信息检索与数据挖掘

实验报告

Information Retrieval & Data Mining

Homework Report

17 级数据班 单宝迪 201700210069

lwshanbd@gmail.com

鸣谢

特别感谢尹建华老师一学期以来的辛勤付出！

实验一 实验报告

姓名：单宝迪 学号：201700210069 班级：17数据

实验环境和实验时间

实验环境：

- 硬件环境: Intel(R) Core(TM) i7-8550U 16GRAM
- 软件环境: Windows 10 专业版　Python3.7
- IDE: Pycharm　Jupyter-Notebook

实验时间：

- 项目创建时间 2019.9.20
- 项目结束时间 2019.9.24
- 项目报告提交时间 2019.9.27

实验目标

- 在tweets数据集上构建Inverted index
- 实现布尔查询。Boolean Retrieval Model：And, Or ,Not
- 进行查询优化：拓展查询词汇数量

实现过程

Step1 倒排索引的建立

首先，将源数据中的text与tweet id提取出来，为了后续的运行速率，将提取出的数据写入文件中，便于后续读取。Step1的代码如下：

```
f = open('tweets.txt', 'r')
x = open('text.txt', 'w')
  for i in f:

    #得到text

    pr1 = i.split(', "text": "')
    line = pr1[1].split('", "timeStr"')
    text1 = line[0]+"\n"

    #得到id

    pr2 = i.split(', "tweetId": "')
```

```
        line = pr2[1].split('", "errorCode": "')
        id = line[0]
        x.write(id+" "+text1.lower())

    f.close()
    x.close()
```

然后，我们以word作为key，docid列表作为value，以字典的形式生成和储存倒排索引。同时，通过TextBlob库，对倒排索引的结果进行处理，得到最终版的倒排索引。

```
Dict = defaultdict(dict)


def makeDict():
    global Dict

    f = open('file/text.txt', 'r')
    x = open('file/word.txt', 'w')

    for line in f:
        word = TextBlob(line).words.singularize()
        word[0] = Word(word[0])
        for i in word[1:]:

            if i not in Dict:
                Dict[i] = []
                Dict[i].append(word[0])
            else:
                Dict[i].append(word[0])
    for i in Dict:
        Dict[i].sort()

    x.write(str(Dict))
```

Step2 编写布尔查询语句

编写布尔查询的语句，实现两个词的And，Or，Not查询

```
def And(term1, term2):
    global Dict
    answer = []
    if (term1 not in Dict) or (term2 not in Dict):
        return answer
    else:
        i = len(Dict[term1])
        j = len(Dict[term2])
        x = 0
        y = 0
        l1 = Dict[term1]
        l2 = Dict[term2]
        while x < i and y < j:
```

```python
            if l1[x] == l2[y]:
                answer.append(l1[x])
                x += 1
                y += 1
            elif l1[x] < l2[y]:
                x += 1
            else:
                y += 1
        return answer


def Or(term1, term2):
    global Dict
    answer = []
    if (term1 not in Dict) or (term2 not in Dict):
        return answer
    else:
        answer = Dict[term1] + Dict[term2]
        return answer


def Not(term1, term2):
    global Dict
    answer = []
    if term1 not in Dict:
        return answer
    elif term2 not in Dict:
        answer = Dict[term1]
        return answer

    else:
        answer = Dict[term1]
        ANS = []
        for ter in answer:
            if ter not in Dict[term2]:
                ANS.append(ter)
        return ANS
```

3.查询优化

拓展程序，使程序可查询的单词数量达到三个。**特别注意，三个词查询时，需要考虑and和or的顺序。**

备注：Jupyter Notebook文件只是中间形式，实验结果以py文件为准。

实验二 实验报告

姓名：单宝迪 学号：201700210069 班级：17数据

实验环境和实验时间

实验环境:

- 硬件环境: Intel(R) Core(TM) i7-8550U 16GRAM
- 软件环境: Windows 10 专业版　 Python3.7
- IDE: Pycharm　 Jupyter-Notebook

实验时间:

- 项目创建时间 2019.9.27
- 项目结束时间 2019.10.9
- 项目报告提交时间 2019.10.9

实验目标

- 在Homework1.1的基础上实现最基本的Ranked retrieval model
- Use SMART notation: lnc.ltn
- 改进Inverted index

实现过程

1.建立倒排索引

相比于Homework1.1,本次作业的倒排索引需要将doc，变为<docid, td>.因此，倒排索引在之前的基础上做了改进，实现源码如下:

```
x = open('file/word.txt', 'w')
for line in f:
    word = TextBlob(line).words.singularize()
    word[0] = Word(word[0])
    # word[0]是 tweet id
    for i in word[1:]:
        # i=Word(i)
        if i not in Dict:
            #tmp={word[0]:1}
            Dict[i]={}
            Dict[i][word[0]] = 1
        else:
            if word[0] not in Dict[i]:
```

```
                Dict[i][word[0]] = 1
            else:
                Dict[i][word[0]]=Dict[i][word[0]]+1

# print(Dict['may'])
x.write(str(Dict))
x.close()
```

倒排索引结果示例如下:


```
{'house': {'28965792812892160': 2, '29208662060830721': 1, '29251730197712897': 1, '29286944739434496': 1,
'29604332601090048': 1, '29610756999749632': 1, '29610893926993920': 1, '29738513024942080': 1, '29803474820538369': 1,
'29963957057880067': 1, '30257918918000640': 1, '30294939292139520': 1, '30366467505528832': 1, '30394202042925056': 1,
'30497840182591488': 1, '30522653617954816': 1, '30650131585966081': 1, '30674081439285248': 1, '30725193940865024': 1,
'30726678577676288': 1, '30731833859645441': 1, '30747501699010560': 1, '30752887374090240': 1, '30755695221547009': 1,
'30769994920890369': 1, '30770417580904448': 1, '30794785610530816': 1, '30799530383380480': 2, '30806167512948736': 1,
'30810994859053056': 1, '30821340269252609': 1, '30825725640577024': 1, '30835299768602624': 1, '30853143038267392': 1,
'30869628771115008': 1, '30914542103957506': 1, '30965474883801088': 1, '31012345664765952': 1, '31629248502435840': 1,
'32095579853033473': 1, '32117684309073920': 2, '32236618538549249': 1, '32441667235610624': 1, '32460968856391680': 1,
'32786191061356545': 1, '32893223353450496': 1, '32902274934120448': 1, '32933688203288576': 1, '32934634908024832': 1,
'32986475041660928': 1, '33163577296687104': 1, '33194653989732353': 1, '33215612851331072': 1, '33279674582831104': 1,
'33293679170953216': 1, '33755473865875456': 1, '35042688218697728': 1, '35066441501900800': 1, '29735665431441408l': 1,
```


2.计算每篇doc的cosine值

cosine的计算公式为:

$$MAP = \frac{1}{Q}\sum_{j=1}^{|Q|}\frac{1}{m_j}\sum_{k=1}^{m_j} Precision(R_{jk})$$

考虑到计算每个doc的cosine的计算量较大，如果再query时计算，对查询速度有影响，因此，我采用了一次计算出所有文本的cosine值导入文件的方法，process代码如下：

```
S=open('file/cosinelog.txt','w')
Dict1 = {}
Cos={}
for line in f:
    word = TextBlob(line).words.singularize()
    word[0] = Word(word[0])
    # word[0]是 tweet id
    Dict1[word[0]] = {}
    for i in word[1:]:
        # i=Word(i)
        if i not in Dict1[word[0]]:
            Dict1[word[0]][i] = 1
        else:
            Dict1[word[0]][i] = Dict1[word[0]][i] + 1

for i in Dict1:
    ans = 0
    for word in Dict1[i]:
        tmp=1+math.log10(int(Dict1[i][word]))
        ans += tmp**2
    ans=math.sqrt(ans)
    print(ans)
    Cos[i]=ans

S.write(str(Cos))
```

3.计算结果

3.1 计算wtq

考虑到查询方式为lnc,ltn,故需要对query中的词频求log，并乘以其idf。

计算公式为：

$$l(logarithm) = 1 + log(tf_{t,d})$$

$$t(idf) = log\frac{N}{df_t}$$

具体函数实现如下：

```
def wtq(terms, term):
    global Dict
    num = 0
    for i in terms:
        if i == term:
            num += 1
    idf = math.log10(N / len(Dict[term]))
    wtq =  1 + math.log10(num)
    return idf * wtq
```

3.2 查询函数

对于doc中wtd的计算，由于计算量较小，我们将求log和除以length的过程整合到了search函数中。Search函数的实现如下：

```
def Search(terms):
    getDict()
    score = {}
    for w in terms:
        Wtq = wtq(terms, w)
        for i in Dict[w]:
            td = int(Dict[w][i])
            wtd = 1 + math.log10(td)
            if i not in score:
                score[i]=wtd*Wtq
            else:
                score[i]+=wtd*Wtq
    for doc in score:
        score[doc]=score[doc]/cos[doc]
    result = sorted(score.items(), key=lambda x: x[1], reverse=True)
    print("tweeetid            评分")
    for i in result[:10]:
        print(str(i[0])+"   "+str(i[1]))
```

运行示例

```
C:\ProgramData\Anaconda3\python.exe C:/Users/lwsha/PycharmProjects/Information-Retrieval/Homework2/Homework2.py
Search Query >> home house
tweeetid            评分
306065308668542977   1.1724449822011096
31912372620759040   1.0552375433506835
302749853958688769   1.0552375433506835
308569513677426688   0.9741662400790102
308586672587698177   0.9251453048478196
307464444605255680   0.9081374378418183
297502230184083457   0.8436735067337706
30651305655533568   0.7976846039441853
33348131680686080   0.7976846039441853
297596505559273472   0.7976846039441853
```

反思与感悟

通过本次实验，对于倒排索引的构建有了更充分的认识，对于*SMART notation*有了更深的了解。

备注：Jupyter Notebook文件只是中间形式，实验结果以py文件为准。

实验三 实验报告

姓名：单宝迪 学号：201700210069 班级：17数据

实验环境和实验时间

实验环境：

- 硬件环境: Intel(R) Core(TM) i7-8550U 16GRAM
- 软件环境: Windows 10 专业版　 Python3.7
- IDE: Pycharm　 Jupyter-Notebook

实验目标

- 实现以下指标评价，并对HW1.2检索结果进行评价
    - Mean Average Precision (MAP)
    - Mean Reciprocal Rank (MRR)
    - Normalized Discounted Cumulative Gain (NDCG)

实现过程

1.三种算法的计算公式

（1） Mean Average Precision (MAP)

$$MAP = \frac{1}{Q}\sum_{j=1}^{|Q|}\frac{1}{m_j}\sum_{k=1}^{m_j}Precision(R_{jk})$$

（2）Mean Reciprocal Rank (MRR)

$$MRR(Q) = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{RANK}$$

> 假定信息需求$q_i\in Q$对应的所有文档集合为${ d_1,\cdots , d_{mj}},R_{jk}$是返回结果中直到遇见$d_k$后所在位置前的所有文档的集合

（3）Normalized Discounted Cumulative Gain (NDCG)

$$NDCG(Q,K) = \frac{1}{|Q|}\sum_{j=1}^{|Q|}Z_{j,k}\sum_{m=1}^{k}\frac{2^{R(j,m)-1}}{log(1m)}$$

2.补写MRR代码

由于源代码中并不含有MRR算法的代码，因此，编写如下代码以计算MRR：

```
#查阅资料得：
#对于一个query，若第一个正确答案排在第n位，则MRR得分就是 1/n
def MRR_eval(qrels_dict, test_dict, k=100):
    MRR_result = []
    for query in qrels_dict:
        test_result = test_dict[query]
        true_list = set(qrels_dict[query].keys())
        length_use = min(k, len(test_result))
        if length_use <= 0:
            print('query ', query, ' not found test list')
            return []
        i = 0
        for doc_id in test_result[0: length_use]:
            i += 1
            if doc_id in true_list:
                MRR = 1 / i
                MRR_result.append(1 / i)
                print('query', query, ', MRR: ', MRR)
                break
    return np.mean(MRR_result)
```

3.结果展示

```
---------------MAP---------------
query: 171 ,AP: 0.9498040597601832
query: 172 ,AP: 0.3412969283276451
query: 173 ,AP: 0.9978136200716846
query: 174 ,AP: 0.5675347800347801
query: 175 ,AP: 0.38910505836575876
query: 176 ,AP: 0.8274129338771129
query: 177 ,AP: 0.1135214393434572
query: 178 ,AP: 0.46296296296296297
query: 179 ,AP: 0.9711632590609263
query: 180 ,AP: 0.07688990983214608
query: 181 ,AP: 1.0
query: 182 ,AP: 0.19305019305019305
query: 183 ,AP: 0.425531914893617
query: 184 ,AP: 0.5847126267573457
query: 185 ,AP: 0.5654754181248164
query: 186 ,AP: 0.9866600790513834
query: 187 ,AP: 0.9092983086630102
query: 188 ,AP: 0.8035238199833755
query: 189 ,AP: 0.3757153614704371
query: 190 ,AP: 0.9351466010296691
query: 191 ,AP: 0.7974447915680067
query: 192 ,AP: 1.0
query: 193 ,AP: 1.0
```

```
query: 194 ,AP: 0.9770716619981326
query: 195 ,AP: 0.2695417789757412
query: 196 ,AP: 0.9615384615384616
query: 197 ,AP: 0.9989192038173244
query: 198 ,AP: 1.0
query: 199 ,AP: 0.2375296912114014
query: 200 ,AP: 0.3731378521601115
query: 201 ,AP: 0.37037037037037035
query: 202 ,AP: 0.5954593161482736
query: 203 ,AP: 0.7136992484156964
query: 204 ,AP: 0.8990918268187825
query: 205 ,AP: 0.5676666645725785
query: 206 ,AP: 0.9103122265208956
query: 207 ,AP: 0.9607505734124471
query: 208 ,AP: 0.303951367781155
query: 209 ,AP: 0.16447368421052633
query: 210 ,AP: 0.9635344674818358
query: 211 ,AP: 0.25226035126159885
query: 212 ,AP: 0.5650377539391153
query: 213 ,AP: 0.398406374501992
query: 214 ,AP: 0.530280317997534
query: 215 ,AP: 0.30120481927710846
query: 216 ,AP: 0.4269032815167319
query: 217 ,AP: 0.625
query: 218 ,AP: 0.30303030303030304
query: 219 ,AP: 0.25524197520567754
query: 220 ,AP: 0.6138226621145667
query: 221 ,AP: 0.1988071570576541
query: 222 ,AP: 0.30126376980342995
query: 223 ,AP: 0.9940746736049804
query: 224 ,AP: 0.5178732378732379
query: 225 ,AP: 0.9920063553263518
MAP = 0.6148422817122279
--------------NDCG---------------
query 171 , NDCG:  0.9398543518229351
query 172 , NDCG:  0.9522319284335552
query 173 , NDCG:  0.8787194969898994
query 174 , NDCG:  0.4307012038436227
query 175 , NDCG:  0.7551540943184635
query 176 , NDCG:  0.7642638365304593
query 177 , NDCG:  0.32326557235468056
query 178 , NDCG:  0.7937060310666214
query 179 , NDCG:  0.9092261961802077
query 180 , NDCG:  0.384578000794295
query 181 , NDCG:  0.9083280342057781
query 182 , NDCG:  0.877578756577689
query 183 , NDCG:  0.9016059435415619
query 184 , NDCG:  0.7456215828590065
query 185 , NDCG:  0.5651704753561145
query 186 , NDCG:  0.9174314725664987
query 187 , NDCG:  0.8568815395907531
query 188 , NDCG:  0.834462410887587
query 189 , NDCG:  0.11401721726142679
query 190 , NDCG:  0.9087219839232467
query 191 , NDCG:  0.8333343147042753
```

```
query 192 , NDCG:  0.8691210155951211
query 193 , NDCG:  0.870741244990849
query 194 , NDCG:  0.9169177532845512
query 195 , NDCG:  0.7066199310564784
query 196 , NDCG:  0.9661544464181389
query 197 , NDCG:  0.9366145863919296
query 198 , NDCG:  0.8656740779203047
query 199 , NDCG:  0.8150900615927696
query 200 , NDCG:  0.8347275757365947
query 201 , NDCG:  0.8802919036981388
query 202 , NDCG:  0.8455666016685564
query 203 , NDCG:  0.5568543671092813
query 204 , NDCG:  0.8819018257589796
query 205 , NDCG:  0.8851402460821168
query 206 , NDCG:  0.8077691566644618
query 207 , NDCG:  0.8228677166265421
query 208 , NDCG:  0.795113510490801
query 209 , NDCG:  0.6682277350065139
query 210 , NDCG:  0.9144104200186212
query 211 , NDCG:  0.046597135518310455
query 212 , NDCG:  0.8308594376764563
query 213 , NDCG:  1.0
query 214 , NDCG:  0.6916266592407506
query 215 , NDCG:  0.5070939854213776
query 216 , NDCG:  0.7612721037995507
query 217 , NDCG:  0.7675078383310092
query 218 , NDCG:  0.8302203434012001
query 219 , NDCG:  0.498155912259978
query 220 , NDCG:  0.5674800702438964
query 221 , NDCG:  0.9266372064962487
query 222 , NDCG:  0.5087328728028815
query 223 , NDCG:  0.9063275712084274
query 224 , NDCG:  0.3773185814513307
query 225 , NDCG:  0.9706077927297266
NDCG = 0.756819929645465
--------------MRR---------------
query 171 , MRR:  0.5
query 172 , MRR:  1.0
query 173 , MRR:  1.0
query 174 , MRR:  0.2
query 175 , MRR:  1.0
query 176 , MRR:  0.16666666666666666
query 177 , MRR:  1.0
query 178 , MRR:  1.0
query 179 , MRR:  1.0
query 180 , MRR:  0.14285714285714285
query 181 , MRR:  1.0
query 182 , MRR:  1.0
query 183 , MRR:  1.0
query 184 , MRR:  1.0
query 185 , MRR:  0.3333333333333333
query 186 , MRR:  1.0
query 187 , MRR:  0.5
query 188 , MRR:  1.0
query 189 , MRR:  0.16666666666666666
```

```
query 190 , MRR:  0.5
query 191 , MRR:  1.0
query 192 , MRR:  1.0
query 193 , MRR:  1.0
query 194 , MRR:  1.0
query 195 , MRR:  1.0
query 196 , MRR:  1.0
query 197 , MRR:  1.0
query 198 , MRR:  1.0
query 199 , MRR:  1.0
query 200 , MRR:  1.0
query 201 , MRR:  1.0
query 202 , MRR:  1.0
query 203 , MRR:  1.0
query 204 , MRR:  0.5
query 205 , MRR:  1.0
query 206 , MRR:  0.3333333333333333
query 207 , MRR:  1.0
query 208 , MRR:  1.0
query 209 , MRR:  1.0
query 210 , MRR:  1.0
query 211 , MRR:  0.0625
query 212 , MRR:  1.0
query 213 , MRR:  1.0
query 214 , MRR:  0.25
query 215 , MRR:  1.0
query 216 , MRR:  1.0
query 217 , MRR:  1.0
query 218 , MRR:  1.0
query 219 , MRR:  0.3333333333333333
query 220 , MRR:  0.3333333333333333
query 221 , MRR:  1.0
query 222 , MRR:  0.3333333333333333
query 223 , MRR:  1.0
query 224 , MRR:  0.2
query 225 , MRR:  1.0
MRR = 0.79737012987013
```

> 以下为程序完整代码

```python
import math
import numpy as np


def generate_tweetid_gain(file_name):
    qrels_dict = {}
    with open(file_name, 'r', errors='ignore') as f:
        for line in f:
            ele = line.strip().split(' ')
            if ele[0] not in qrels_dict:
                qrels_dict[ele[0]] = {}
            # here we want the gain of doc_id in qrels_dict > 0,
            # so it's sorted values can be IDCG groundtruth
```

```
            if int(ele[3]) > 0:
                qrels_dict[ele[0]][ele[2]] = int(ele[3])
    return qrels_dict


def read_tweetid_test(file_name):
    # input file format
    # query_id doc_id
    # query_id doc_id
    # query_id doc_id
    # ...
    test_dict = {}
    with open(file_name, 'r', errors='ignore') as f:
        for line in f:
            ele = line.strip().split(' ')
            if ele[0] not in test_dict:
                test_dict[ele[0]] = []
            test_dict[ele[0]].append(ele[1])
    return test_dict


def MAP_eval(qrels_dict, test_dict, k=100):
    AP_result = []
    for query in qrels_dict:
        test_result = test_dict[query]
        true_list = set(qrels_dict[query].keys())
        # print(len(true_list))
        # length_use = min(k, len(test_result), len(true_list))
        length_use = min(k, len(test_result))
        if length_use <= 0:
            print('query ', query, ' not found test list')
            return []
        P_result = []
        i = 0
        i_retrieval_true = 0
        for doc_id in test_result[0: length_use]:
            i += 1
            if doc_id in true_list:
                i_retrieval_true += 1
                P_result.append(i_retrieval_true / i)
                # print(i_retrieval_true / i)
        if P_result:
            AP = np.sum(P_result) / len(true_list)
            print('query:', query, ',AP:', AP)
            AP_result.append(AP)
        else:
            print('query:', query, ' not found a true value')
            AP_result.append(0)
    return np.mean(AP_result)


def NDCG_eval(qrels_dict, test_dict, k=100):
    NDCG_result = []
    for query in qrels_dict:
        test_result = test_dict[query]
```

```
        # calculate DCG just need to know the gains of groundtruth
        # that is [2,2,2,1,1,1]
        true_list = list(qrels_dict[query].values())
        true_list = sorted(true_list, reverse=True)
        i = 1
        DCG = 0.0
        IDCG = 0.0
        # maybe k is bigger than arr length
        length_use = min(k, len(test_result), len(true_list))
        if length_use <= 0:
            print('query ', query, ' not found test list')
            return []
        for doc_id in test_result[0: length_use]:
            i += 1
            rel = qrels_dict[query].get(doc_id, 0)
            DCG += (pow(2, rel) - 1) / math.log(i, 2)
            IDCG += (pow(2, true_list[i - 2]) - 1) / math.log(i, 2)
        NDCG = DCG / IDCG
        print('query', query, ', NDCG: ', NDCG)
        NDCG_result.append(NDCG)
    return np.mean(NDCG_result)

#查阅资料得:
#对于一个query，若第一个正确答案排在第n位，则MRR得分就是 1/n
def MRR_eval(qrels_dict, test_dict, k=100):
    MRR_result = []
    for query in qrels_dict:
        test_result = test_dict[query]
        true_list = set(qrels_dict[query].keys())
        length_use = min(k, len(test_result))
        if length_use <= 0:
            print('query ', query, ' not found test list')
            return []
        i = 0
        for doc_id in test_result[0: length_use]:
            i += 1
            if doc_id in true_list:
                MRR = 1 / i
                MRR_result.append(1 / i)
                print('query', query, ', MRR: ', MRR)
                break
    return np.mean(MRR_result)


def evaluation():
    k = 100
    # query relevance file
    file_qrels_path = 'qrels.txt'
    # qrels_dict = {query_id:{doc_id:gain, doc_id:gain, ...}, ...}
    qrels_dict = generate_tweetid_gain(file_qrels_path)
    # ur result, format is in function read_tweetid_test, or u can write by ur own
    file_test_path = 'result.txt'
    # test_dict = {query_id:[doc_id, doc_id, ...], ...}
    test_dict = read_tweetid_test(file_test_path)
    MAP = MAP_eval(qrels_dict, test_dict, k)
```

```python
    print('MAP', ' = ', MAP, sep='')
    NDCG = NDCG_eval(qrels_dict, test_dict, k)
    print('NDCG', ' = ', NDCG, sep='')
    MRR = MRR_eval(qrels_dict, test_dict, k)
    print('MRR', ' = ', MRR, sep='')


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

实验二 实验报告

姓名：单宝迪 学号：201700210069 班级：17数据

实验环境和实验时间

实验环境：

- 硬件环境1: Intel(R) Core(TM) i7-8550U 16GRAM
- 硬件环境2: Intel(R) E5 128GRAM
- 软件环境: Windows 10 专业版　 Python3.7
- IDE: Pycharm　 Jupyter-Notebook

> 备注：基于digits数据的聚类实验在本地PC运行，基于fetch_20newsgroups的数据在服务器上运行。

实验目标

- 测试sklearn中部分聚类算法在digits,fetch_20newsgroups两个数据集上的聚类效果。

实现过程

数据一

1.数据的加载

通过下述代码加载数据集，并读取数据集的有关信息

```
from sklearn import metrics

digits = load_digits()
data = scale(digits.data)

n_samples, n_features = data.shape
n_digits = len(np.unique(digits.target))
labels = digits.target

sample_size = 1797
```

2.聚类操作

由于digits数据属于sklearn标准数据集，对于其聚类操作，仅需要简单的调用api，而不需要进行过多的处理。设置聚类对象的代码如下：

```
bench_k_means(KMeans(init='k-means++', n_clusters=n_digits, n_init=10),
              name="k-means++", data=data)

bench_k_means(KMeans(init='random', n_clusters=n_digits, n_init=10),
              name="random", data=data)

bench_AffinityPropagation(AffinityPropagation(convergence_iter=20),
                          name="AP", data=data)

bench_MeanShift(MeanShift(), name="MeanShift", data=data)


bench_SpectralClustering(SpectralClustering(
```

```
    n_clusters=n_digits), name="Spectral", data=data)

bench_AgglomerativeClustering(AgglomerativeClustering(n_clusters=n_digits, linkage='ward', connectivity=None),
                         name="Ward-hier", data=data)

bench_AgglomerativeClustering(AgglomerativeClustering(n_clusters=n_digits, linkage='complete',
connectivity=None),
                         name="Agglomerative", data=data)

bench_DBSCAN(DBSCAN(eps=5, min_samples=3), name="DBSCAN", data=data)

bench_GaussianMixture(mixture.GaussianMixture(n_components=n_digits, covariance_type='full'),
                      name="GaussMix", data=data)
```

3.聚类评估

经过运行对应代码，得到了如下的Evaluation：

init	time	inertia	NMI	Homogeneity	Completeness
k-means++	1.58s	69432	0.626	0.602	0.650
random	0.33s	69694	0.689	0.669	0.710
AP	13.30s	89	0.655	0.932	0.460
MeanShift	9.69s	unknown	0.063	0.014	0.281
Spectral	578.17s	unknown	0.012	0.001	0.271
Ward-hier	0.52s	unknown	0.797	0.758	0.836
Agglomerative	0.65s	unknown	0.065	0.017	0.249
DBSCAN	1.01s	unknown	1.000	1.000	1.000
GaussMix	0.89s	16	0.642	0.610	0.676
PCA-based	0.10s	70804	0.685	0.671	0.698

数据二

1.数据的加载

```
import logging
from optparse import OptionParser
import sys
from time import time

import numpy as np
name=[]
NMI=[]
Homogeneity=[]
Completeness=[]
# Display progress logs on stdout
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s %(levelname)s %(message)s')

op = OptionParser()
op.add_option("--lsa",
              dest="n_components", type="int",
              help="Preprocess documents with latent semantic analysis.")
op.add_option("--no-minibatch",
              action="store_false", dest="minibatch", default=False,
              help="Use ordinary k-means algorithm (in batch mode).")
op.add_option("--no-idf",
```

```python
              action="store_false", dest="use_idf", default=True,
              help="Disable Inverse Document Frequency feature weighting.")
op.add_option("--use-hashing",
              action="store_true", default=False,
              help="Use a hashing feature vectorizer")
op.add_option("--n-features", type=int, default=10000,
              help="Maximum number of features (dimensions)"
                   " to extract from text.")
op.add_option("--verbose",
              action="store_true", dest="verbose", default=False,
              help="Print progress reports inside k-means algorithm.")


def is_interactive():
    return not hasattr(sys.modules['__main__'], '__file__')


argv = [] if is_interactive() else sys.argv[1:]
(opts, args) = op.parse_args(argv)
if len(args) > 0:
    op.error("this script takes no arguments.")
    sys.exit(1)



categories = [
    'alt.atheism',
    'talk.religion.misc',
    'comp.graphics',
    'sci.space',
]

print("Loading 20 newsgroups dataset for categories:")
print(categories)

dataset = fetch_20newsgroups(subset='all', categories=categories,
                             shuffle=True, random_state=42)

print("%d documents" % len(dataset.data))
print("%d categories" % len(dataset.target_names))
print()

labels = dataset.target
true_k = np.unique(labels).shape[0]

print("Extracting features from the training dataset "
      "using a sparse vectorizer")
t0 = time()
if opts.use_hashing:
    if opts.use_idf:
        hasher = HashingVectorizer(n_features=opts.n_features,
                                   stop_words='english', alternate_sign=False,
                                   norm=None, binary=False)
        vectorizer = make_pipeline(hasher, TfidfTransformer())
    else:
        vectorizer = HashingVectorizer(n_features=opts.n_features,
                                       stop_words='english',
                                       alternate_sign=False, norm='l2',
                                       binary=False)
else:
    vectorizer = TfidfVectorizer(max_df=0.5, max_features=opts.n_features,
                                 min_df=2, stop_words='english',
                                 use_idf=opts.use_idf)
X = vectorizer.fit_transform(dataset.data)

print("done in %fs" % (time() - t0))
print("n_samples: %d, n_features: %d" % X.shape)
print()
```

```
if opts.n_components:
    print("Performing dimensionality reduction using LSA")
    t0 = time()
    svd = TruncatedSVD(opts.n_components)
    normalizer = Normalizer(copy=False)
    lsa = make_pipeline(svd, normalizer)

    X = lsa.fit_transform(X)

    print("done in %fs" % (time() - t0))

    explained_variance = svd.explained_variance_ratio_.sum()
    print("Explained variance of the SVD step: {}%".format(
        int(explained_variance * 100)))

    print()
```

2.聚类操作

根据Scikit-Learn官方文档提供的操作，通过更改Api接口的调用，实现多个聚类方法的实现。

Kmeans

```
km = KMeans(n_clusters=true_k, init='k-means++', max_iter=100, n_init=1,
            verbose=opts.verbose)

print("Clustering sparse data with %s" % km)
t0 = time()
km.fit(X)
#print(km.cluster_centers_)
print("done in %0.3fs" % (time() - t0))
```

DBSCAN

```
km = DBSCAN(eps=5, min_samples=3)

print("Clustering sparse data with %s" % km)
t0 = time()
km.fit(X)
#print(km.cluster_centers_)
print("done in %0.3fs" % (time() - t0))
```

AffinityPropagation

```
km = AffinityPropagation(convergence_iter=20)

print("Clustering sparse data with %s" % km)
t0 = time()
km.fit(X)
#print(km.cluster_centers_)
print("done in %0.3fs" % (time() - t0))
```

Ward hierarchical clustering

```
km = AgglomerativeClustering(n_clusters=4, linkage='ward',connectivity=None)

print("Clustering sparse data with %s" % km)
t0 = time()
km.fit(X)
```

```
#print(km.cluster_centers_)
print("done in %0.3fs" % (time() - t0))
```

AgglomerativeClustering

```
km = AgglomerativeClustering(n_clusters=4, linkage='complete',connectivity=None)

print("Clustering sparse data with %s" % km)
t0 = time()
km.fit(X)
#print(km.cluster_centers_)
print("done in %0.3fs" % (time() - t0))
```

GaussianMixture

```
from sklearn import mixture
km = mixture.GaussianMixture(n_components=4, covariance_type='full')

print("Clustering sparse data with %s" % km)
t0 = time()
km.fit(X)
#print(km.cluster_centers_)
print("done in %0.3fs" % (time() - t0))
```

MeanShift

经实验，MeanShift不适用于文本聚类

3.聚类评估

init	time	NMI	Homogeneity	Completeness
k-means++	4.13s	0.426	0.426	0.529
AP	11.870s	0.885	0.885	0.191
Spectral	578.17s	0.012	0.001	0.271
Ward-hier	46.737s	0.797	0.758	0.836
Agglomerative	47.260s	0.066	0.064	0.068
DBSCAN	295.972s	0	0	1.000
GaussMix	181.503s	0.569	0.534	0.607

备注：Jupyter Notebook文件只是中间形式，实验结果以py文件为准。