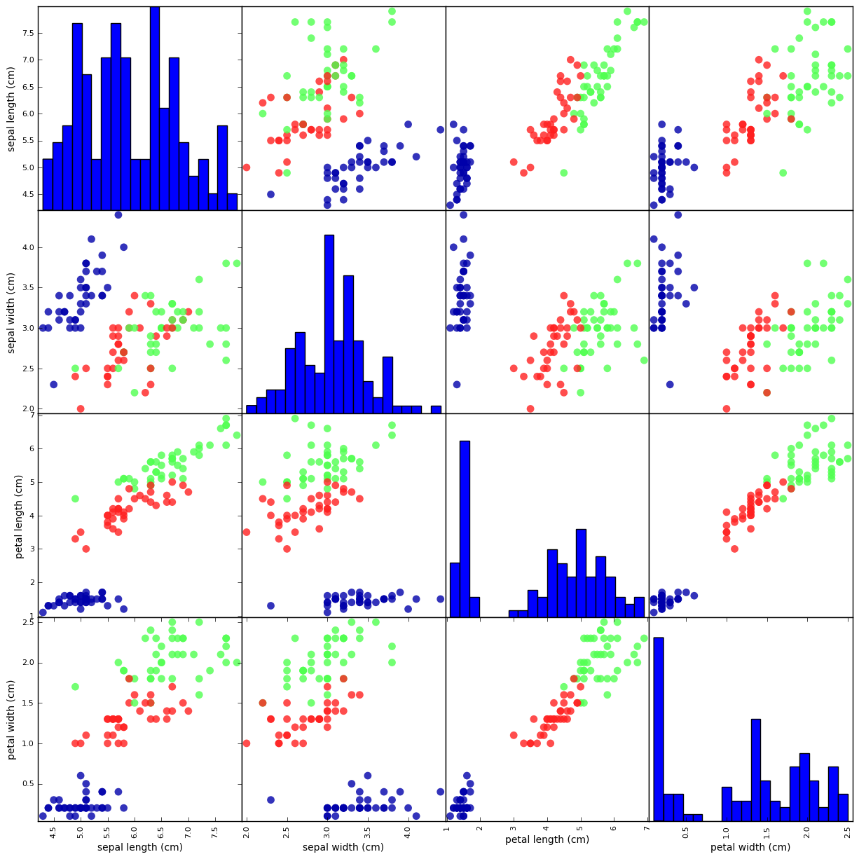
【3】 利用Sklearn的标准KNN和KMeans方法，数据集为“wine.csv”（见微信群），通过KNN算法，对葡萄酒的测试集进行标注，然后对比预测标签值和已知标签值，得到KNN算法的预测准确率。通过Kmeans算法，对无标签的“wine.csv”进行分类，自己设定K值和初始中心点值。

|  |
| --- |
| from sklearn.neighbors import KNeighborsClassifier from sklearn.model\_selection import train\_test\_split from sklearn.cluster import KMeans from sklearn import metrics import pandas as pd import numpy as np  dataset = pd.read\_csv(**'wine.csv'**) dataset[**'Class'**] = dataset[**'Class'**].map({**'one'**: 1, **'two'**: 2, **'three'**: 3}) data = dataset.iloc[:, :-1] label = dataset.Class  X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, label, test\_size=0.2)  knn\_clf = KNeighborsClassifier(n\_neighbors=3) knn\_clf.fit(X\_train, y\_train)  print(**'KNN score:'**, knn\_clf.score(X\_test, y\_test))  y\_pred = KMeans(n\_clusters=3,  init=np.array([[14.23, 1.71, 2.43, 15.6, 127, 2.8, 3.06, .28, 2.29, 5.64, 1.04, 3.92, 1065],  [12.37, .94, 1.36, 10.6, 88, 1.98, .57, .28, .42, 1.95, 1.05, 1.82, 520],  [13.71, 5.65, 2.45, 20.5, 95, 1.68, .61, .52, 1.06, 7.7, .64, 1.74, 740]])).fit\_predict(  data)  print(**'KMeans score:'**, metrics.calinski\_harabasz\_score(data, y\_pred)) |

1. 利用KMeans算法对“iris.csv”数据集的无标签数据分为3类，用三维图形可视化分类结果。

|  |
| --- |
| from sklearn.cluster import KMeans import pandas as pd import matplotlib.pyplot as plt from mpl\_toolkits.mplot3d import Axes3D  dataset = pd.read\_csv(**'iris.csv'**)  data = dataset.iloc[:, 1:4]  y\_pred = KMeans(n\_clusters=3, init=**'random'**).fit\_predict(data)  ax = plt.subplot(111, projection=**'3d'**)  mark = [**'r'**, **'b'**, **'g'**]  for i in range(len(y\_pred)):  ax.scatter(data.values[i][0], data.values[i][1], data.values[i][2], c=mark[y\_pred[i]])  plt.show() |

1. 利用KMeans算法对“iris.csv”数据集的无标签数据分为3类，任取2个特征值，显示分类结果，用二维图形可视化分类结果，类似下图。



|  |
| --- |
| from sklearn.cluster import KMeans import pandas as pd import matplotlib.pyplot as plt  dataset = pd.read\_csv(**'iris.csv'**)  for i in range(4):  for j in range(4):  if i != j:  data = dataset.iloc[:, [i + 1, j + 1]]  y\_pred = KMeans(n\_clusters=3, init=**'k-means++'**).fit\_predict(data)  loc = i \* 4 + (j + 1)  plt.subplot(4, 4, loc)  mark = [**'r'**, **'b'**, **'g'**]  for k in range(len(y\_pred)):  plt.scatter(data.values[k][0], data.values[k][1], c=mark[y\_pred[k]], s=5)  else:  data = dataset.iloc[:, i + 1]  loc = i \* 4 + (j + 1)  plt.subplot(4, 4, loc)  plt.hist(data, 20)  plt.show() |

【6】你认为KMeans算法和KNN算法的缺陷是什么？针对这些缺点，通过查阅资料，了解到有什么改进的方法？

# **KNN**

## **KNN的缺陷**

1. 计算量大,计算复杂度高
2. k值难以确定
3. 对于不平衡的样本比较敏感

## **解决方案**

1. 浓缩训练样本集
2. 使用多个k值，挑选最好的
3. 采用权值的方式（增大距离小的样本的权值）

# **KMeans**

## **KMeans的缺陷**

1. 对于离群点和孤立点敏感
2. k值选择
3. 初始聚类中心的选择
4. 只能发现球状簇

## **解决方案**

1. 使用离群点检测的LOF算法，去除离群点后再聚类
2. 手肘法
3. 选择批次距离尽可能远的K个点

首先随机选择一个点作为第一个初始类簇中心点，然后选择距离该点最远的那个点作为第二个初始类簇中心点，然后再选择距离前两个点的最近距离最大的点作为第三个初始类簇的中心点，以此类推，直至选出K个初始类簇中心点。

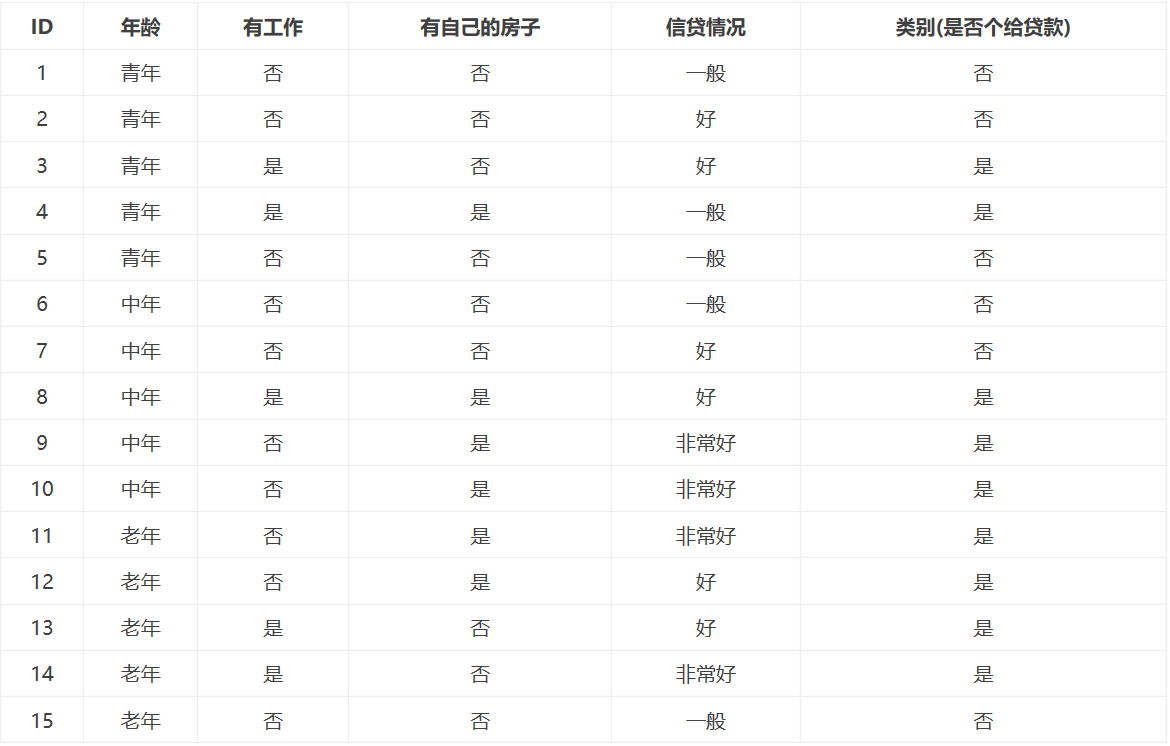
**作业清单（5/20）**

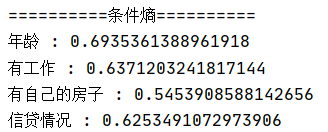
1. 数据集如下图所示，根据我们对决策树的理解，设计一棵决策树，并输入{Age:36,Salary:H,STU:No,Credit:OK}测试数据，是否与预期结果一致？@注意，不允许直接调用Sklearn提供的决策树方法。

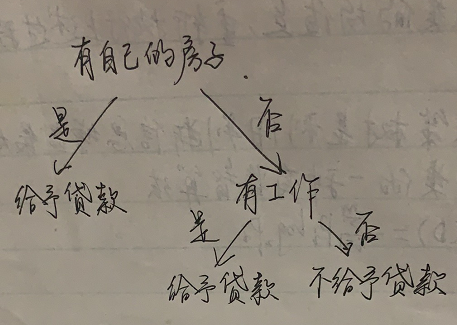
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Age** | **Salary** | **STU** | **Credit** | **Buy Computer** |
| <30 | H | No | OK | No |
| <30 | H | No | Good | No |
| 30-40 | H | No | OK | Yes |
| >40 | M | No | OK | Yes |
| >40 | L | Yes | OK | Yes |
| >40 | L | Yes | Good | No |
| 30-40 | L | Yes | Good | Yes |
| <30 | M | No | OK | No |
| <30 | L | Yes | OK | Yes |
| >40 | M | Yes | OK | Yes |
| <30 | M | Yes | Good | Yes |
| 30-40 | M | No | Good | Yes |
| 30-40 | H | Yes | OK | Yes |
| >40 | M | No | Good | No |

|  |
| --- |
| import pandas as pd import math   *# 获取一个DataFrame中某一个feature下的所有类别、每个类别的数量、每个类别的索引* def get\_all\_classes(dataFrame, feature):  df\_group = dataFrame.groupby(by=feature)  classes = list(df\_group.groups.keys())  num\_classes = []  groups = []  for i in classes:  num\_classes.append(len(df\_group.get\_group(name=i)))  groups.append(df\_group.get\_group(name=i).index)  return classes, num\_classes, groups   *# 计算某一个DataFrame中某个feature的信息熵* def calculate\_info\_entropy(dataFrame, feature):  total\_num = len(dataFrame)  classes, num\_classes, groups = get\_all\_classes(dataFrame, feature)  info\_entropy = 0  for i in range(len(classes)):  p = num\_classes[i] / total\_num  info\_entropy += - p \* math.log(p, 2)  return info\_entropy   *# 计算feature2关于feature1的条件熵* def calculate\_conditional\_entropy(dataFrame, feature1, feature2):  total\_num = len(dataFrame)  classes, num\_classes, groups = get\_all\_classes(dataFrame, feature1)  conditional\_entropy = 0  for i in range(len(classes)):  p = num\_classes[i] / total\_num  info\_entropy = calculate\_info\_entropy(dataFrame.loc[groups[i], :], feature2)  conditional\_entropy += p \* info\_entropy  return conditional\_entropy   *# 计算feature2关于feature1的信息增益* def calculate\_KL\_divergence(dataFrame, feature1, feature2):  info\_entropy = calculate\_info\_entropy(dataFrame, feature2)  conditional\_entropy = calculate\_conditional\_entropy(dataFrame, feature1, feature2)  return info\_entropy - conditional\_entropy   *# 建树* def create\_Decision\_tree(dataFrame):  features = dataFrame.columns  KL\_divergences = []  info\_entropy = calculate\_info\_entropy(dataFrame, features[-1])  for i in range(len(features) - 1):  KL\_divergence = calculate\_KL\_divergence(dataFrame, features[i], features[-1])   if KL\_divergence == info\_entropy:  classes, num\_classes, groups = get\_all\_classes(dataFrame, features[i])  decision\_tree = {}  for k in range(len(groups)):  decision\_tree[features[i] + **' '** + classes[k]] = dataFrame.loc[groups[k][0]][-1]  return decision\_tree  else:  KL\_divergences.append(calculate\_KL\_divergence(dataFrame, features[i], features[-1]))  most\_gain = KL\_divergences.index(max(KL\_divergences))  classes, num\_classes, groups = get\_all\_classes(dataFrame, features[most\_gain])  decision\_tree = {}  for i in range(len(groups)):  index = [j for j in range(len(features))]  index.pop(most\_gain)  if calculate\_info\_entropy(dataFrame.iloc[groups[i], index], features[-1]) == 0:  decision\_tree[features[most\_gain] + **' '** + classes[i]] = dataFrame.loc[groups[i][0]][-1]  else:  decision\_tree[features[most\_gain] + **' '** + classes[i]] = create\_Decision\_tree(  dataFrame.iloc[groups[i], index])  return decision\_tree   *# 使用决策树进行分类* def judge(decision\_tree, data):  if type(data[**'Age'**]) == int:  if data[**'Age'**] < 30:  data[**'Age'**] = **'<30'** elif data[**'Age'**] > 40:  data[**'Age'**] = **'>40'** else:  data[**'Age'**] = **'30-40'** for i in decision\_tree.keys():  if data[i.split()[0]] == i.split()[1]:  if type(decision\_tree[i]) == str:  return decision\_tree[i]  else:  return judge(decision\_tree[i], data)   df = pd.read\_csv(**'p1.csv'**)  decision\_tree = create\_Decision\_tree(df) result = judge(decision\_tree, {**'Age'**: 36, **'Salary'**: **'H'**, **'STU'**: **'No'**, **'Credit'**: **'OK'**}) print(result) |

【2】数据集如下图所示，计算这棵决策树的类别信息熵，并计算基于每个特征值的类别信息熵。根据公式计算每个特征值的信息增益，画出一棵决策树（用签字笔画出即可）。







【3】 根据下述数据集，编写代码实现信息熵、条件熵、信息增益等决策树的关键环节，并撰写实验报告。（最后一列是类别：是否提供贷款）

dataSet = [ [0, 0, 0, 0, 'no'], #数据集

[0, 0, 0, 1, 'no'],

[0, 1, 0, 1, 'yes'],

[0, 1, 1, 0, 'yes'],

[0, 0, 0, 0, 'no'],

[1, 0, 0, 0, 'no'],

[1, 0, 0, 1, 'no'],

[1, 1, 1, 1, 'yes'],

[1, 0, 1, 2, 'yes'],

[1, 0, 1, 2, 'yes'],

[2, 0, 1, 2, 'yes'],

[2, 0, 1, 1, 'yes'],

[2, 1, 0, 1, 'yes'],

[2, 1, 0, 2, 'yes'],

[2, 0, 0, 0, 'no']]

labels = ['年龄', '有工作', '有自己的房子', '信贷情况']

|  |
| --- |
| import pandas as pd import math   *# 获取一个DataFrame中某一个feature下的所有类别、每个类别的数量、每个类别的索引* def get\_all\_classes(dataFrame, feature):  df\_group = dataFrame.groupby(by=feature) *# 按照feature分组* classes = list(df\_group.groups.keys())  num\_classes = []  groups = []  for i in classes:  num\_classes.append(len(df\_group.get\_group(name=i)))  groups.append(df\_group.get\_group(name=i).index)  return classes, num\_classes, groups   *# 计算某一个DataFrame中某个feature的信息熵* def calculate\_info\_entropy(dataFrame, feature):  total\_num = len(dataFrame)  classes, num\_classes, groups = get\_all\_classes(dataFrame, feature)  info\_entropy = 0 *# 信息熵初始化为0* for i in range(len(classes)):  p = num\_classes[i] / total\_num  info\_entropy += - p \* math.log(p, 2)  return info\_entropy   *# 计算feature2关于feature1的条件熵* def calculate\_conditional\_entropy(dataFrame, feature1, feature2):  total\_num = len(dataFrame)  classes, num\_classes, groups = get\_all\_classes(dataFrame, feature1)  conditional\_entropy = 0  for i in range(len(classes)): *# 遍历关于feature1的每个分组* p = num\_classes[i] / total\_num  info\_entropy = calculate\_info\_entropy(dataFrame.loc[groups[i], :], feature2) *# 计算每个分组的信息熵* conditional\_entropy += p \* info\_entropy *# 将每个分组的信息熵乘以该分组的概率然后加到条件熵里* return conditional\_entropy   *# 计算feature2关于feature1的信息增益* def calculate\_KL\_divergence(dataFrame, feature1, feature2):  info\_entropy = calculate\_info\_entropy(dataFrame, feature2)  conditional\_entropy = calculate\_conditional\_entropy(dataFrame, feature1, feature2)  return info\_entropy - conditional\_entropy   *# 建树* def create\_Decision\_tree(dataFrame):  features = dataFrame.columns *# 获取dataFrame的所有特征* KL\_divergences = []   *# 获取dataFrame关于最后一列的信息熵，用于计算信息增益* info\_entropy = calculate\_info\_entropy(dataFrame, features[-1])  for i in range(len(features) - 1): *# 计算各特征信息增益* KL\_divergence = calculate\_KL\_divergence(dataFrame, features[i], features[-1])   if KL\_divergence == info\_entropy: *# 如果该特征的条件熵为0说明可以直接当做叶子节点了* classes, num\_classes, groups = get\_all\_classes(dataFrame, features[i])  decision\_tree = {}  for k in range(len(groups)):  decision\_tree[features[i] + **' '** + str(classes[k])] = dataFrame.loc[groups[k][0]][-1]  return decision\_tree  else:  KL\_divergences.append(calculate\_KL\_divergence(dataFrame, features[i], features[-1]))  most\_gain = KL\_divergences.index(max(KL\_divergences)) *# 获取最大信息增益点的feature* classes, num\_classes, groups = get\_all\_classes(dataFrame, features[most\_gain]) *# 获取分组信息* decision\_tree = {}  for i in range(len(groups)):  index = [j for j in range(len(features))]  index.pop(most\_gain)  if calculate\_info\_entropy(dataFrame.iloc[groups[i], index], features[-1]) == 0: *# 该组条件熵为0就可以当做叶子节点* decision\_tree[features[most\_gain] + **' '** + str(classes[i])] = dataFrame.loc[groups[i][0]][-1]  else: *# 条件熵不为0就递归调用该方法直到找到叶子节点* decision\_tree[features[most\_gain] + **' '** + str(classes[i])] = create\_Decision\_tree(  dataFrame.iloc[groups[i], index])  return decision\_tree   *# 使用决策树进行分类* def judge(decision\_tree, data):  for i in decision\_tree.keys():  if data[i.split()[0]] == i.split()[1]:  if type(decision\_tree[i]) == str:  return decision\_tree[i]  else:  return judge(decision\_tree[i], data)   df = pd.read\_csv(**'p3.csv'**, error\_bad\_lines=False)  features = df.columns  print(**'==========决策树=========='**) df = pd.read\_csv(**'p3.csv'**, error\_bad\_lines=False) print(create\_Decision\_tree(df)) print(**'==========决策树=========='**) |

实验报告链接：<https://www.jianshu.com/p/7b298900fef1>

1. 对问题【3】中实现的决策树，实现可视化。（选做）

**作业清单（5/27）**

【1】在下列事务数据集中

|  |  |
| --- | --- |
| TID | 项集 |
| 1 | {面包,牛奶} |
| 2 | {面包,尿布,啤酒,鸡蛋} |
| 3 | {牛奶,尿布,啤酒,可乐} |
| 4 | {面包,牛奶,尿布,啤酒} |
| 5 | {面包,牛奶,尿布,可乐} |

项集{啤酒，尿布，牛奶}的支持度为 40% 。

如果将最小支持度定为3，则数据集中的频繁项集有

面包 牛奶 尿布 。

规则{牛奶，尿布}→{啤酒}的支持度为 40% ，置信度为 66.7% 。

【2】阅读微信群发布的“关联规则例子.py”，并根据交易单为（T1，T2，T3，T4，T5，T6，T7，T8，T9），每笔交易的货物清单为{{I1，I2，I5},{I2，I4},{I2，I3},{I1，I2，I4},{I1，I3},{I2，I3},{I1，I3},{I1，I2，I3，I5},{I1，I2，I3}}，编写代码得到关联规则。

|  |
| --- |
| data\_set = [[**'I1'**, **'I2'**, **'I5'**], [**'I2'**, **'I4'**], [**'I2'**, **'I3'**], [**'I1'**, **'I2'**, **'I4'**], [**'I1'**, **'I3'**], [**'I2'**, **'I3'**],  [**'I1'**, **'I3'**], [**'I1'**, **'I2'**, **'I3'**, **'I5'**], [**'I1'**, **'I2'**, **'I3'**]]  min\_sup = 0.2 min\_con = 0.8   def get\_fre\_item\_sets(data\_set, min\_sup):  num\_record = len(data\_set)  min\_sup\_num = min\_sup \* num\_record  fre\_item\_sets = []  fre\_item\_sets.append({})   *# 统计每个元素的频次* for record in data\_set:  for item in record:  if item in fre\_item\_sets[0].keys():  fre\_item\_sets[0][item] += 1  else:  fre\_item\_sets[0][item] = 1   *# 删除低于最小支持度的项* for item in list(fre\_item\_sets[0].keys()):  if fre\_item\_sets[0][item] < min\_sup\_num:  del fre\_item\_sets[0][item]   can\_item\_len = 2  while True:  if len(fre\_item\_sets[can\_item\_len - 2]) < 2:  break  else:  next\_fre\_item\_set = get\_next\_fre\_item\_set(data\_set, fre\_item\_sets[can\_item\_len - 2], can\_item\_len,  min\_sup\_num)  if next\_fre\_item\_set == None:  break  else:  fre\_item\_sets.append(next\_fre\_item\_set)  can\_item\_len += 1  return fre\_item\_sets   def get\_next\_fre\_item\_set(data\_set, fre\_item\_set, can\_item\_len, min\_sup\_num):  fre\_items = list(fre\_item\_set.keys())   next\_fre\_item\_set = {}  for i in range(len(fre\_items) - 1):  for j in range(i + 1, len(fre\_items)):  tempi = set()  if isinstance(fre\_items[i], str):  tempi.add(fre\_items[i])  else:  tempi = set(list(fre\_items[i]))   tempj = set()  if isinstance(fre\_items[j], str):  tempj.add(fre\_items[j])  else:  tempj = set(list(fre\_items[j]))   tempi.update(tempj)   if len(tempi) > can\_item\_len:  continue  if tempi in list(set(item) for item in next\_fre\_item\_set.keys()):  continue  for record in data\_set:  if tempi.issubset(set(record)):  if tempi in list(set(item) for item in next\_fre\_item\_set.keys()):  next\_fre\_item\_set[tuple(tempi)] += 1  else:  next\_fre\_item\_set[tuple(tempi)] = 1   for key in list(next\_fre\_item\_set.keys()):  if next\_fre\_item\_set[key] < min\_sup\_num:  del next\_fre\_item\_set[key]   if len(list(next\_fre\_item\_set.keys())) < 1:  return None  else:  return next\_fre\_item\_set   *# 计算置信度* def calculate\_confidence(fre\_item\_sets, subset, fre\_item):  len\_mother = len(subset)  len\_son = len(fre\_item)  mother\_key = None  son\_key = None  if len\_mother == 1:  mother\_key = subset[0]  else:  mother\_keys = list(fre\_item\_sets[len\_mother - 1].keys())  for i in range(len(mother\_keys)):  if set(subset) == set(mother\_keys[i]):  mother\_key = mother\_keys[i]  break  son\_keys = list(fre\_item\_sets[len\_son - 1].keys())  for i in range(len(son\_keys)):  if set(fre\_item) == set(son\_keys[i]):  son\_key = son\_keys[i]  break  return fre\_item\_sets[len\_son - 1][son\_key] / fre\_item\_sets[len\_mother - 1][mother\_key]   *# 获取关联规则* def get\_association\_rules(fre\_item\_sets, min\_con):  def subsets(itemset):  N = len(itemset)  subsets = []  for i in range(1, 2 \*\* N - 1):  tmp = []  for j in range(N):  if (i >> j) % 2 == 1:  tmp.append(itemset[j])  subsets.append(tmp)  return subsets   association\_rules = []  for i in range(1, len(fre\_item\_sets)):  fre\_item\_set = fre\_item\_sets[i]  for fre\_item in list(fre\_item\_set.keys()):  tmp = {}  all\_subsets = subsets(fre\_item)  for s1 in range(len(all\_subsets) - 1):  for s2 in range(s1 + 1, len(all\_subsets)):  subset1 = all\_subsets[s1]  subset2 = all\_subsets[s2]  if len(subset1) + len(subset2) == len(fre\_item) and len(set(subset1) & set(subset2)) == 0:  confidence = calculate\_confidence(fre\_item\_sets, subset1, fre\_item)  if confidence > min\_con:  temp = str(subset1) + **' > '** + str(subset2)  tmp[temp] = confidence  confidence = calculate\_confidence(fre\_item\_sets, subset2, fre\_item)  if confidence > min\_con:  temp = str(subset2) + **' > '** + str(subset1)  tmp[temp] = confidence  if tmp.keys():  association\_rules.append(tmp)  return association\_rules   fre\_item\_sets = get\_fre\_item\_sets(data\_set, min\_sup)  association\_rules = get\_association\_rules(fre\_item\_sets, min\_con)  for i in association\_rules:  print(i)  运行结果： |