深 圳 大 学 实 验 报 告

课程名称 人工智能与机器学习

项目名称 实验三：金融数据分析

学　　院 金融科技学院

专　　业 金融科技专业

指导教师 刘凤、高灿

报 告 人 谢晓锋 学号 2018031275

实验时间 2021 年 10 月 25 日至 2021 年 11 月 28 日

实验报告提交时间 2021 年 11 月 28 日

教务处制

一、实验目的与要求

实验目的：

1. 了解常用的机器学习方法；
2. 掌握 PCA 和 LDA 降维方法；
3. 掌握有监督分类方法和无监督聚类方法；
4. 运用基本机器学习方法，实现金融数据分析；

二、实验内容与方法

实验内容：

1. 实现金融数据的降维（PCA 或 LDA）；
2. 实现一种有监督分类算法，并应用于金融数据分析；
3. 实现一种无监督聚类算法，并应用于金融数据分析（**可选做**）；

实验平台：Scikit-learn 或华为 Mindspore

实验数据：实验提供数据或**自选数据**

三、实验步骤与过程

本实验的思维导图如图 1 所示，老师、助教审批辛苦了！

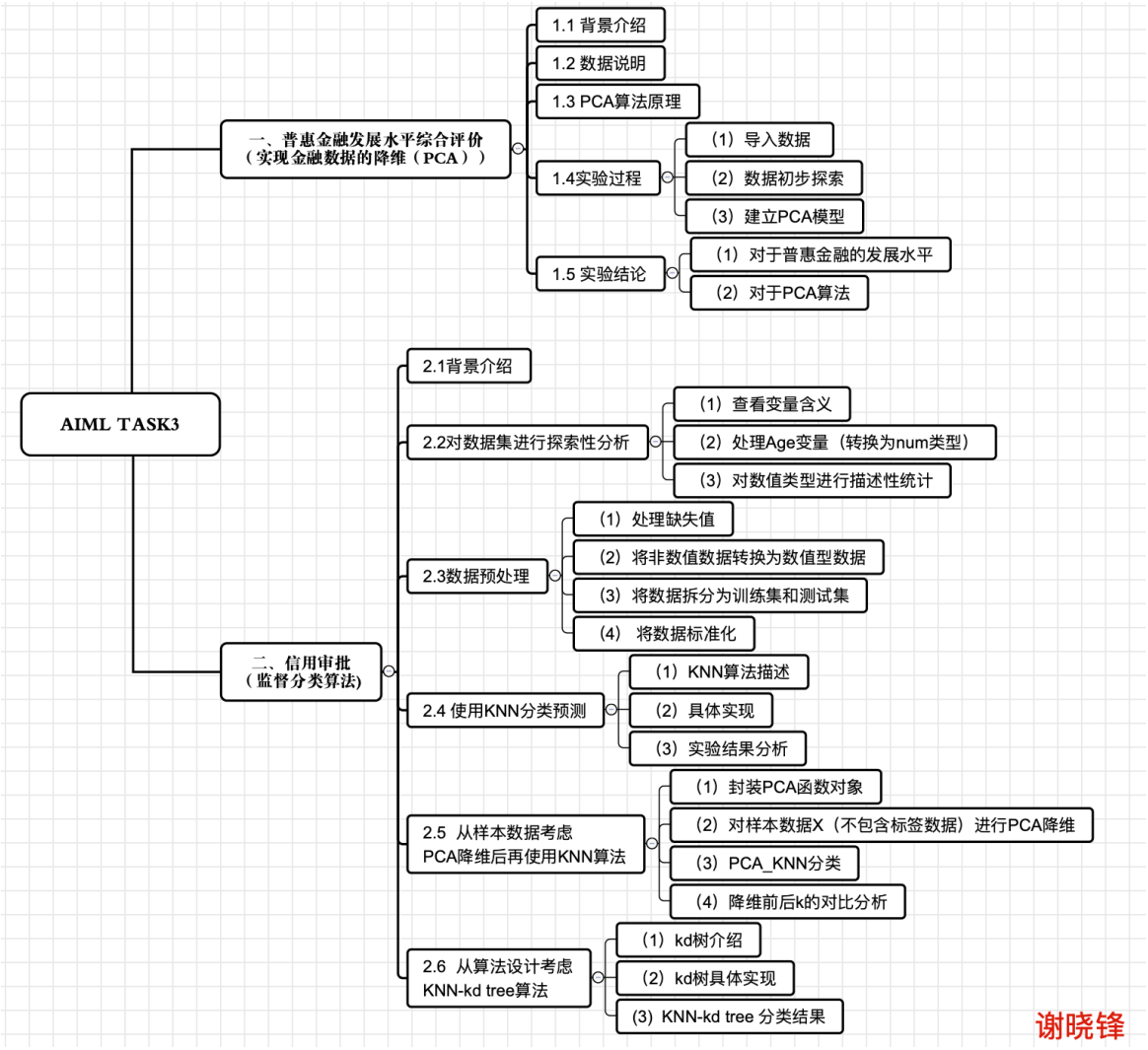


图 1 Task3 实验整体实现思路思维导图

一、普惠金融发展水平综合评价（实现金融数据的降维（PCA））

1.1 背景介绍

普惠金融是指金融机构根据机会平等和商业可持续性发展原则,向所有需要的个人和企业,按照合理的定价提供方便、快捷和可靠的全方位、多层次金融服务[1]。当前,中国普惠金融整体发展水平还有待提高,尤其是区域间(不同省际之间、东中西部之间)发展不平衡,差异大。因此,**构建合理、协调、有效的普惠金融发展水平综合评价指标体系,采用恰当的综合评价方法进行可靠的综合评价,对准确分析不同区域间的差异,厘清制约各地区普惠金融发展的主要因素,已成为中国金融改革与发展的重要理论及现实问题,也是当前金融学界的研究热点和难点问题。**

而根据文献调研，**在综合评价中,主成分分析(PCA)**可以排除评价过程的人为干扰,提取原有指标的绝大部分信息形成主成分及权重,**保持评价结果的客观性**[2],按照影响我国普惠金融发展水平多个强相关性的实测变量通过数学降维的方法,形成具有代表性的要素变量。

基于对普惠金融的研究兴趣，笔者将在综述有关文献的基础上，建立适合中国普惠金融发展水平综合评价的指标体系，运用本课程所学的 **PCA 方法对省际普惠金融发展水平进行综合评价和排序分析。**

1.2 数据说明

在普惠金融发展水平的评价上不能用投入、产出单一的价值指标去衡量，而需要考虑到各个方面，结合《2019 年中国普惠金融发展报告》公布的信息和依据相关文献可以得出[3]，普惠金融发展水平的**主要影响因素有 GDP（亿元）、银行业金融机构网点数（个）、银行业金融机构从业人员数（人）、资产总额（亿元）、本外币存款余额（亿元）、本外币贷款余额（亿元）、保费收入（亿元）、保险密度（元/人）、保险深度（%）等。**

笔者将以 2019 年基于上述指标的普惠金融发展水平进行分析，所有的评价指标数据均来自公开资料，包括《中国统计年鉴》《中国保险年鉴》《中国金融年鉴》和《中国各省市金融运行报告》，由于都是公开资料数据，考虑到可比性、公开性和可获得性，没有对数据做人为修改和调整。原始数据如图 2 所示：

	银行业金融机构网点数（个）	银行业金融机构从业人员数（人）	资产总额（亿元）	本外币存款余额（亿元）	本外币贷款余额（亿元）	保费收入（亿元）	GDP（亿元）	保险密度（元/人）	保险深度（%）
北京	4647	119505	221995	144086	69556.2	1973	28000.4	9085.3	7.05
天津	3129	64606	47928.7	30940.8	31602.5	565	18595.4	3629.2	3.038385837
河北	11689	181096	74030.7	60451.3	43000	1714.5	35964	2287.5	4.77
上海	4099	117599	147074	112462	67182	1587.1	30134	6562.8	5.266808256
江苏	14297	257743	166703	134776	104007	3450	85900.9	4312	4.016255941
广东	17287	358342	227089	195000	126000	4305	89900	3878	4.78865406
浙江	12581	246108	141027	107320.5	90233.3	1844.4	51768	3260	3.56281873
福建	6539	122049	95476	44086.8	41899.7	1032.1	32298.28	2639	3.195526201
海南	1608	20373	15000	10096.4	8459.3	164.83	4462.5	1780.021598	3.693669468
山东	15380	248274	114887	91018.7	70873.9	2738	72678.2	2737	3.76729198
河南	13071	254935	75967	60037.6	42546.8	2020	44988.2	2113.3	4.490066284
山西	7317	123843	42014.8	32844.9	22573.8	823.9	14973.5	2225	5.502387551
湖北	7885	128264	66732	52000	40000	1346.8	36523	2288.6	3.687539359
湖南	9821	133548	56978	48000	32000	1110	34590.6	1617	3.208964285
安徽	8473	123011	59853.6	46146.9	35162	1107.2	27518.7	1757.4	4.023445875
江西	7195	104453	42372	32535.7	25900.4	728	20818.5	1584	3.496889786
内蒙古	5818	99619	34363	23092.7	21566.3	569.9	16103.2	2279.6	3.53904814
重庆	4124	71198	47064	34853.5	28417.5	745	19500.3	2418	3.820454044
贵州	5340	73520	34755	26088.89	20860.34	387.7	13540.83	1107.8	2.863192286
宁夏	1353	23217	9112	5867	6461	165	3454	2423	4.777070064
青海	1118	18658	8975	5843.2	6353.1	80	2642.8	1339	3.027092478
云南	5641	78141	39847	30160.7	25857.6	613.28	16531.3	1282	3.709811086
陕西	7227	103447	47368.4	38153.3	26924.5	868.7	21898.8	2264.9	3.96688403
甘肃	5224	68981	25544	17777.22	17707.24	366.38	7677	1408	4.77243715
四川	14249	231058	92629.5	73079.4	49144.1	1939.4	36980.2	2347	5.244428099
广西	6297	91368	35891	28000	23000	565	20396.3	1157	2.770110265
西藏	664	9426	6621	4959.1	4043.6	28	1310.6	830.861	2.136426064
新疆	3649	63937	30826	21753.1	17477.6	524	10920	2182	4.798534799
吉林	5102	115429	31283	21696.9	18010.3	642	15288.9	2361	4.199124855
黑龙江	6627	122092	37827	23796	19466.1	931	16199.9	2452	5.746949055
辽宁	9393	182637	78767	54249	41279	1275	23942	2913	5.325369643

图 2: 普惠金融发展水平评价指标原始数据

---

[1] 焦瑾璞,黄亭亭,汪天都,张韶华,王瑱.中国普惠金融发展进程及实证研究[J].上海金融,2015(04):12-22.DOI:10.13910/j.cnki.shjr.2015.04.003.

[2] 魏厦.河北省科技竞争力评价研究——基于主成分分析[J].调研世界,2019(06):45-48.DOI:10.13778/j.cnki.11-3705/c.2019.06.008.

[3] 于晓虹,楼文高,余秀荣.中国省际普惠金融发展水平综合评价与实证研究[J].金融论坛,2016,21(05):18-32.DOI:10.16529/j.cnki.11-4613/f.2016.05.003.

1.3 PCA 算法原理

PCA 算法描述如图 3 所示：

输入： 样本集 $D=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_m\}$;
低维空间维数 d'.

过程：

1: 对所有样本进行中心化: $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_i$;
2: 计算样本的协方差矩阵 $\mathbf{X}\mathbf{X}^{\mathrm{T}}$;
3: 对协方差矩阵 $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ 做特征值分解;
4: 取最大的 d' 个特征值所对应的特征向量 $\boldsymbol{w}_1,\boldsymbol{w}_2,\ldots,\boldsymbol{w}_{d'}$.

输出： 投影矩阵 $\mathbf{W}=(\boldsymbol{w}_1,\boldsymbol{w}_2,\ldots,\boldsymbol{w}_{d'})$.

图 3 PCA 算法描述[4]

1.4 实验过程

（1）导入数据

使用语句 df = pd.read_excel('PCA_finance2.xlsx')将数据集导入，同时，考虑到 pandas 库中文字体不兼容的问题，笔者将 DataFrame 的 index 和 columns 修改成英文格式，导入后数据如图 5 所示：

	Banking	Employees	Assets	Deposit	Loan	GDP	IS_income	IS_density	IS_depth
BeiJing	4647	119505	221995.0	144086.0	69556.2	1973.0	28000.4	9085.3	7.046328
TianJin	3129	64606	47928.7	30940.8	31602.5	565.0	18595.4	3629.2	3.038386
HeBei	11689	181096	74030.7	60451.3	43000.0	1714.5	35964.0	2287.5	4.767267
ShangHai	4099	117599	147074.0	112462.0	67182.0	1587.1	30134.0	6562.8	5.266808
JiangSu	14297	257743	166703.0	134776.0	104007.0	3450.0	85900.9	4312.0	4.016256

图 5 导入后数据

（2）数据初步探索

由于此数据集无缺失值，固无需进行数据预处理，为了进一步理清数据处理步骤，笔者对数据进行进一步探索。

1、使用 **df.describe()**语句可以查看每一个维度上数据的数量，均值，标准差，最小值，最大值及其各分位值的相关信息（图略）。

2、使用 **df.corr()**语句可以将每个维度数据的相关性呈现出来，并使用 **sns.heatmap 方法进行热力图**可视化，如图 6 所示，可以发现大部分数据维度都与其他维度呈现出较大的相关性，但是随意删除其中数据又可能会导致信息丢失，所以运用 PCA 进行降维处理是十分有必要的。

3、使用 **df.plot.area()方法绘制面积图**，面积图使用颜色填充的方式很好地突出了趋势分布信息，例如从图 7 可以很明显地看到 Employees 这个维度上数据呈现出比较大地振幅，说明各省之间银行业金融机构从业人员数有着比较明显的差别。

[4] 周志华，西瓜书

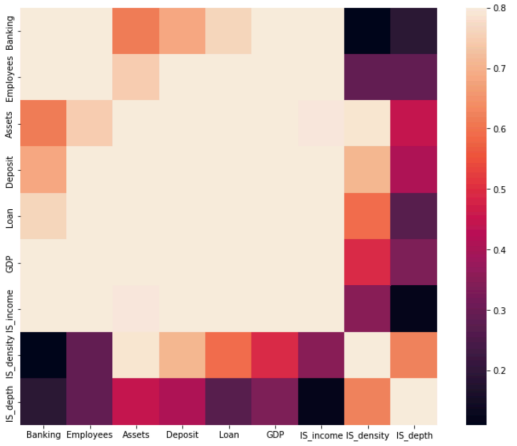


图 6 相关性矩阵热力图

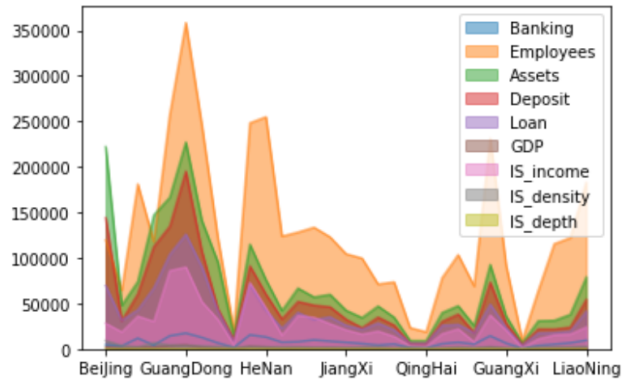


图 7 区域面积图

（3）建立 PCA 模型

Step1： 把 df 转换 numpy 矩阵形式，储存在 data 变量（语句：data = np.mat(df)）

Step2： 数据去中心化，即计算每一列数据对应的均值（X_mean = np.mean(data, axis=0)）,将每一列的各个数据去中心化 (减去均值)（去中心化的数据储存在 dataMax，其中 dataMax.shape = (31, 9)）

Step3： 计算协方差矩阵（covMat = np.cov(dataMax, rowvar=0)）（注意：其中一行为一个样本时，covMat=0;当一列为一个样本时，rowvar 不为 0）

Step4： 计算特征值和特征向量（语句：eigVal, eigVect = np.linalg.eig(np.mat(covMat))

Step5： 调用 sklearn 库中的 minmaxScaler 方法来对特征向量做归一化的处理。

```
from sklearn.preprocessing import MinMaxScaler
minMax = MinMaxScaler()
Norn_eigVect = minMax.fit_transform(eigVect)
Norn_eigVect
```

```
array([[0.95097711, 0.52752331, 0.65187253, 0.37460628, 0.30998711,
        0.        , 0.57212853, 0.06017286, 0.00314866],
       [0.        , 1.        , 0.87294412, 0.41248924, 0.39677813,
        0.75132927, 0.96927375, 0.03072186, 0.00341023],
       [0.33194186, 0.        , 1.        , 0.        , 0.38245295,
        0.69187289, 1.        , 0.03857146, 0.00340888],
```

图 8：minmaxScaler 方法来对特征向量做归一化的处理

Step6： 根据得到的一系列特征值，计算各个特征的方差解释率，并画出**碎石图**。

从表 1 中，可以直观看到各个特征的方差解释率和累计方差解释率，**第 1 个主成分的解释率为 88.46%，第 2 个主成分的解释率为 10.56%，第 3 个主成分的解释率为 0.65%**，之后的成分的解释率几乎为 0，从图 9 碎石图也可以看到，直线几乎不再变化。按照常理，本数据使用两个主成分即可解释方差的 99%，这里笔者为了可视化图表的展示以及更高的解释率，**使用 99.5%的方差解释率选取前 3 个特征作为主成分。**

同时为了程序更好的可扩展性，笔者通过 np.sort(eigVal)语句对特征值进行排序，使用变量 **percentage 来控制主成分的方差贡献率**，可通过要求方差贡献率的大小来确定主成分的个数 n。这里笔者要求使用 99.5%的方差解释率，可以看到图 10 输出的 n=3。

表 1 各个特征的方差解释率

特征	方差解释率	累计方差解释率
特征 1	88.46%	88.46%
特征 2	10.56%	99.02%

特征 3	0.65%	99.67%
特征 4	0.21%	99.88%
特征 5	0.11%	99.99%
特征 6	0.01%	100.00%
特征 7	0.00%	100.00%
特征 8	0.00%	100.00%
特征 9	0.00%	100.00%

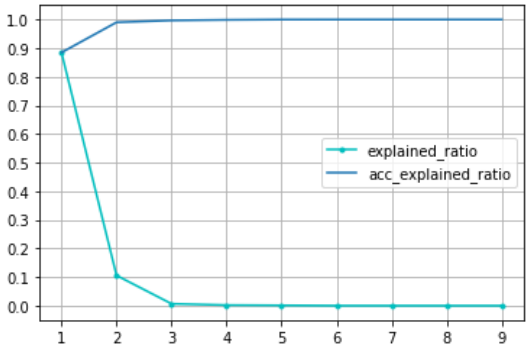


图 9：碎石图

```
percentage = 0.995
sortVal = np.sort(eigVal)[-1::-1]
percentSum, componentNum = 0, 0
for i in sortVal:
    percentSum += i
    componentNum += 1
    if percentSum >= sum(sortVal) * percentage:
        break
n = componentNum
n
```

```
3
```

图 10：确定主成分数量

Step7：通过前面确**定的主成分个数 n = 3，确定主成分的特征向量 n_eigVect**

笔者通过调研，发现 sklearn 还有另外一种归一化的方法 **StandardScale**。**MinMaxScale** 是将样本的特征值转换到同一量纲下把数据映射到[0,1]或者[-1, 1]区间内，仅由变量的极值决定，是一种区间放缩法。StandardScale 是依照特征矩阵的列处理数据，其通过求 z-score 的方法，转换为标准正态分布，和整体样本分布相关，每个样本点都能对标准化产生影响。它们的相同点在于都能取消由于量纲不同引起的误差，且都是一种线性变换，都是对向量 X 按照比例压缩再进行平移。

在对特征向量归一化的过程之中，实际上是使的各个特征的大小范围一致，笔者**使用的 sklearn 中 MinMaxScaler 的方法，**其默认将每种特征的值都归一化到[0，1]之间，归一化后，每个特征均值为 0，标准差为 1。（归一化后的数值大小范围可根据参数 feature_range 调整）。所以 3 个主成分特征向量归一化后的数据如图 11 所示：

	0	1	2
0	0.950977	0.527523	0.651873
1	0.000000	1.000000	0.872944
2	0.331942	0.000000	1.000000
3	0.458870	0.190050	0.542216
4	0.644622	0.379654	0.199330
5	0.987792	0.488216	0.661205
6	0.735100	0.534647	0.000000
7	0.989072	0.463859	0.712548
8	1.000000	0.488094	0.673155

图 11 归一化后的主成分特征向量

	0	1	2
0	118194.046975	23579.341188	205009.484235
1	-31119.096392	-76281.538922	-91705.975697
2	19152.000584	63885.394075	59743.437779
3	75452.226422	14254.106951	108393.381168
4	166250.366902	207679.750966	276074.516423
5	234306.418775	332005.939147	463521.808393
6	106900.155043	165171.232149	219679.336655
7	10131.947816	-3536.903886	16992.642961
8	-80592.132168	-142673.703516	-181736.198428
9	96661.775822	169739.090214	184775.043287
10	27389.899835	143095.720134	126800.981233
11	-37838.948911	-20361.363525	-44774.217261

图 12 降维后的数据(部分)（low_dataMat.shape=(31,3)）

Step8：使用归一化后的主成分特征向量，找到每一个样本在低维空间上的映射（投影），即乘以降维前的数据 dataMax（**low_dataMat = dataMax * n_eigVec**t），**得到降维之后的 31 组样本数据 low_dataMat**。（图 12 所示）

Step9：对 PCA 进行可视化，将这 31 组数据放在以 **3 个主成分为坐标轴形成的散点图中**，从图 13 可以发现这些点虽然是相对分散的，但是对各个样本数据进行普惠金融发展水平评价仍有一定的难度，主要在确定衡量的指标上面。

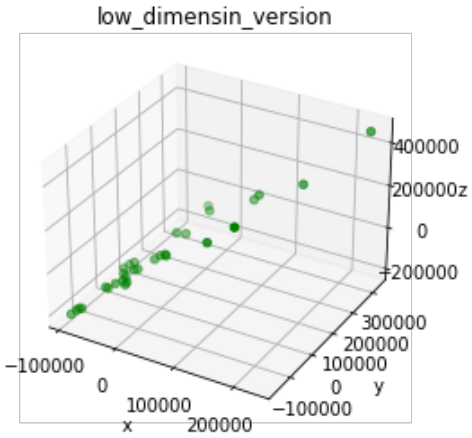


图 13 PCA 效果可视化

Step10：为了得到各省的分数，设置一个**权重 weight（=每一个主成分的方差解释率/总方差解释率）**，将权重乘以投影后的特征向量作为各省的最终得分，并进行排序（表 2 所示）和可视化（图 14 所示）。

```
weight=[]    #计算权重
for i in range(n):
    weight.append(explained_ratio[i] / sum(explained_ratio)  )

#计算各省普惠金融发展水平评价得分
for j in range(31):
    for z in range(9):
        #根据主成分提取的解释程度,计算得分
        score+=(dataMaxList[j][z]*n_eigVect[z][0]*weight[0]+dataMaxList[j][z]*n_eigVect[z][1]*weight[1]+dataMaxList[j][z]*n_eigVect[z][2]*weight[2])
    Score[j].append(score)
    score=0
```

表 2 我国各省普惠金融发展水平综合评价得分

排名	省份	得分	排名	省份	得分	排名	省份	得分
1	广东	246158.59	12	湖北	7190.83	23	内蒙古	-46605.35
2	江苏	171359.10	13	湖南	-2256.35	24	吉林	-49632.44
3	浙江	113812.15	14	安徽	-7529.17	25	贵州	-52054.47
4	北京	108742.07	15	陕西	-26834.21	26	新疆	-60509.42
5	山东	104980.21	16	江西	-32836.87	27	甘肃	-64853.00
6	上海	69186.26	17	重庆	-35511.41	28	海南	-87831.39
7	四川	47848.98	18	山西	-36033.23	29	宁夏	-92681.43

8	河南	40297.66	19	天津	-36300.20	30	青海	-95186.63
9	河北	24156.47	20	广西	-41645.87	31	西藏	-100602.31
10	辽宁	11599.24	21	云南	-42417.77			
11	福建	8729.16	22	黑龙江	-42739.20			

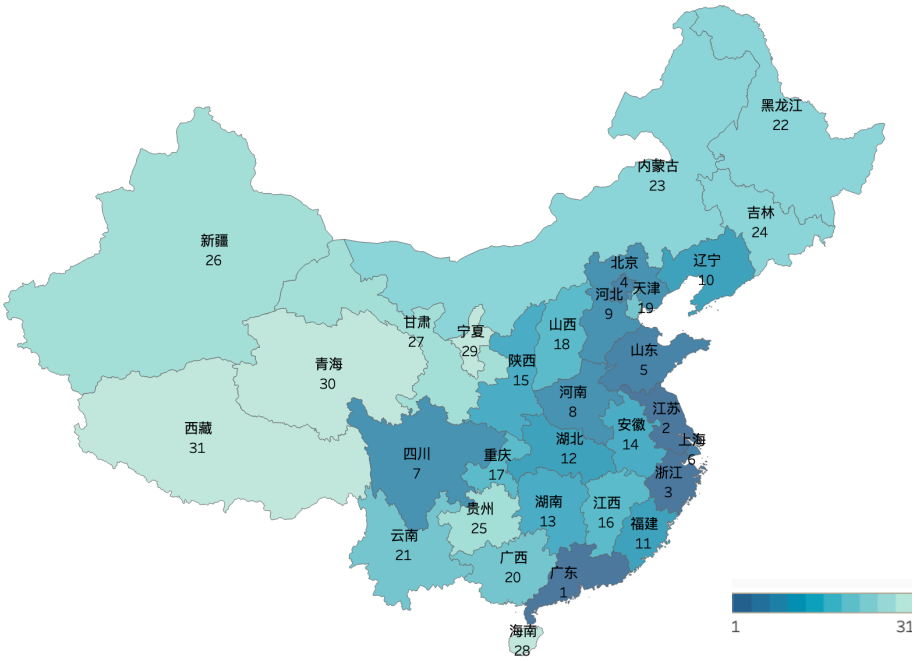


图 14 我国各省普惠金融发展水平地图

1.5 实验结论

（1）对于普惠金融的发展水平

- 笔者通过使用 **PCA 方法和数据可视化方法对我国普惠金融发展水平进行综合评价**，得到国各省**普惠金融发展水平综合评价得分和我国各省普惠金融发展水平地图**，创新了我国普惠金融发展水平评价方法。
- 普惠金融发展各省之间**发展存在不均衡性**。总的来看，以上海为中心的华东地区整体上普惠金融发展水平相对较高，其次是以北京为中心的京津地区，广东省在华南地区独树一帜，而四川省在西部地区明显高出一筹。整体上讲，华南和西部地区普惠金融发展水平相对较低。
- 基于文献调研，相较于传统的实证分析，使用 PCA 方法进行综合评价创新了研究方法，可以**从宏观上把握评价水平，但是由于缺少具体各个指标的回归系数，较难提出针对性的建议。**

（2）对于 PCA 算法

- 本实验笔者只是实现了**最基本的 PCA 算法**，由于本实验的实验数据只有 9 个维度，所以基本的 PCA 算法也基本满足了条件，但是在 PCA 降维中,需要计算样本协方差矩阵XX^T的最大k个特征向量,再将其排序、组成矩阵得到低维投影。倘若样本量太多,样本特征过多,会导致协方差矩阵XX^T计算量变得很大,效率低下。
- 对于效率问题，一种方法是**使用 SVD 分解协方差矩阵**，传统的 PCA 算法通过特征分解求解,而 PCA 算法也可以通过 SVD 来完成。**事实上,scikit-learn 的 PCA 算法就是用 SVD 分解,而不是特**

征值分解。在不求解协方差矩阵的情况下,得到右奇异矩阵 V,即 PCA 算法可以不通过特征分解,而是采用 SVD 方法,这个方法在样本量很大的时候很有效。

- 还可以**结合 GBDT 算法:**基于改进主成分分析法的特征约简算法,结合数据挖掘,引入 GBDT 算法进行维度缩减。针对具有高阶相关性的属性特征,或属性主特征的非正交方向上有多个方差较大的情况下,PCA 算法的属性约简性能会大大降低。因此,引入 GBDT 算法更新 PCA 算法的初始输入,在 GBDT 算法多参数的干预下实现数据个性化预处理,降低样本属性特征的高阶相关性,从而快速计算得到方差较大的方向向量,实现样本属性约简。

二、信用审批 (实现一种有监督分类算法，并应用于金融数据分析)

2.1 背景介绍

商业银行会收到很多**信用卡申请**，他们中的许多人会因为多种原因而被拒绝，例如有较高的贷款负债、较低的收入水平或较低的个人征信。如果靠人工手动分析这些信用卡申请，效率很低，容易出错且耗时，所以现在几乎每家商业银行都通过机器学习的进行自动化处理。在本次实验中，笔者将**使用机器学习构建一个像真实的银行一样的自动信用卡审批分类器。**

2.2 对数据集进行探索性分析

（1）查看变量含义

笔者在本次实验使用的数据集是:“Credit Approval.csv”（来自 UCI 机器学习存储库）[5]首先，笔者对数据集进行探索性分析。通过 data.head()命令**查看原始数据集**，如图 15 所示：

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1.0	f	g	202	0.0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6.0	f	g	43	560.0	+
2	a	24.5	0.500	u	g	q	h	1.50	t	f	0.0	f	g	280	824.0	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5.0	t	g	100	3.0	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0.0	f	s	120	0.0	+

图 15 data.head()命令查看原始数据集

可以看到，为了保护隐私，这个数据集的特征名称已经被匿名化了，所以数据集的看起来不知所云。通过查阅资料[6]，笔者对数据集有了清晰的了解（表 3）：**前 15 个变量是信用申请属性,Approved 是信用审批状态，其值为“+”或“-”，表示是否已通过信用卡审批。**

表 3：数据集特征含义

列	属性	类型	数据
0	Male(性别)	chr	b, a
1	Age(年龄)	chr	continuous.
2	Debt(债务)	num	continuous.
3	Married(婚姻)	chr	u, y, l, t.
4	BankCustomer(银行客户)	chr	g, p, gg.

[5] UCI Machine Learning Repository： http://archive.ics.uci.edu/ml/datasets/credit+approval
[6] 参考：http://rstudio-pubs-static.s3.amazonaws.com/73039_9946de135c0a49daa7a0a9eda4a67a72.html

5	EducationLevel(教育程度)	chr	c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff.
6	Ethnicity(种族)	chr	v, h, bb, j, n, z, dd, ff, o.
7	YearsEmployed(工作年限)	num	continuous.
8	PriorDefault(违约记录)	num	t, f.
9	Employed(就业状况)	num	t, f.
10	CreditScore(信用评分)	num	continuous.
11	DriversLicense(驾照)	chr	t, f.
12	Citizen(公民)	chr	g, p, s.
13	ZipCode(邮政编码)	chr	continuous.
14	Income(收入)	num	continuous.
15	Approved（是否批准）	chr	+,- (class attribute)

（2）处理 Age 变量（转换为 num 类型）

通过 **data.shape 命令查看数据集格式**，发现数据格式为(695, 16)，通过 data.info()查看数据集变量类型，如图 X 所示，数据集包含数值和非数值类型（float64 和 object）。其中，特征 2、7、10 和 14 是**数值类型，其他都是非数值类型。**

可以发现**特征 1 为 object 类型**，但根据表 3，这个变量代表的是 Age，通过语句 data[1] = pd.to_numeric(data[1])将其转换为 num 类型，以便后续的处理。这里需要注意的是这个变量因为存在缺失值为“？”，所以再转换类型前需要先通过 data.replace('?', np.NaN, inplace = True)语句**将“？”替代为空值**。（笔者刚开始没有先对 Age 变量进行处理，是后面使用 LabelEncoder 对数据集做数值变换才发现问题，所以才返回到开头这里先处理 Age 变量）

```
shape of the data: (695, 16)
-----------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 695 entries, 0 to 694
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       693 non-null    object
 1   1       690 non-null    object   Age
 2   2       690 non-null    float64
 3   3       690 non-null    object
 4   4       690 non-null    object
 5   5       690 non-null    object
 6   6       690 non-null    object
 7   7       690 non-null    float64
 8   8       690 non-null    object
 9   9       690 non-null    object
 10  10      690 non-null    float64
 11  11      690 non-null    object
 12  12      690 non-null    object
 13  13      690 non-null    object
 14  14      690 non-null    float64
 15  15      690 non-null    object
dtypes: float64(4), object(12)
memory usage: 87.0+ KB
None
```

图 16 data.shape 和 data.info() 查看数据集大小和类型

（3）对数值类型进行描述性统计

进一步地，笔者使用 **data.describe()语句查看连续型数值的描述性统计信息**，如最小值、最大值、平均值和标准差，同时，可以看到该数据集特征范围不一致（0-28、0-67、0-100000）。如图 X 所示，列 1、2、7、10、14 分别代表 Age(年龄)、Debt(债务)、YearsEmployed(工作年限)、CreditScore(信用评分)、Income(收

入)。

	1	2	7	10	14
count	678.000000	690.000000	690.000000	690.00000	690.000000
mean	31.568171	4.758725	2.223406	2.40000	1017.385507
std	11.957862	4.978163	3.346513	4.86294	5210.102598
min	13.750000	0.000000	0.000000	0.00000	0.000000
25%	22.602500	1.000000	0.165000	0.00000	0.000000
50%	28.460000	2.750000	1.000000	0.00000	5.000000
75%	38.230000	7.207500	2.625000	3.00000	395.500000
max	80.250000	28.000000	28.500000	67.00000	100000.000000

图 17 data.describe()查看连续型数值的描述性统计信息

2.3 数据预处理

可以看到数据集混合了数值型和非数值型数据，此外还包含了许多缺失值，所以接下来必须对数据集进行预处理。这里提出的一个问题是：**为什么不能直接忽略缺失值？**因为忽略缺失值会严重影响机器学习模型的性能。在忽略缺失值的同时，机器学习模型可能会错过对其训练有用的数据集信息，同时也有许多模型不能自动处理缺失值（如 LDA）。

（1） 处理缺失值

首先，通过查看数据集，发现数据集中的缺失值被标为 “?”，笔者先通过 data.replace('?', np.nan)语句暂时**用 NaN 替换这些缺失值的“？”**

对于数值型缺失值的处理，有两种方法，一是简单地使用**均值插补**，用特征值的平均值来替代缺失值；二是检查数值之间的关系并使**用线性回归来替代缺失值**。这里笔者简单使用 data.fillna(data.mean(), inplace=True)语句进行均值插补。（注意：这里只是处理了数值列中的缺失值，均值插补对非数值列中的缺失值不起作用）

对于非数值型缺失值的处理，笔者使用出现**频次最高的值进行插补**（关键语句：data= data.fillna(data[col].value_counts().index[0])），一般而言，在为分类数据插补缺失值时，这是一种常见且有效的做法[7]

最后，通过 print(data.isnull().values.sum())语句计算数据集中的 NaN 数量并打印进行验证，可以看到这是数据集中**缺失值数量为 0**（见 jupyter 文件）。

（2） 将非数值数据转换为数值型数据

接下来需要将所有非数值值转换为数值值。这是因为这不仅计算更快，而且许多机器学习模型都要求输入的数据为数值型格式。笔者在这里使用的工具是 **LabelEncoder[8]，完成将离散型的数据转换成 0 到 n−1 之间的数**，n 是一个列表的不同取值的个数，即某个特征的所有不同取值的个数。

```
1. from sklearn.preprocessing import LabelEncoder #导入标签编码器
2. le = LabelEncoder() # 实例化
3.
4. for col in data: # 迭代每列的所有值并提取它们的数据类型
5.     if data[col].dtype=='object': # 判断数据类型是否为对象类型
6.         data[col]=le.fit_transform(data[col]) #使用 LabelEncoder 做数值变换
7. data.tail(15)
```

结果如图 18 所示，数据集已经全部都转化为数值型数据。

[7] 参考：https://www.datacamp.com/community/tutorials/categorical-data
[8] 参考：https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
680	1	41.58	1.040	2	1	0	8	0.665	1	1	0.0	1	1	61	237.0	1
681	1	30.58	10.665	2	1	11	4	0.085	1	2	12.0	2	1	11	3.0	1
682	2	19.42	7.250	2	1	10	8	0.040	1	2	1.0	1	1	1	1.0	1
683	1	17.92	10.210	2	1	6	3	0.000	1	1	0.0	1	1	0	50.0	1
684	1	20.08	1.250	2	1	2	8	0.000	1	1	0.0	1	1	0	0.0	1
685	2	19.50	0.290	2	1	9	8	0.290	1	1	0.0	1	1	74	364.0	1

图 18 转化为数值型数据

（3） 将数据拆分为训练集和测试集

理想情况下，不应使用测试集中的任何信息来扩展训练集，接下来笔者将数据集分成训练集和测试集。此外，在此次预测信用卡审批中，像 **DriversLicense 和 ZipCode 这样的特征**并不是很重要，所以笔者进行了**特征选择**，在这里将这两个特征去除，以设计具有最佳特征集的机器学习模型。

```
from sklearn.model_selection import train_test_split
#去掉特征 11 和 13 并将 DataFrame 转换为 NumPy 数组
data = data.drop([11, 13], axis=1)
data = data.values
#将特征和标签分离到单独的变量中
X,y = data[:,0:13] , data[:,13]
# 分成训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.33, random_state=42)
```

最终的数据格式下表 4 所示：(注意，训练样本中 X 是大写的，而标签 y 是小写的，因为矩阵习惯用大写字母表示，向量习惯用小写字母来表示)

表 4：数据集变量解释表

变量	含义	大小
data	整个数据集	(695, 14)
X_train	训练集样本	(465, 13)
y_train	训练集标签	(465,1)
X_test	测试集样本	(230,13)
y_test	测试集标签	(230,)

（4） 将数据标准化

如在 PCA 中使用的，笔者同样**使用 MinMaxScaler 方法对数据进行标准化。**

```
from sklearn.preprocessing import MinMaxScaler
#实例化 MinMaxScaler 并缩放 X_train 和 X_test
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX_train = scaler.fit_transform(X_train)
rescaledX_test = scaler.fit_transform(X_test)
```

2.4 使用 KNN 算法对数据集进行分类预测

（1）KNN 算法描述

KNN 算法原理较为简单，其算法描述如下表 5 所示，笔者不再进行论述。

表 5 KNN 算法描述

KNN 分类算法

输入： 训练集 $T=\{(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)\}$，其中$x_i$为 n 维特征向量，$y_i \in \{c_1,c_2,\ldots,c_K\}$，测试样本$x^*$

1：根据给定的距离度量，在训练集 T 中找出与x^*距离最近的 k 个点，记为：$T_k(x^*)$

2：在$T_k(x^*)$中根据分类决策规则（如多数投票表决）来决定x^*的类别y^*：

$$y^* = \underset{c_j}{\operatorname{argmax}} \sum_{x_j \in T_k(x^*)} I(y_i = c_j) \quad i = 1,2,\ldots,N; j = 1,2,\ldots,K$$

输出： 测试样本x^*的类别y^*

（2）具体实现

首先构建 KNN 类对象，**步骤如下：**

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

具体实现如下图 19 所示，笔者对每一行代码都进行了注释。值得注意的是，这里笔者原先直接使用了数组类型进行运算，结果出现了广播扩展问题，经过查阅资料，发现 **ndarray 数组可以自己广播扩展然后对位运算**，所以在开始就把 X,y 转化成 ndarray 数组类型，再进行运算。

```
class KNN1:
    #初始化方法，k表示邻居的个数
    def __init__(self, k):
        #用当前的k来初始化KNN1对象中的k
        self.k = k

    # X：训练集样本，类数组类型；  y：训练集标签，类数组类型
    def fit(self, X, y):
        # 把X,y转化成ndarray数组类型
        self.X = np.asarray(X)
        self.y = np.asarray(y)

    #  根据参数传递的样本，对测试集样本进行预测。X：训练集样本，类数组类型；
    def predict(self,X):
        X = np.array(X)
        # 储存预测的结果，数组类型
        result = []
        # 对ndarray数组进行遍历，每次取数组中的一行。
        for x in X:
            # 求测试集中的每个样本与训练集中所有点的距离。
            # x代表测试集中的一行，X代表训练集中的所有行。ndarray数组可以自己广播扩展然后对位运算。
            dis = np.sqrt(np.sum((x - self.X) ** 2,axis=1))
            # argsort可以将数组排序后，每个元素在原数组中的索引位置。
            index = dis.argsort()
            # 进行截断，只取前k个元素【取距离最近的k个元素的索引】
            index = index[:self.k]
            # 返回数组中每个元素出现的次数。元素必须是非负的整数。
            count = np.bincount(self.y[index])
            #  返回ndarray数组中，值最大的元素对应的索引，该索引就是预测判定的类别。
            result.append(count.argmax())
        return np.asarray(result)
```

图 19： KNN 具体实现代码

由于 KNN 算法对 K 的选取很敏感，笔者**设置 maxK 变量控制 K 进行迭代运行（K 只取奇数）**，并创建 KNN 对象，对数据集进行预测，得到每个 K 下的正确率，储存在 rate_set 列表（如图 X 所示）。并**计算最高准确率、最低准确率及其对应的 K 值、平均准确率和方差，并进行可视化。**

```python
rate_set = []
maxK = 51 #控制K的大小
for k in range(1,maxK): #迭代K
    if k % 2 == 1:
        knn = KNN1(k)  # 创建KNN对象
        knn.fit(X_train,y_train)  # 进行训练
        result = knn.predict(X_test) # 进行测试，获得测试的结果
        accRate = np.sum(result == y_test)/len(result) # 计算正确率
        rate_set.append(accRate) #添加到list
```

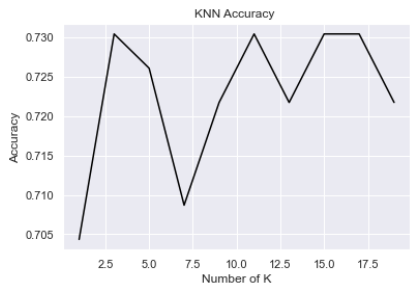
图 20 控制 k 大小运行 KNN

（3）实验结果分析

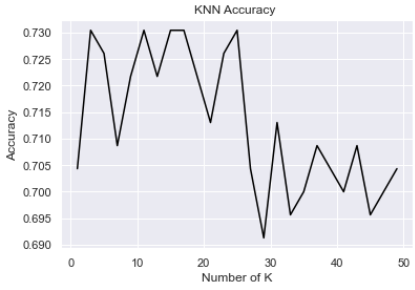
从表 6 可以看出，根据 maxK 的变动，可以看到 KNN 对此数据集分类预测的**最大准确率为 73.04%，此时 k=3，可能由于样本量过少，这个准确率表现较为一般。且随着 k 的增加，当 k 达到 255 后，算法的准确率将稳定在 64.63%,保持在最低准确率。**

表 6：k 变动时，KNN 算法准确率数据

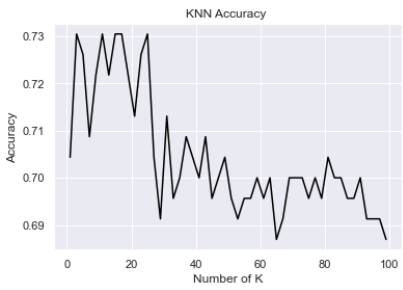
maxK	最高准确率(k)	最低准确率(k)	平均准确率	方差
21	73.04% (k=3)	70.43% (k=1)	72.26%	0.01%
51	73.04% (k=3)	69.13% (k=29)	71.29%	0.02%
101	73.04% (k=3)	68.70% (k=65)	70.44%	0.02%
201	73.04% (k=3)	62.17% (k=195)	68.26%	0.08%
301	73.04% (k=3)	54.78% (k=255)	64.63%	0.34%



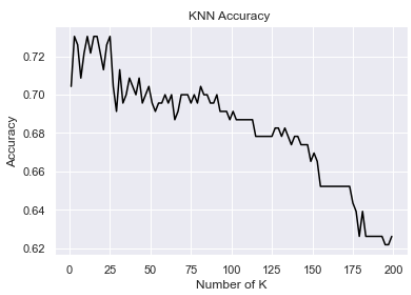
MaxK = 21



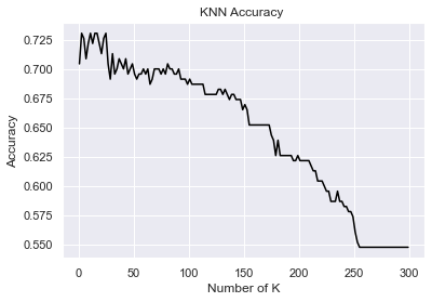
MaxK = 51



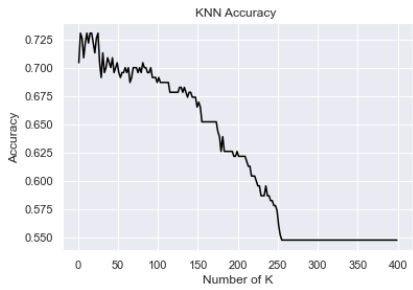
MaxK =101



MaxK = 201



MaxK = 301



MaxK =401

图 21：k 变动时，KNN 算法准确率曲线

2.5 从样本数据考虑：PCA 降维后再使用 KNN 算法

笔者考虑到可能是因为样本数据的原因，所以尝试将数据进行 PCA 降维后再使用 KNN 算法。

（1）封装 PCA 函数对象

首先对前面所详细叙述的 PCA 算法进行封装，实现通过输入要降维的数据和要求的方差解释率，实现对数据进行降维，结构如下所示：

```
class PCApercent(object):
    def __init__(self, X, percentage):
    def percent2n(self, eigVal): # 通过方差百分比选取前 n 个主成分
    def _fit(self):  #PCA 算法
    def fit(self):
    def show(self): #画碎石图
```

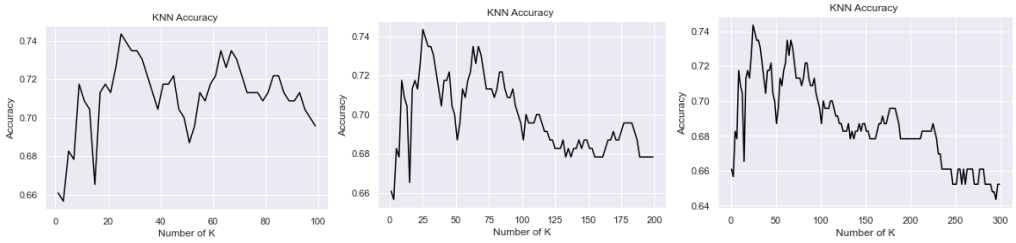
（2）对样本数据 X（不包含标签数据）进行 PCA 降维

从图 22 中可以看到 99.9999%的方差解释率下，**将 13 维度保留 5 个主成分**，从碎石图可以看到，第 1 个主成分已经解释了 99.9%的数据，笔者为了保留几乎完全的数据信息，**将方差解释率提高到 99.9999%。**同样进行训练集、测试集拆分和数据归一化，可以看到 X_train 和 X_test 的大小变为（465，5）和（230，5）

图 22 对样本数据 X 进行 PCA 降维

（3）PCA_KNN 分类

同样使用 KNN 分类器，通过 k 进行迭代和可视化，结果得到，**当 k=25 时，分类的准确率达到最高，为 74.35%，这个准确率比未降维前的最高准确率提高了 1%，效果不是很明显。**



MaxK = 101　　MaxK = 201　　MaxK =301

图 23：k 变动时，PCA_KNN 算法准确率曲线

（4）降维前后 k 的对比分析

对比当未降维前，k=3 时取得最大准确率，降维后，k=25 时取得最大准确率，这是因为 k 的选择反映了训练误差（近似误差）与测试误差（估计误差）的权衡，即：

若 k 取较小值（模型复杂），预测实例较依赖于近邻样本，样本整体利用率低，模型对噪声数据敏感，且可能出现训练误差小（过拟合）、测试误差大的情况；

若 k 值较大（模型简单），预测实例可利用较多的样本信息，模型抗干扰性强，但计算复杂，且可能出现训练误差大（欠拟合）、测试误差小的情况。

2.6 从算法设计考虑： KNN-kd tree 算法[9] [10] [11]

（1）kd 树介绍

上述实现 KNN 算法时，笔者使用的是最简单的线性扫描的方式，即计算所有样本点与输入实例的距离，再取 k 个距离最小的点作为 k 近邻点。而 KNN 主要考虑的问题是如何**在训练样本集中快速 k 近邻**

[9] 参考：https://cloud.tencent.com/developer/article/1178474
[10] 参考：https://www.cnblogs.com/21207-ihome/p/6084670.html
[11] 参考：https://blog.csdn.net/sinat_34072381/article/details/84104440

搜索，所以另一种更优化的想法是，构建数据索引，即通过构建树对输入空间进行划分，kd 树就是此种实现。

kd 树（k-dimension tree，k 是指特征向量的维数），是一种存储 k 维空间中数据的平衡二叉树型结构，主要用于范围搜索和最近邻搜索。kd 树实质是一种空间划分树，其每个节点对应一个 k 维的点，每个非叶节点相当于一个分割超平面，将其所在区域划分为两个子区域。**kd 树的结构可使得每次在局部空间中搜索目标数据，减少了不必要的数据搜索，从而加快了搜索速度。**

（2）kd 树具体实现

kd 树的具体实现包括 Step1 :构建 kd 树和 Step 2:在 kd 树中搜索输入样本的 k 近邻，由于篇幅所限，笔者在这里只阐述实现思路，具体代码请见附件。

表 7：knn - kd 树具体实现思路

Step1 构建 kd 树

（1）输入数据集$T=\{\boldsymbol{x}_1,\cdots,\boldsymbol{x}_N\}$,其中$\boldsymbol{x}_i=\left(x_i^1,\cdots,x_i^k\right)$

（2）构建根节点，根节点对应于包含T的k维空间的超矩形区域。选取x^1为切分轴、T中所有点x^1坐标的中位数为切分点,使用过切分点且与垂直于切分轴的超平面,将根节点对应的超矩形区域切分为两个子区域,并对应于其左右子节点。其中,在节点区域各点的x^1坐标不大于切分点,右节点区域各点的x^1坐标大小于切分点,并将切分点保存在根节点。

（3）对子节点重复步骤I,即对于深度为j的节点j_i,选择x^l为切分轴、j_i包含的区域中所有点x^l坐标的中位数为切分点,其中$l=(j+1)\ \mathrm{mod}\ k$,将j_i对应的区域划分为两个子区域，并对应其左右子节点，直至两个子区域没有实例为止。

Step2 在 kd 树中搜索输入样本的 k 近邻

（4）首先构建空的最大堆 (列表), 从根节点出发, 计算当前节点与输入实例的距离, 若最大堆元素小于 k 个, 则将距离插入最大堆中, 否则比较该距离是否小于堆顶距离值，若小于，则使用该距离替换堆顶元素

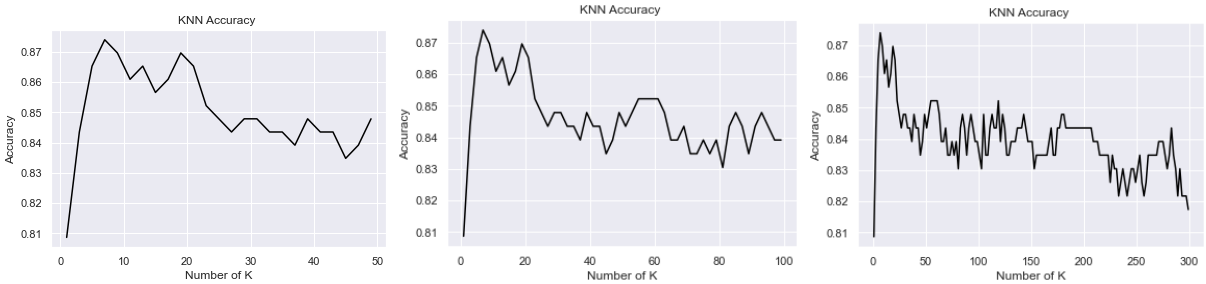
（5）递归的遍历 kd 树中的节点, 通过如下方式控制进入分支:

- 若堆中元素小于 k 个或该节点中的样本点与输入实例形成的超球体包含堆顶样本点, 则进入左右子节点搜索;
- 否则, 若输入实例当前维的坐标小于该节点当前维的坐标, 则进入左子节点搜索;
- 否则, 进入右子节点搜索;

（6）当到达叶节点时, 搜索结束。最后最大堆中的 k 个节点, 即为输入样本的 k 近邻点

(3) KNN-kd tree 分类结果

同样通过 k 进行迭代和可视化，结果得到，**当 k=7 时，分类的准确率达到最高，为 87.39%，这个准确率比原始 KNN 的最高准确率提高了 14.35%（87.39%-73.04%），效果提升明显。且当 k=1 时，算法的准确率最低，但也达到了 80.87%。**



MaxK = 51　　MaxK = 101　　MaxK =301

图 24：k 变动时,KNN_kd tree 算法准确率曲线

四、实验结论或体会

1、本次实验笔者对于题目 1: 实现金融数据的降维，笔者基于个人对普惠金融的研究兴趣，在初步进行文献调研后，决定使用 PCA 算法对我国普惠金融发展水平进行综合评价分析，在这个过程中，对 PCA 算法进行了较为深入的理解和扩展，最后对各省的普惠金融发展水平进行了排名，并画出了我国普惠金融发展水平地图。这是一个将所学算法知识应用于研究兴趣的过程，收获颇丰，也对机器学习这门课程产生了更加浓厚的兴趣。同时，笔者也感叹数据获取来之不易，即使有很好的算法，如果没有质量好的数据，也是巧妇难为无米之炊…

2、对于题目 2: 实现一种有监督分类算法，并应用于金融数据分析，笔者使用老师给的信用审批数据进行分析。完整地探索了数据预处理的过程，并且通过普通 KNN 算法（最高准确率为 73.04%，k=3）、PCA 降维后的 KNN 算法（最高准确率为 74.35%，k=25）、KNN kd tree 算法（最高准确率为 87.39%，k=7）对此数据集进行对比分析和可视化。在这个过程中遇到最主要的困难是两个，一个是如何将算法进行封装调用，并且根据变量进行可视化，需要细心考虑整个数据集整体的变量；另一个是 KNN kd tree 算法笔者是自编实现的，这个过程有点复杂，虽然借鉴了很多网上的代码，但是还是要理解后根据自己的数据和可视化过程对函数进行封装。

3、总的来说，本次实验过程中也踩了很多坑，收获了很多经验。虽然为了整体实验的逻辑性，笔者耗费了很多时间，但是其实对于一个数据集如何去进行处理的问题，其实还有很多算法是笔者想继续进行尝试的。同时，整个过程也感受到了理解算法的推导过程多么地重要，这是单纯做一个调包侠所不能比拟的。总之，笔者将继续探索和努力，将所学转换为所用。

五、思考题

简述机器学习分类与聚类概念并分析其差异

一、分类有如下几种说法，但表达的意思是相同的。

- 分类任务就是通过学习得到一个目标函数 f，把每个属性集 x 映射到一个预先定义的类标号 y
- 是根据一些给定的已知类别标号的样本，训练某种学习机器（即得到某种目标函数），使它能够对未知类别的样本进行分类。这属于监督学习。
- 通过学习来得到样本属性与类标号之间的关系。

二、聚类的相关的一些概念如下

- 指事先并不知道任何样本的类别标号，希望通过某种算法来把一组未知类别的样本划分成若干类别，聚类并不关心某一类是什么需要实现的目标只是把相似的东西聚到一起，是无监督学习
- 通常，人们根据样本间的某种距离或者相似性来定义聚类。
- 聚类的目标：组内的对象相互之间时相似的（相关的），而不同组中的对象是不同的（不相关的）。组内的相似性越大，组间差别越大，聚类就越好。

三、分类与聚类的比较

（1）类别是否预先定义是最直观区别

分类是把某个对象划分到某个具体的已经定义的类别当中，而聚类是把一些对象按照具体特征组织到若干个类别里。虽然都是把某个对象划分到某个类别中，但是分类的类别是已经预定义的，而聚类操

作时，某个对象所属的类别却不是预定义的。所以，对象所属类别是否为事先，是二者的最基本区别。而这个区别，仅仅是从算法实现流程来看的。

（2）二者解决的具体问题不一样

分类算法的基本功能是做预测。我们已知某个实体的具体特征，然后想判断这个实体具体属于哪一类，或者根据一些已知条件来估计感兴趣的参数。预测的结果为离散值，当预测结果为连续值时，分类算法可以退化为计量经济学中常见的回归模型。分类算法的根本目标是发现新的模式、新的知识，与数据挖掘数据分析的根本目标是一致的。

聚类算法的功能是降维。假如待分析的对象很多，我们需要归归类，划划简，从而提高数据分析的效率，这就用到了聚类的算法。聚类方法只能起到降低被分析问题的复杂程度的作用，即降维，一百个对象的分析问题可以转化为十个对象类的分析问题。聚类的目标不是发现知识，而是化简问题，聚类算法并不直接解决数据分析的问题，而最多算是数据预处理的过程。

（3）有监督和无监督

分类是有监督的算法，而聚类是无监督的算法。有监督的算法并不是实时的，需要给定一些数据对模型进行训练，有了模型就能预测。新的待估计的对象来了的时候，套进模型，就得到了分类结果。而聚类算法是实时的，换句话说是一次性的，给定统计指标，根据对象与对象之间的相关性，把对象分为若干类。分类算法中，对象所属的类别取决于训练出来的模型，间接地取决于训练集中的数据。而聚类算法中，对象所属的类别，则取决于待分析的其他数据对象。

（4）数据处理的顺序不同

分类算法中，待分析的数据是一个一个处理的，分类的过程，就像给数据贴标签的过程，来一个数据，我放到模型里，然后贴个标签。聚类算法中，待分析的数据同时处理，来一堆数据过来，同时给分成几小堆。因此，数据分类算法和数据聚类算法的最大区别是时效性问题。在已有数据模型的条件下，数据分类的效率往往比数据聚类的效率要高很多，因为一次只是一个对象被处理，而对于聚类结果来说，每当加入一个新的分析对象，类别结果都有可能发生改变，因此很有必要重新对所有的待分析对象进行计算处理。

（5）典型的分类算法与聚类算法

典型的分类算法有：决策树，神经网络，支持向量机模型，Logistic 回归分析，以及核估计等等。聚类的方法有，基于链接关系的聚类算法，基于中心度的聚类算法，基于统计分布的聚类算法以及基于密度的聚类算法等。

指导教师批阅意见：   成绩评定： 指导教师签字：刘凤、高灿 2021 年 12 月 2 日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。