分布式系统lab2

设计逻辑

- 总的来说，首先将.dat文件转换成document.txt文件，一个文件对应document中的一行
- 然后使用MapReduce统计词频
  - 每个mapper处理一行，计算tf
  - 每个reducer处理一个词语，计算idf和tf-idf
  - 最后将结果储存在result.txt
- 关键词查询
  - 读取result.txt内容，储存成字典，在字典内查询并得到相关文件

数据处理

```
需要将.dat文件转换成MapReduce可以处理的.txt文件
```

- 数据特点
  - 待处理的文本数据 document.dat 使用 gb18030 编码格式，使用以下函数打开：

    ```
    with open('document.dat', 'r', encoding='gb18030') as input_file:
        content = input_file.read()
    ```

  - 文档中对应若干条数据，每条数据格式如下：

    ```
    <doc>
    <url>http://news.cn.yahoo.com/newspic/news/24978/1/</url>
    <docno>67b4d6a64aad368d-8c437189a1acd500</docno>
    <contenttitle>广西都安：暴雨袭击万人受灾（</contenttitle>
    <content></content>
    </doc>
    ```

  - 只有中的内容可以用来做词频分析，用docno来识别不同的文档
- 数据处理
  - 每条数据生成一个行，格式为docno + " " + contenttitle+" " + content
  - 使用正则表达式匹配来得到标签中的内容，并传入txt文件

    ```
    doc_pattern = re.compile(r'<doc>.*?</doc>', re.DOTALL)
    ```

```
docs = doc_pattern.findall(content)
contents = []

for doc in docs:
    docno_match = re.search(r'<docno>(.*?)</docno>', doc)
    contenttitle_match = re.search(r'<contenttitle>(.*?)
</contenttitle>', doc)
    content_match = re.search(r'<content>(.*?)</content>', doc)

    if docno_match and contenttitle_match:
        docno = docno_match.group(1)
        contenttitle = contenttitle_match.group(1)
        content = content_match.group(1) if content_match else ""
with open(f'document.txt', 'w', encoding='utf-8') as output_file:
    for content in contents:
        output_file.write(content + '\n')
```

TF-IDF文档的生成

- 使用 MapReduce 框架，在 Map 阶段计算某一文档的TF，在 Reduce 阶段计算IDF和最终的TF-IDF
- Map阶段
  - 每个Map处理一行line（本质上是一个文件）
  - 使用jieba分词后分别统计词数，计算tf
  - 每个word yield一次，同时包含文件名和该word的tf值

```
def mapper(self, _, line):
    filename, content = line.split(' ', 1)

    # 使用jieba分词
    words = list(jieba.cut(line))
    word_count = len(words)

    # Emit word-document pairs with term frequency
    for word in set(words):
        yield word, (filename, words.count(word) / word_count)
```

- Reduce阶段
  - 接收到的是已经被整合过的数据，values为(filename, words.count(word) / word_count)组成的列表
  - 需要知道一共有多少文件Y，本质上是判断document.txt的行数，在MapReduce开始前完

成

```
@classmethod
def run(cls):
    # 在 TFIDFJob.run() 之前，统计文档的总行数
    with open('document.txt', 'r', encoding='utf-8') as file:
        cls.total_lines = sum(1 for _ in file)

    # 调用父类的 run 方法，启动 MapReduce 作业
    super(TFIDFJob, cls).run()
```

- 需要计算idf，列表长度Yw就是word在多少文件中出现

```
document_list = list(values)
document_frequency = len(document_list)

# Calculate inverse document frequency (IDF)
idf = math.log(self.total_lines / (1 + document_frequency))
```

- 然后针对这个word所在的每个文件，分别计算tf-idf并将结果输出

```
# Calculate TF-IDF for each document
tfidf_values = []
for doc, tf in document_list:
    tfidf = tf * idf
    tfidf_values.append([doc, tfidf])

yield word, tfidf_values
```

关键词查询

- MapReduce的结果储存在result.txt中，格式为Keyword [[Document1, TF-IDF1], [Document2, TF-IDF2],...]
- 从result.txt中读取数据，储存为dict结构（keyword_dict），其中key为word，value为Document组成的列表

```
for line in contents:
    if(len(line) == 0):
        break
    word, tfidf_list = line.split("\t", 1)
    my_list = ast.literal_eval(tfidf_list)
    keyword_dict[word] = [sublist[0]+'.txt' for sublist in my_list]
```

- 循环读取待查询的词，在不存在时提示没有这个词，存在时返回结果列表

```
while(1):
    word = input(">>> ")
    if(word == "quit"):
        break
    if word in keyword_dict:
        print(keyword_dict[word])
    else:
        print("No such word")
```

结果展示

tf-idf文档

- result.txt，以unicode编码
- 储存形式为：

```
"\u6807\u8bed\u724c"    [["2f8ab9714bad368d-8c437189a1acd500", 0.018796613004033 03]]
"\u6811\u679d"  [["f3c3df47b9ad368d-8c437189a1acd500", 0.022854442610362 74]]
"\u6811\u7acb"  [["0f07319f95e7bc78-49f37189a1acd500", 0.005017568790776 4955], ["a286bd3d96
"\u6821\u56ed\u7f51"    [["e3736c5949da2142-69713306c0bb3300", 0.004573158079432068]]
"\u6821\u5e86"  [["62ee74314bad368d-8c437189a1acd500", 0.075494593212919 54]]
"\u6821\u957f"  [["db5a937a94e7bc78-49f37189a1acd500", 0.004962467872831996]]
"\u682a\u5f0f\u4f1a\u793e"  [["ccb4334497e7bc78-49f37189a1acd500", 0.060594344552474 89]]
"\u6837\u54c1"  [["0f07319f95e7bc78-49f37189a1acd500", 0.011003990886470947]]
"\u6837\u5f0f"  [["199f57b44aad368d-8c437189a1acd500", 0.008771752735215414]]
```

关键词查询

- 代码结果

```
>>> 武汉
['35d7097793e7bc78-49f37189a1acd500.txt', '5d3f170796e7bc78-49f37189a1acd500.txt', '65d2a0bf4aad368d-8c437189a1acd500.txt',
>>> 武器
['c172394d49da2142-69713306c0bb3300.txt', '3155b08a4aad368d-8c437189a1acd500.txt', '642d2a54b9ad368d-8c437189a1acd500.txt']
>>> 你好
['dbb4554e49da2142-69713306c0bb3300.txt', '53de3b2994e7bc78-49f37189a1acd500.txt']
>>> 不
['dbb4554e49da2142-69713306c0bb3300.txt', 'e4103f4f49da2142-69713306c0bb3300.txt', 'b610ca6149da2142-69713306c0bb3300.txt',
```