**PythonTF-IDF算法对文本进行统计词频**

**相关知识**

TF-IDF（term frequency–inverse document frequency）是一种用于资讯检索与资讯探勘的常用加权技术。TF-IDF是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。TF-IDF加权的各种形式常被搜寻引擎应用，作为文件与用户查询之间相关程度的度量或评级。除了TF-IDF以外，因特网上的搜寻引擎还会使用基于连结分析的评级方法，以确定文件在搜寻结果中出现的顺序。

在一份给定的文件里，词频 (term frequency, TF) 指的是某一个给定的词语在该文件中出现的次数。这个数字通常会被归一化（分子一般小于分母 区别于IDF），以防止它偏向长的文件。（同一个词语在长文件里可能会比短文件有更高的词频，而不管该词语重要与否。）

逆向文件频率 (inverse document frequency, IDF) 是一个词语普遍重要性的度量。某一特定词语的IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到。

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的TF-IDF。因此，TF-IDF倾向于过滤掉常见的词语，保留重要的词语。

[](https://www.ipieuvre.com/doc/exper/285d757a-91ad-11e9-beeb-00215ec892f4/img/01.png)

sklearn中使用 TfidfTransformer的默认设置，TfidfTransformer(norm='l2', use\_idf=True, smooth\_idf=True, sublinear\_tf=False) 统计词频，词出现在给定文档的数量乘以该 idf值，idf值为：

[](https://www.ipieuvre.com/doc/exper/285d757a-91ad-11e9-beeb-00215ec892f4/img/02.png)

这里

[](https://www.ipieuvre.com/doc/exper/285d757a-91ad-11e9-beeb-00215ec892f4/img/03.png)

是文档总数 ， 并且

[](https://www.ipieuvre.com/doc/exper/285d757a-91ad-11e9-beeb-00215ec892f4/img/04.png)

是包含 t的文档数量。 tf-idf向量的结果被规范化通过欧几里得范数Euclidean norm:

[](https://www.ipieuvre.com/doc/exper/285d757a-91ad-11e9-beeb-00215ec892f4/img/05.png)

scikit-learn中 TfidfTransformer 与 TfidfVectorizer， idf 设置为：

[](https://www.ipieuvre.com/doc/exper/285d757a-91ad-11e9-beeb-00215ec892f4/img/06.png)

在 TfidfTransformer 和 TfidfVectorizer 设置smooth\_idf=False, 此时idf为：

[](https://www.ipieuvre.com/doc/exper/285d757a-91ad-11e9-beeb-00215ec892f4/img/07.png)

**系统环境**

Linux Ubuntu 16.04

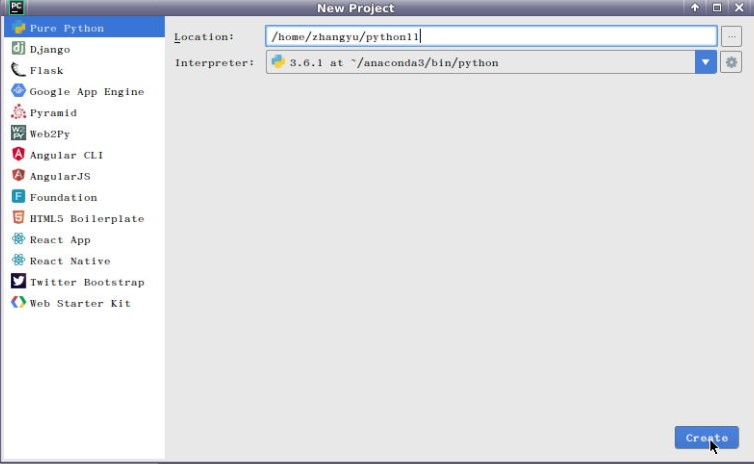
Python3.6

**任务内容**

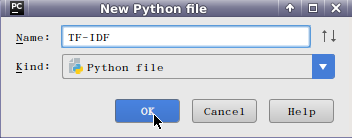
tf-idf统计词频

**任务步骤**

1.新建Python project，名为python11.

[](https://www.ipieuvre.com/doc/exper/285d757a-91ad-11e9-beeb-00215ec892f4/img/09.jpg)

在python11下，新建Python file，名为TF-IDF

[](https://www.ipieuvre.com/doc/exper/285d757a-91ad-11e9-beeb-00215ec892f4/img/08.png)

2.导入实验所需的外包

[view plain](https://www.ipieuvre.com/e/180/311/9539) [copy](https://www.ipieuvre.com/e/180/311/9539)

1. from sklearn.feature\_extraction.text **import** TfidfTransformer
2. from sklearn.feature\_extraction.text **import** CountVectorizer

3.自定义文本

[view plain](https://www.ipieuvre.com/e/180/311/9539) [copy](https://www.ipieuvre.com/e/180/311/9539)

1. corpus = ["我 来到 北京 清华大学",  # 第一类文本切词后的结果，词之间以空格隔开
2. "他 来到 了 网易 杭研 大厦",  # 第二类文本的切词结果
3. "小明 硕士 毕业 与 中国 科学院",  # 第三类文本的切词结果
4. "我 爱 北京 天安门"]  # 第四类文本的切词结果

4.将文本中的词语转换为词频矩阵

[view plain](https://www.ipieuvre.com/e/180/311/9539) [copy](https://www.ipieuvre.com/e/180/311/9539)

1. vectorizer = CountVectorizer()
2. vectors=vectorizer.fit\_transform(corpus)

5.该类会统计每个词语的tf-idf权值

[view plain](https://www.ipieuvre.com/e/180/311/9539) [copy](https://www.ipieuvre.com/e/180/311/9539)

1. transformer = TfidfTransformer()
2. tfidf = transformer.fit\_transform(vectors)

6.获取词袋模型(vectorizer)中的所有词语

[view plain](https://www.ipieuvre.com/e/180/311/9539) [copy](https://www.ipieuvre.com/e/180/311/9539)

1. word = vectorizer.get\_feature\_names()

7.将tf-idf矩阵抽取出来，元素a[i][j]表示j词在i类文本中的tf-idf权重

[view plain](https://www.ipieuvre.com/e/180/311/9539) [copy](https://www.ipieuvre.com/e/180/311/9539)

1. weight = tfidf.toarray()

8.打印每类文本的tf-idf词语权重，第一个for遍历所有文本，第二个for遍历某一类文本下的词语权重

[view plain](https://www.ipieuvre.com/e/180/311/9539) [copy](https://www.ipieuvre.com/e/180/311/9539)

1. **for** i **in** range(len(weight)):
2. print("-------这里输出第", i, "类文本的词语tf-idf权重------")
3. **for** j **in** range(len(word)):
4. print(word[j], weight[i][j])

9.完整代码

[view plain](https://www.ipieuvre.com/e/180/311/9539) [copy](https://www.ipieuvre.com/e/180/311/9539)

1. from sklearn.feature\_extraction.text **import** TfidfTransformer
2. from sklearn.feature\_extraction.text **import** CountVectorizer
3. **if** \_\_name\_\_ == "\_\_main\_\_":
4. corpus = ["我 来到 北京 清华大学",  # 第一类文本切词后的结果，词之间以空格隔开
5. "他 来到 了 网易 杭研 大厦",  # 第二类文本的切词结果
6. "小明 硕士 毕业 与 中国 科学院",  # 第三类文本的切词结果
7. "我 爱 北京 天安门"]  # 第四类文本的切词结果
8. vectorizer = CountVectorizer()  # 该类会将文本中的词语转换为词频矩阵，矩阵元素a[i][j] 表示j词在i类文本下的词频
9. transformer = TfidfTransformer()  # 该类会统计每个词语的tf-idf权值
10. tfidf = transformer.fit\_transform(
11. vectorizer.fit\_transform(corpus))  # 第一个fit\_transform是计算tf-idf，第二个fit\_transform是将文本转为词频矩阵
12. word = vectorizer.get\_feature\_names()  # 获取词袋模型中的所有词语
13. weight = tfidf.toarray()  # 将tf-idf矩阵抽取出来，元素a[i][j]表示j词在i类文本中的tf-idf权重
14. **for** i **in** range(len(weight)):  # 打印每类文本的tf-idf词语权重，第一个**for**遍历所有文本，第二个**for**遍历某一类文本下的词语权重
15. print("-------这里输出第", i, "类文本的词语tf-idf权重------")
16. **for** j **in** range(len(word)):
17. print(word[j], weight[i][j])

运行结果：

[view plain](https://www.ipieuvre.com/e/180/311/9539) [copy](https://www.ipieuvre.com/e/180/311/9539)

1. -------这里输出第 0 类文本的词语tf-idf权重------
2. 中国 0.0
3. 北京 0.52640543361
4. 大厦 0.0
5. 天安门 0.0
6. 小明 0.0
7. 来到 0.52640543361
8. 杭研 0.0
9. 毕业 0.0
10. 清华大学 0.66767854461
11. 硕士 0.0
12. 科学院 0.0
13. 网易 0.0
14. -------这里输出第 1 类文本的词语tf-idf权重------
15. 中国 0.0
16. 北京 0.0
17. 大厦 0.525472749264
18. 天安门 0.0
19. 小明 0.0
20. 来到 0.414288751166
21. 杭研 0.525472749264
22. 毕业 0.0
23. 清华大学 0.0
24. 硕士 0.0
25. 科学院 0.0
26. 网易 0.525472749264
27. -------这里输出第 2 类文本的词语tf-idf权重------
28. 中国 0.4472135955
29. 北京 0.0
30. 大厦 0.0
31. 天安门 0.0
32. 小明 0.4472135955
33. 来到 0.0
34. 杭研 0.0
35. 毕业 0.4472135955
36. 清华大学 0.0
37. 硕士 0.4472135955
38. 科学院 0.4472135955
39. 网易 0.0
40. -------这里输出第 3 类文本的词语tf-idf权重------
41. 中国 0.0
42. 北京 0.61913029649
43. 大厦 0.0
44. 天安门 0.78528827571
45. 小明 0.0
46. 来到 0.0
47. 杭研 0.0
48. 毕业 0.0
49. 清华大学 0.0
50. 硕士 0.0
51. 科学院 0.0
52. 网易 0.0