2021 寒假美研 R 计划项目远程——数据挖掘方向（社交网络应用中的数据挖掘与分析）项目总结报告

姓名：赵君骁

学校：纽约大学 (New York University)

学院：文理学院 (College of Arts and Sciences)

专业：数学与计算机联合专业

时间：2021 年 4 月

目录

第一章 项目背景

1.1 大数据时代

自从互联网革命，尤其是移动互联网革命以来，互联网已经成为了人们生活中密不可分的一部分；而人们在网上的一切操作，如浏览、购物等，都会产生大量数据。可以说，数据已经成为了我们这个信息化时代重要的生产要素。因此，如何发掘和分析这些由人们网上活动产生的大量未经处理的数据就显得尤为重要；而通过发掘和分析这些数据，我们可以越来越精准的判断人民的需求，以便提供更加便捷与个性化的服务。而随着万物互联时代的到来，物联网的设备快速增加，由此产生的数据亦将再次爆炸式的增长。如何充分利用好这些数据来造福于人将会在很长一段时间里是研究的热门方向。

1.2 社交网络应用中的数据

相比于传统媒体如电视、广播、报纸等无差别的信息传播，各大主流社交媒体平台的推送显然更加“投其所好”，而他们的个性化推荐算法都离不开大数据的支持。与此同时，人们在社交媒体平台上花费大量的时间进行浏览与互动，期间产生的大量的数据被用来进行用户画像和算法模型以及人工智能的训练，从而更加精准的进行信息的推送和广告的投放。因此，通过挖掘与分析用户数据，社交媒体平台可以实现更加精准的推送来增加用户活跃度与粘性，并利用更多用户产生的数据来进一步优化推送算法。

第二章 项目过程

2.1 数据挖掘与分析概述

数据挖掘是指从大量未经整理的数据中寻找隐藏的有价值的信息；而通过系统性的分析这些信息，我们可以从中找出规律。因此，进行数据的挖掘与分析需要大量的数据、较高的运算能力（用以快速处理数据）和准确高效的算法（在尽可能保留数据特征的情况下降低数据维度与复杂度以便更加快速的处理数据）。

2.2 算法学习

由于我们要分析的是大量未处理的数据，首先我们要对数据进行预处理后才能进一步寻找规律。在本次 R 计划项目中，主要教授了 4 类算法，分别对应数据预处理、非监督式学习、监督式学习与规律寻找。

2.2.1 数据预处理

由于我们收集到的数据大多杂乱无章且维度较高，无法直接进行挖掘与分析。因此，我们先要对数据进行整理并降维。本次项目在此方面主要涉及以下 2 个算法：词频-逆向文件频率 (term frequency – inverse document frequency / TF-IDF) 与主成分分析 (Principal Component Analysis / PCA)。

2.2.1.1 词频-逆向文件频率 (term frequency–inverse document frequency / TF-IDF)

词频-逆向文件频率 (term frequency – inverse document frequency / TF-IDF) 是一种统计方法，用以评估一个字词对于一个文件集的其中一份文件的重要程度。字词的重要性与其在文件中出现的个数成正相关，但也与包含此字词的文档个数成负相关。其计算方式如下：

$$w_{i,d} = tf_{i,d} \times \log(n / df_i)$$

由公式可以看出，一个词的 TF-IDF 值是由其词频 (Term frequency / TF) 乘以其逆向文件频率 (Inverse document frequency / IDF) 来得到的。具体代码如下：

```
import thulac, zipfile, os, csv, math

def count(zip_name, folder_path=os.getcwd()):
    """Count the times each word appears in each txt file

    Args:

            zip_name: the name of the zip file.

            folder_path: the dictory where stores the zip file, and its defaul
t value is current working dictory.
    """
    dic = dict()
    z = zipfile.ZipFile(folder_path + "/" + zip_name, "r")
    for file_name in z.namelist()[0:6000]:
        if (file_name[-1] == "t" and file_name[0] == "T"):
            txt_name = file_name.split("/")[-1]
            print(txt_name)
            for line in z.open(file_name).readlines():
                l = thulac.thu1.cut(line.decode("utf-8-
sig"), text = True).split(" ")
                for word in l:
                    dic.setdefault(word, {txt_name: 0})
                    dic[word].setdefault(txt_name, 0)
                    dic[word][txt_name] += 1
    file_names(dic)

def file_names(dic):
    """Get all the txt files' names and from a list

    Args:

            dic: the dictionary that stores the count
    """
```

```
    file_name_list = []
    for dics in dic.values():
        file_name_list += dics.keys()
    file_name_list = sorted(list(set(file_name_list)), key=lambda each: int(ea
ch.split(".")[0]))
    draw_table(dic.copy(), file_name_list, "tfidf")
    draw_table(dic.copy(), file_name_list, "tf")

def draw_table(dic, field, mode):
    """Draw TF table or the TFIDF table

    Args:

            dic: the dictionary that stores the count

            field: the list of files' names

            mode: TF table or the TFIDF table
    """
    csv_name = mode + " table_Ch.csv"
    with open(csv_name,'w', newline='', encoding='utf-8-sig') as csvFile:
        writer = csv.writer(csvFile)
        writer.writerow([""] + field)
        n = len(field)
        for items in list(dic.items()):
            word = items[0]
            each_row = [word]
            for name in field:
                if (mode == "tf"):
                    data = items[1].setdefault(name, 0)
                else:
                    df = len(list(items[1].keys()))
                    idf = math.log(n/(df+1))
                    tf = items[1].setdefault(name, 0)
                    data = tf * idf
                each_row.append(data)
            writer.writerow(each_row)

def main():
    """Main method

    Enter the zip file name in the current working dictory to start analysis
```

```
    """
    thulac.thu1 = thulac.thulac(seg_only=True)
    zip_name = input("Please enter the name of the zip file you would like to
analyze: ")
    count(zip_name)
    print("Finish!")

main()
```

其实现步骤如下：

1. 遍历数据集中的每篇文档，使用分词工具进行分词，建立字典来统计每个字词出现的次数以及包含此字词的文档个数
2. 依照 TF-IDF 公式进行计算，并将结果写入 csv 文档

其结果（部分）如下[1]：

		0.txt	1.txt	10.txt	100.txt	101.txt	102.txt	1000.txt	1001.txt	1002.txt	1003.txt	1004.txt	1005.txt	1006.txt	1007.txt	1008.txt	1009.txt	1010.txt	1011.txt	1012.txt	1013.txt	1014.txt	1015.txt	1016.txt	1017.txt
2	马晓旭	14.62511	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	意外	5.763473	0	0	0	0	0	0	2.846645	0	0	0	0	2.823917	0	2.818315	0	0	0	0	0	0	0	2.779954	0
4	受伤	5.539816	1.846605	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	让	1.53548	0.511827	0.511827	0	0	0	0	1.52881	1.019207	1.52881	1.019207	0	1.018096	1.018096	0	9.152882	0.508493	2.542467	2.542467	1.52548	0.508493	0.508493	0	0.507939
6	国奥	12.79253	0	0	0	11.85252	0	0	0	5.607805	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	警惕	11.21561	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	无奈	5.88891	0	8.829317	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	大雨	36.56277	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	格外	9.018386	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	青睐	10.8815	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	股	7.312553	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	家	1.095947	3.287842	0	0	2.187907	1.093953	3.28186	8.751627	7.657674	2.187907	1.093953	1.093953	4.375814	2.187907	0	24.04507	1.092958	5.464789	6.557747	5.464789	0	15.28749	0	4.363879
14	军	5.115329	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15		-0.00167	-0.00367	-0.00634	-0.00434	-0.00467	-0.00233	-0.002	-0.005	-0.00467	-0.00467	-0.002	-0.00233	-0.00267	-0.00233	-0.00133	-0.00867	-0.00167	-0.00967	-0.00434	-0.00434	-0.003	-0.006	-0.003	-0.00334
16	记者	1.434818	4.304453	0	0	0	1.430625	1.430625	4.291875	5.722499	5.722499	0	1.429231	0	1.427839	4.283518	2.855679	1.427839	0	1.426449	0	2.850123	5.700246	1.425062	1.425062
17	傅亚雨	7.312553	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	沈阳	39.31727	0	0	0	0	0	4.709864	9.419728	18.83946	14.12959	0	14.02049	0	0	4.604503	0	13.71514	9.143427	18.28685	0	0	0	8.95868	0
19	报道	1.620506	0	0	0	0	0	3.22422	9.67266	0	1.610439	1.610439	1.610439	0	0	1.607106	3.214211	0	1.605443	1.605443	1.605443	0	0	1.602126	3.204253
20	来	0.713839	1.606138	0.17846	1.784598	0.17846	0.17846	0.535379	1.070759	0.892299	1.249218	0.892299	0.35692	0.892299	1.427678	0	3.561223	0.178061	1.60255	0.534183	0.534183	0	0.532988	0.355325	0.177663
21	到	1.010026	0.252506	1.010026	2.77757	0	0.504154	1.008309	1.512463	2.520771	1.512463	1.76454	1.512463	0.252077	1.260386	1.008309	5.545697	0.504154	2.772848	1.512463	0.504154	1.512463	0.756231	1.512463	0.252077
22	。	0.038044	0.070082	0.074086	0.088103	0.030035	0.046054	0.035041	0.12815	0.078091	0.073085	0.059069	0.042049	0.035041	0.053062	0.029034	0.200234	0.027032	0.102119	0.088103	0.046054	0.050058	0.065076	0.063074	0.052061
23	国奥队	49.71153	0	0	0	0	0	0	11.04159	0	0	0	0	0	0	0	10.12252	0	5.009968	15.02991	0	0	0	0	0
24	依然	1.872303	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.83609	0	0	0	0	1.827757
25	没有	3.379318	1.931039	0.48276	1.448279	0.96552	2.896559	1.448279	0	0	1.445038	0.481679	0	0	1.4418	0.4806	7.689599	0.4806	3.844799	0.4806	0	0.960121	2.400304	0.960121	1.920243
26	摆脱	3.598981	0	0	0	0	0	0	0	0	0	3.494841	0	0	0	0	0	0	0	0	0	0	0	0	0
27	雨水	21.93766	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	的	0.218275	0.457339	0.675614	0.602855	0.207881	0.239063	0.197487	0.758766	0.530097	0.415762	0.623644	0.28064	0.270246	0.311822	0.187093	0.945859	0.187093	0.71719	0.509309	0.415762	0.33261	0.249457	0.467733	0.33261
29	困扰	2.975263	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	。	0.079612	0.119418	0.153538	0.164911	0.062552	0.119418	0.108045	0.284329	0.142165	0.170598	0.187657	0.085299	0.096672	0.102359	0.062552	0.557285	0.022746	0.324135	0.147851	0.102359	0.090985	0.147851	0.125105	0.136478
31	7月	9.829316	19.65863	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	31日	8.584257	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	下午	14.4947	0	0	0	3.58686	0	7.149768	0	0	0	0	0	0	0	0	0	0	3.462406	0	3.451824	0	3.441352	0	0
34	6	1.730495	1.730495	0	0	0	0	0	0	0	6.018013	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	点	5.00132	0	0	0	0	0	0	1.656562	1.656562	0	0	0	1.651331	0	0	0	0	0	0	0	0	0	0	0
36	日常	5.366643	0	5.29765	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	训练	24.28366	1.867974	3.735947	0	0	0	0	0	0	0	0	0	0	0	0	3.684772	0	0	0	0	0	1.831915	0	0
38	再度	2.3853	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	受到	5.260845	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	干扰	7.602016	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
41	之下	3.878566	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
42	队员	8.048072	0	0	5.345964	0	48.02693	0	0	0	5.307685	0	0	0	0	0	0	0	0	0	0	0	10.41209	0	0
43	们	6.011643	7.514553	3.005821	4.508732	0	9.008475	0	0	0	0	0	0	2.984941	0	0	0	0	1.486553	0	0	0	0	0	0
44	只慢	7.312553	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
45	跑	2.468366	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
46	了	0.489444	0.384563	0.349603	0.314643	0.384563	0.454484	0.314643	0.699206	0.734166	0.594325	0.454484	0.419523	0.244722	0.384563	0.174801	0.839047	0.139841	0.699206	0.419523	0.314643	0.174801	0.279682	0.279682	0.279682
47	25	3.959669	0	0	1.975015	0	0	0	0	0	0	0	0	0	0	0	0	1.946577	0	0	0	0	0	0	0
48	分钟	2.096407	0	0	0	0	0	0	0	0	1.04062	0	0	0	0	0	0	0	0	0	0	0	0	0	0
49	就	1.346968	1.346968	0.538787	0.808181	1.077574	0.538787	0.269394	2.155149	1.616361	2.693936	1.616361	0.538787	0.538787	0.808181	0.808181	3.232723	0	2.151656	1.344785	0.268957	2.151656	0.268957	1.613742	0.537914
50	草草收场	6.907088	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
51	上午	8.684278	0	0	0	0	0	4.221511	0	0	4.177059	0	0	0	0	0	0	0	0	0	0	0	0	0	0

算法的优点及缺点：

优点：可将长度不固定的文本转化为长度固定的矩阵

缺点：不对语义进行识别，且需要消耗大量算力与存储空间

[1] 此结果是借助清华大学分词工具 THULAC 对 THUCNews 数据集中前 3000 份文档进行 TF-IDF 计算所得结果的部分截图

2.2.1.2 主成分分析 (Principal Component Analysis / PCA)

主成分分析 (Principal Component Analysis / PCA) 是一种通过正交线性变换将数据变换到新的坐标系统中，使得这一数据的任何投影的第一大方差在第一个坐标（称为第一主成分）上，第二大方差在第二个坐标（第二主成分）上，依次类推，从而实现数据降维的算法。其计算方式如下：

1. 对所有的样本进行“中心化”，即 $x_i = x_i - \frac{1}{m}\sum_{j=1}^{m} x_j$ ，然后将数据按列组成 n 行 m 列矩阵 X′ 等效
2. 计算样本的协方差矩阵：$C = \frac{1}{m} X' X'^{T}$
3. 求出协方差矩阵的特征值及对应的特征向量
4. 将特征向量按对应特征值大小从上到下按行排列成矩阵，取前 k 行组成矩阵 P
5. Y = PX 即为降维到 k 维后的数据

具体代码如下：

```
from sklearn.decomposition import PCA
import csv, numpy as np, matplotlib.pyplot as plt

def draw(data):
    print("Drawing Scatter Plot...")
    plt.suptitle('PCA of tf*idf of THUCNews')
    plt.scatter(data[0], data[1])
    plt.savefig("PCA tfidf_Ch.png")
    plt.show()

def write_file(data):
    print("Writing files...")
    with open('PCA tfidf_Ch.csv','w', newline='', encoding='utf-
8') as csvFile:
```

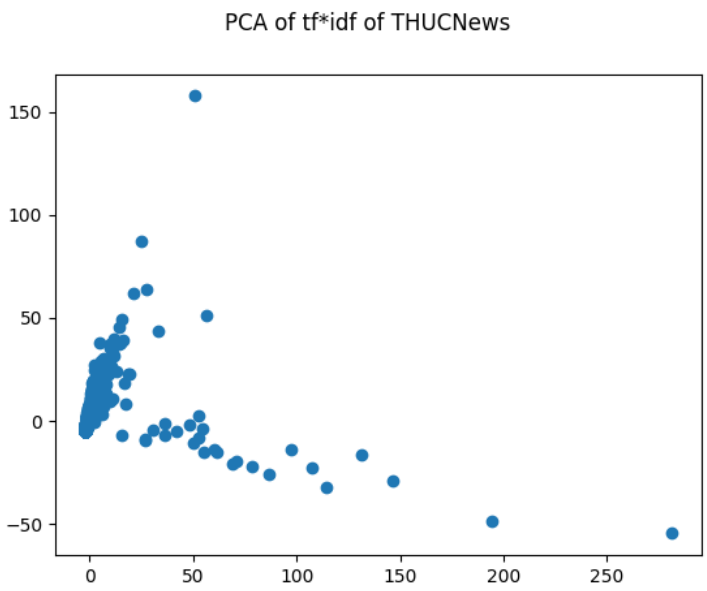
```
        writer = csv.writer(csvFile)
        for rows in data:
            writer.writerow(rows)

def P_C_A(data):
    print("PCAing...")
    pca = PCA(n_components=2)
    pca.fit(data)
    return pca.fit_transform(data)

def read_file(file_name):
    print("Reading files...")
    data = list()
    with open(file_name, encoding='utf-8') as f:
        reader = list(csv.reader(f))
        length = len(reader[0])
        for i in range(length):
            data.append(list())
        for column in reader[1:]:
            for i in range(1, length):
                data[i].append(column[i])
    data = P_C_A(np.array(data[1:], dtype=object))
    write_file(data)
    draw(transform(data))
    k_means(data)
    dbscan(data)

def main():
    read_file(input("Please enter the csv file name: "))
    print("Finish!")

main()
```

其实现步骤如下：

1. 从 csv 文件中读取已经过 TF-IDF 处理后的数据
2. 将数据按列（即逐文档）读取，进行 PCA 降维处理
3. 将降维后的数据写入 csv 文件并进行可视化处理

其结果（部分）以及可视化散点图如下 [2]：

	A	B	C	D
1	6.221454	3.045437		
2	16.97793	18.66154		
3	12.74609	24.24556		
4	18.4248	22.50446		
5	9.828578	9.590963		
6	11.11872	10.78883		
7	86.67167	-25.9618		
8	131.475	-16.5606		
9	146.2972	-29.2684		
10	48.09163	-2.12133		
11	114.204	-32.039		
12	61.18939	-14.9004		
13	69.25765	-20.7908		
14	36.05636	-1.39813		
15	26.74	-8.52018		
16	281.5002	-54.562		
17	26.86601	-9.29418		
18	194.1487	-48.6443		
19	97.19255	-13.6376		
20	54.32918	-3.60076		
21	71.22496	-19.3839		
22	52.53552	2.372504		
23	60.41403	-13.6049		
24	78.50066	-21.9577		
25	30.80233	-4.16661		
26	36.43907	-6.97071		
27	107.5725	-22.4279		
28	15.28056	-6.83541		
29	54.89608	-15.1365		
30	52.41896	-8.10249		
31	41.84612	-5.18861		
32	49.88983	-10.5906		
33	17.11559	8.184754		
34	3.39422	15.8799		
35	2.049816	9.487712		
36	3.774472	14.31551		
37	56.1711	51.5246		
38	1.905737	6.410232		
39	1.401983	1.601867		
40	1.347597	10.03264		
41	4.33419	24.907		
42	0.250421	10.85162		
43	2.040781	24.59816		
44	0.417059	4.749445		
45	10.15248	35.55538		
46	3.230001	12.27909		
47	1.763486	19.80793		
48	2.944261	10.08655		
49	1.489058	8.867424		
50	4.304313	15.54673		
51	-0.88752	1.028208		



算法的优点：

对数据进行特征提取（降维）后便于计算与存储，还可进行可视化处理（仅限降至 2 维或 3 维的数据）

[2] 此结果是对之前 THUCNews 数据集中前 3000 份文档进行 TF-IDF 计算后降至 2 维后的结果的部分截图及可视化

2.2.2 非监督式学习

对大量数据进行分类有助于我们找到数据之间的关系，而非监督学习就是在数据没有标签的情况下，通过对所研究对象的大量样本的数据分析实现对样本分类的一种数据处理方法。

2.2.2.1 聚类 (Cluster)

聚类 (Cluster) 是将对象分到不同的簇中，同一簇内的对象彼此相似，而与其他簇内的对象相异。

K 均值聚类算法 (K-Means)

K 均值聚类算法 (K-Means) 就是将 n 个数据点进行聚类分析，得到 k 个聚类，使得每个数据点到聚类中心的距离最小。其计算方式如下：

1. 随机初始化 k 个聚类中心
2. 将每个数据点分配到其与聚类中心具有最小平方欧氏距离的聚类集合中
3. 计算节点分配有聚类集合的新的样本中心
4. 重复 2、3 两步直到聚类中心不再发生变化，或者是达到一定的迭代次数

具有噪声的基于密度的聚类方法 (Density-Based Spatial Clustering of Applications with Noise / DBSCAN)

具有噪声的基于密度的聚类方法 (Density-Based Spatial Clustering of Applications with Noise / DBSCAN) 是一种基于空间密度的聚类算法。该算法将具有足够密度的区域划分为簇，其计算方式如下：

1. 设定以任何一点为圆心画圆的半径长度 x 和圆内包含的点的最小个数 n
2. 遍历每一个点。如果有个点还未被访问过且以此点为圆心，半径为 x 的圆内有超过 n 个数的点，则创建一个新的簇 C，并将此点添加到 C 中
3. 遍历此圆中的点。如果点还未访问，则标记为已访问，且以此点为圆心，半径为 x 的圆内有超过 n 个数的点，则遍历圆内点。如果此点还未被添加到任何簇，将其添加到 C 中
4. 直到遍历完所有点，此时未被添加到任何簇内的点即为噪声

其代码如下：

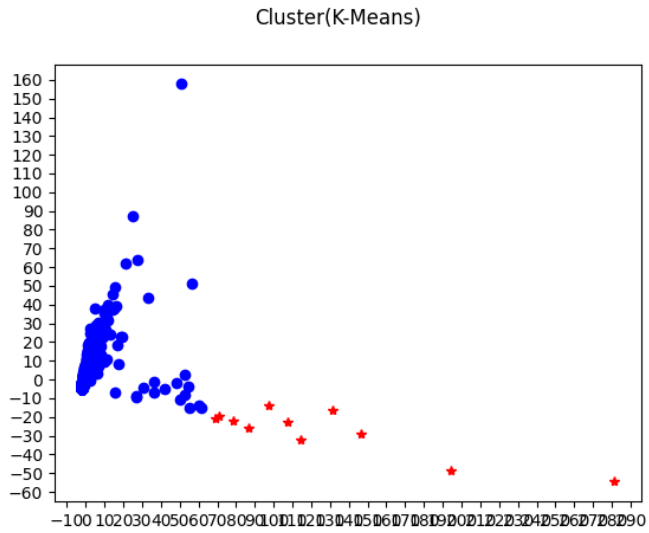
```
from sklearn.cluster import KMeans, DBSCAN
import csv, numpy as np, matplotlib.pyplot as plt

def dbscan(data):
    print("Clustering(DBSCAN)...")
    dbscan_model = DBSCAN(3)
    dbscan_model.fit(data)
    visualize(data, dbscan_model.labels_, "DBSCAN")

def visualize(data, labels, type):
    print("Drawing Clusters("+type+")...")
    plt.suptitle("Cluster("+type+")")
    plt.xticks(np.arange(-20, 300, 10))
    plt.yticks(np.arange(-80, 180, 10))
    for i in range(len(labels)):
        point = data[i]
        if labels[i] == -1:
            plt.plot(point[0],point[1],'k^')
        elif labels[i] == 0:
            plt.plot(point[0],point[1],'bo')
        elif labels[i] == 1:
            plt.plot(point[0],point[1],'r*')
        else:
            plt.plot(point[0],point[1],'yv')
```

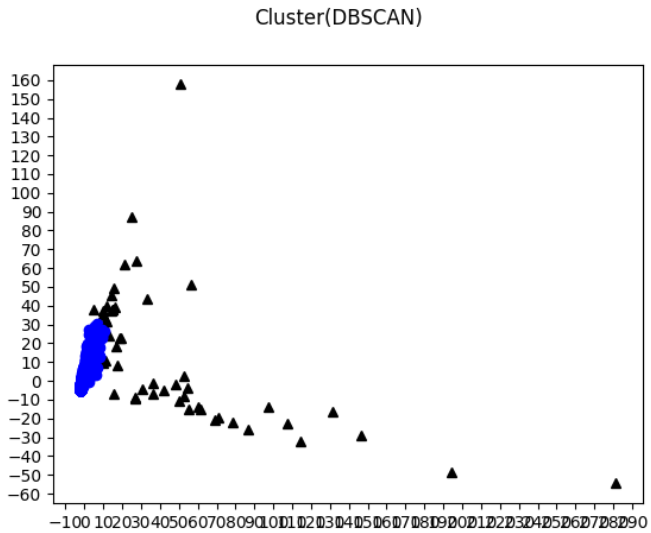
```
    plt.savefig(type + ".png")
    plt.show()

def k_means(data):
    print("Clustering(K-Means)...")
    kmeans_model = KMeans(n_clusters=2)
    kmeans_model.fit(data)
    visualize(data, kmeans_model.labels_, "K-Means")

def read_file(file_name):
    print("Reading files...")
    data = list()
    with open(file_name, encoding='utf-8') as f:
        reader = csv.reader(f)
        for rows in reader:
            data.append(rows)
    k_means(data)
    dbscan(data)

def main():
    read_file(input("Please enter the csv file name: "))
    print("Finish!")

main()
```

其实现步骤如下：

1. 从 csv 文件中读取之前已被降至 2 维的数据
2. 用 K-Means 和 DBSCAN 分别进行聚类
3. 将聚类后的数据可视化

其可视化散点图如下：[3]





算法的缺点：

K-Means 算法初始化聚类中心的位置和 DBSCAN 算法半径以及圆内最小个数的取值对结果有较大影响

[3] 此结果是对之前 THUCNews 数据集中前 3000 份文档进行 TF-IDF 计算后降至 2 维后的结果进行聚类并可视化

2.2.2.2 主题模型 (Topic Model)

主题模型 (Topic Model) 是对文件集的隐藏语义结构进行聚类的统计模型，其弥补了 TF-IDF 模型不对语义进行识别且存储空间及计算量大的缺点。

潜在语义索引 (Latent Semantic Index / LSI) 模型

潜在语义索引 (Latent Semantic Index / LSI) 模型是基于奇异值分解（Singular Value Decomposition / SVD）的方法来得到文本的主题的模型。其计算方式如下：

1. 对 TF-IDF 矩阵 (m 行（词）n 列（文档）) 进行奇异值分解：$X = TSD^T$
2. 矩阵 T 可以看出词和词义之间的相关性，矩阵 D 可以看出文本与主题之间的关系
3. 利用余弦相似度公式，即 $sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\left|\vec{d}_j\right|\left|\vec{d}_k\right|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2}\sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$，计算文本间相似性

其代码如下：

```
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import Normalizer
from sklearn.pipeline import make_pipeline
from sklearn.decomposition import PCA

def transform(data):
    x = []
    y = []
    for row in data:
```

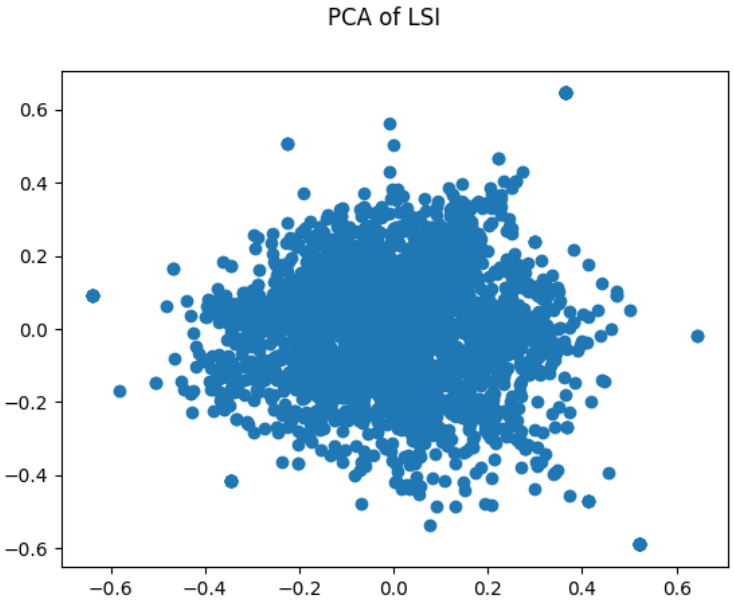
```
        x.append(row[0])
        y.append(row[1])
    return [x,y]

def draw(data):
    print("Drawing Scatter Plot...")
    plt.suptitle('PCA of LSI')
    plt.scatter(data[0], data[1])
    plt.savefig("PCA of LSI.png")
    plt.show()

def P_C_A(data):
    print("PCAing...")
    pca = PCA(n_components=2)
    pca.fit(data)
    draw(transform(pca.fit_transform(data)))

def Lsi(keywords, name_list):
    print("LSIing...")
    svd = TruncatedSVD(n_components=100)
    norm = Normalizer(copy=False)
    lsi = make_pipeline(svd, norm)
    all_topic_vector = lsi.fit_transform(keywords)
    P_C_A(np.array(all_topic_vector, dtype=object))

def tfidf(data, names):
    print("TF*iDFing...")
    vec = TfidfVectorizer(ngram_range=(1, 1), use_idf=True, smooth_idf=True, s
ublinear_tf=True, stop_words=None)
    keywords = vec.fit_transform(data)
    Lsi(keywords, names)

def read_file(file_name):
    print("Reading files...")
    data = list()
    with open(file_name, encoding='utf-8') as f:
        reader = list(csv.reader(f))
        length = len(reader[0])
        for i in range(length):
            data.append("")
        for column in reader[1:]:
            for i in range(1, length):
```

```
                data[i] += str(column[i]) + " "
    tfidf(data[1:], reader[0][1:])


def main():
    read_file(input("Please enter the csv file name: "))
    print("Finish!")

main()
```

其实现步骤如下：

1. 读取存储在 csv 文件中的词频数据
2. 进行 TF-IDF
3. 进行 SVD 计算后训练 LSI 主题模型
4. 进行 PCA 降维以便可视化

其可视化散点图如下：[4]



[5]

[4] 此结果是对之前 THUCNews 数据集中前 3000 份文档进行 TF-IDF 计算后进行 SVD 后降至 2 维的数据可视化

[5] 由于 THUCNews 数据集中前 3000 份文档基本都是与体育相关，因此可视化后只呈现出一簇

算法缺点：

1. 对于高维度矩阵进行 SVD 计算需要消耗大量的时间与算力
2. 主题值的选取对结果的影响非常大，很难选择合适的 k 值
3. LSI 得到的不是一个概率模型，缺乏统计基础，结果难以直观的解释。因此，LSI 主题模型现在已经很少使用，被概率主题模型，如一元模型 (Unigram model) ，一元模型 (Unigram model) ，概率潜在语义索引 (Probabilistic Latent Semantic Index / PLSI) 模型，文档主题生成 (Latent Dirichlet Allocation / LDA) 模型等所取代

2.2.3 监督式学习

监督式学习通过学习带有标签的数据从而建立一个模型，并且依照此模型推测新的数据的标签

2.2.3.1 朴素贝叶斯分类器 (Naïve Bayes Classifier)

朴素贝叶斯分类器 (Naïve Bayes Classifier) 是基于贝叶斯定理，即 P(A|B)=P(A)P(B|A)/P(B)，使用概率统计的知识对样本数据集进行分类。其计算方法如下：

1. 假设特征条件之间相互独立，计算单个条件发生的概率
2. 通过多个单一条件发生概率相乘得出多条件发生的概率

其代码如下：

```
import thulac, csv, random, numpy as np, pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score

def acc(train_data, train_category, test_data, test_category):
    nb_model = MultinomialNB(alpha=0.001)
    nb_model.fit(train_data, train_category)
    nb_predict = nb_model.predict(test_data)
    print("The accuracy of Multinomial Naïve Bayes text classification is: ",
accuracy_score(nb_predict, test_category))

    ber_model = BernoulliNB(alpha=0.001)
    ber_model.fit(train_data, train_category)
    ber_predict = ber_model.predict(test_data)
    print("The accuracy of Bernoulli Naïve Bayes text classification is: ", ac
curacy_score(ber_predict, test_category))


def tfidf(data, test_data):
    print("'tf-idf'ing...")
    document = [" ".join(each) for each in data]
    test_document = [" ".join(each) for each in test_data]
    tfidf_model = TfidfVectorizer(token_pattern=r"(?u)\b\w+\b")
    train = tfidf_model.fit_transform(document)
    test = tfidf_model.transform(test_document)
    return train, test

def read_csv(filename):
    print("Reading files...")
    thulac.thu1 = thulac.thulac(seg_only=True)
    pd_all = pd.read_csv(filename)
    data = list()
    category = list()
    test_data = list()
    test_category = list()
    for moodes in range(3):
        cur_list = list(pd_all[pd_all.label == moodes]["review"])
        for lines in cur_list[0:int(len(cur_list)/10)]:
```

```
            category.append(moodes)
            data.append(thulac.thu1.cut(lines, text=True).split(" "))
        for lines in random.sample(cur_list[int(len(cur_list)/10):], int(len(c
ur_list)/500)):
            test_category.append(moodes)
            test_data.append(thulac.thu1.cut(lines, text=True).split(" "))
    return data, category, test_data, test_category

def main():
    data, category, test_data, test_category = read_csv(input("Please enter
the csv file name: "))
    train, test = tfidf(data, test_data)
    acc(train, category, test, test_category)

main()
```

其实现步骤如下：

1. 从 csv 文件中读取带有标签的数据
2. 将数据分为训练集和测试集，各自进行分词后存储字词与标签
3. 对字词集进行 TF-IDF
4. 对生成的 TF-IDF 值进行朴素贝叶斯计算，训练模型
5. 用模型预测测试集数据的标签，并于实际标签对比，计算出准确率

其结果如下：

如数据集有 2 种不同的标签 [6]：

```
Reading files...
Model loaded succeed
'tf-idf'ing...
The accuracy of Multinomial Naïve Bayes text classification is:  0.7773109243697479
The accuracy of Bernoulli Naïve Bayes text classification is:  0.8109243697478992
```

[6] 情感分类数据集，10 万多条带情感标注；来源于新浪微博，正负向评论约各 5 万条

如数据集中有 3 种不同的标签 [7][8]：

```
Reading files...
Model loaded succeed
'tf-idf'ing...
The accuracy of Multinomial Naïve Bayes text classification is:  0.6661211129296236
The accuracy of Bernoulli Naïve Bayes text classification is:  0.6513911620294599
```

算法的缺点：

朴素贝叶斯分类器默认各条件相互独立，而实际上数据级属性直接往往相互关联，二者可能导致准确率下降

2.2.3.2 决策树模型 (Decision Tree Mode)

决策树模型 (Decision Tree Mode) 一种树状结构，它的每一个叶子结点对应着一个分类，非叶子结点对应着在某个属性上的划分，根据样本在该属性上的不同取值降气划分成若干个子集。其基本原理是通过递归切割的方法来寻找最佳分类标准，进而最终形成规则。

决策树模型通过信息增益准则选择特征，通过计算每个特征的信息增益，选取信息增益较大的特征来增强分类能力。而计算信息增益需要先计算信息熵，其公式如下：

$$Entropy(S) = -\sum_{i=1}^{C} p_i \log(p_i)$$

[7] 情感分类数据集，36 万多条，带情感标注；来源于新浪微博，包含 4 种情感，其中喜悦约 20 万条，愤怒、厌恶、低落各约 5 万条

[8] 此数据集中厌恶与低落数据重复，因此只选取前三类进行训练

而信息增益即经验熵与条件熵的差值，其公式如下：

$$Gain(S,A) = Entropy(S) - \sum_{v \in A} \frac{|S_v|}{|S|} Entropy(S_v)$$

在决策树算法中，ID3 算法在各个节点利用信息增益选择特征，递归构筑决策树。其实现步骤如下：

1. 计算经验熵
2. 计算各个条件熵
3. 计算各个信息增益，并选取信息增益最大的特征进行分支
4. 递归 2、3 两步，直到所有特征的信息增益很小或没有特征可供选择为止

算法的优点与缺点：

优点：决策树便于理解和实现

缺点：容易出现过拟合，即在训练过程中过度拟合了噪音，导致判断测试集数据标签时准确率不佳。可以通过剪枝 (Pruning) 来缓解过拟合。

2.2.3.3 支持向量机 (Support Vector Machines / SVM)

支持向量机 (Support Vector Machines / SVM) 是一种利用特征空间上最大间隔进行分类的二分类模型。其核心计算方法是将原始特征空间映射到更高的维度从而使非线性可分问题转化为线性可分问题。其中计算最大间隔的公式为：

$$\max M = \frac{2}{\|w\|} \Rightarrow \min \frac{1}{2} w^T w$$

由于直接采用上面的公式会产生误差，因此在其基础上引入损失函数构造新的优化问题。新的公式被称为软边界，其公式如下：

$$L_P \equiv \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i - \sum_{i=1}^{l}\alpha_i[y_i(w\cdot x_i + b) - 1 + \xi_i] - \sum_{i=1}^{l}\mu_i\xi_i$$

2.2.4 规律寻找

2.2.4.1 协同过滤 (Collaborative Filtering)

协同过滤 (Collaborative Filtering) 是指利用用户之间的相似性来推导缺省值、推荐相似商品、寻找品味相同的人等。以推导不同用户对电影评分的缺省值举例：

方法一：利用用户之间的相关性计算

首先，我们需要利用一下公式计算用户之间的相关性：

$$w_{u,v} = \frac{\sum_{i\in I}(r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i\in I}(r_{u,i} - \bar{r}_u)^2}\sqrt{\sum_{i\in I}(r_{v,i} - \bar{r}_v)^2}}$$

其次，利用用户间相关性和平均评分计算缺省值：

$$P_{a,i} = \bar{r}_a + \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_u) \cdot w_{a,u}}{\sum_{u \in U} |w_{a,u}|}$$

方法二：利用贝叶斯公式计算

将用户视为独立特征，将评分视为标签，借助贝叶斯公式推导缺省值：

$$class = \underset{j \in classSet}{\arg\max} P(class_j) \prod_o P(X_o = x_o | class_j)$$

2.2.4.2 关联规则 (Association Rule)

关联规则 (Association Rule) 是指从大量数据挖掘中找出的事物之间的相关性。以购物清单分析举例：

首先，为了找出物品之间的规则，我们需要根据用户的购物清单计算规则的支持度：

$$Support(X) = \frac{\#X}{n}$$

$$Support(X \to Y) = \frac{\#(X \cup Y)}{n}$$

其次，为了找出共同出现在用户购物中较多的组合，我们还需要计算规则的可靠性：

$$Confidence(X \to Y) = \frac{\#(X \cup Y)}{\#(X)}$$

或

$$Confidence(X \rightarrow Y) = \frac{Support(X \cup Y)}{Support(X)}$$

最后，支持度较高且可靠性较强的规则即代表用户经常购买的物品集。

算法的缺点：

由于要计算每层所有组合之间的相似度，导致要消耗大量的算力和时间。因此，我们使用先验算法 (Apriori Algorithm) 来缓解计算量，其原理如下：

因为频繁项集的非空子集也一定是频繁的，所以我们只需要寻找最频繁出 1 项集的集合，形成 L_1，再通过 L_1 找出频繁 2 项集的集合 L_2，以此类推，直至找不出频繁 k 项集。其计算方法如下图所示：

C_k: Candidate itemset of size k
L_k: Frequent itemset of size k

$L_1 \leftarrow \{frequent\ items\}$
for (k=1; $L_k \neq \emptyset$; k++)
  $C_{k+1} \leftarrow candidate(L_k)$
  for each transaction t
    $Q \leftarrow \{c \mid c \in C_{k+1} \wedge c \subseteq t\}$
    $count[c] \leftarrow count[c] + 1, \quad \forall c \in Q$
  end for
  $L_{k+1} \leftarrow \{c \mid c \in C_{k+1} \wedge count[c] / N \geq \sigma\}$
end for
return $\bigcup_k L_k$

第三章 总结与收获

作为一名还未学习线性代数的大一新生，在听老师讲解特征提取、分解奇异值等涉及矩阵运算的算法时只能听懂个大概，因此课后要花大量的时间查找资料来完成作业。但是，老师的耐心讲解和课后对作业中出现的问题的答疑给予了我很大的帮助。在此次项目中，我学习到了数据挖掘领域的基础知识，了解了数据挖掘大致的流程框架，同时掌握了如何借助 python 中不同库，如 sklearn, numpy, panda 和 matplotlib 等，进行数据处理、挖掘和可视化。与此同时，在做作业与抓虫的过程中，锻炼了我的自学能力，以及对文档编码、读取以及数据结构等方面更为深入的理解。

此次项目提升了我的编程能力，并且让我对数据挖掘这一领域有了更加深入和全面的了解。我将利用好从此次项目所学的知识和获得的能力，进一步探索此领域，为日后在大数据及人工智能领域的发展做铺垫。