

文件指纹说明文档

1.文件指纹数据结构

```
1 class FileHash:
2     def __init__(self):
3         # domain_pred_res: 领域预测结果, str
4         # domain_pred_match_num: 领域预测结果的置信度, float
5         # words: 文件名 / 标题关键词提取结果, set 其中不包括跨领域词
6         # domain_matched: 各领域下匹配到的文本关键词, dict {领域: [文本关键词, 匹配
7             的最佳领域词, 相似度]} 其中包括了非预测领域的结果
8         # use_other 是否使用了other
9         # file_key_word 存储schema和对应schema的结果 为string:list list中为提取出
10            的信息
11
12         self.domain_pred = ""
13         self.domain_pred_match_num = 0.0
14         self.words = set()
15         self.domain_matched = {}
16         self.use_other = False
17         self.file_key_word = dict()
18
19         self.hash1 = generate_ngram_lsh_fingerprint("hash1")
20         self.hash2 = generate_ngram_lsh_fingerprint("hash2")
21         self.hash3 = {}
22         # hash3的具体结构为 {领域:[预测关键词, 已提取关键词]}
23         self.hash3All = generate_ngram_lsh_fingerprint("hash3")
```

Tips1: hash3对应的原始数据在代码中以独立数组+参数返回值的方式进行处理, 而没有直接耦合到指纹计算部分, 主要原因是其对应的变量为multiprocessing生成的多线程变量, 其在赋值dict时存在问题, 对多层dict和复杂结构的dict赋值会出现失效情况

可参考[python多线程变量Manager.dict\(\)深度赋值无效问题解决 Sp4rkW的博客-CSDN博客](#)
[multiprocessing --- 基于进程的并行 — Python 3.11.4 文档](#)

如果指涉对象包含了普通 `list` 或 `dict` 对象, 对这些内部可变对象的修改不会通过管理器传播, 因为代理无法得知被包含的值什么时候被修改了。但是把存放在容器代理中的值本身是会通过管理器传播的(会触发代理对象中的 `__setitem__`) 从而有效修改这些对象, 所以可以把修改过的值重新赋值给容器代理:

Tips2: generate_ngram_lsh_fingerprint函数的返回值本身是一个ndarray数组, 而非常规的一串字符串, 因此对于hash1等量的初始化不可取消

2. 文件相似度定义

```
1 def generate_ngram_lsh_fingerprint(text, n=1, num_perm=128, threshold=0.5):
2     """
3     使用N-gram + LSH算法生成文本的指纹(哈希值)。
4     :param text: 要生成指纹的文本。
5     :param n: N-gram的大小。
6     :param num_perm: MinHash的排列数。
7     :param threshold: LSH的阈值。
8     :return: 文本的指纹(哈希值)。
```

```

9      """
10     # 创建MinHash对象
11     minhash = MinHash(num_perm=num_perm)
12     # 创建哈希函数（可以选择MD5、SHA-1等哈希算法）
13     hash_function = hashlib.md5
14     # 生成N-gram序列
15     ngram_sequence = ngrams(text, n)
16     # 添加N-gram序列到MinHash中
17     for ngram in ngram_sequence:
18         for item in ngram:
19             minhash.update(hash_function(item.encode()).digest())
20     # 返回指纹的哈希摘要
21     fingerprint_digest = minhash.digest()
22     return fingerprint_digest
23
24
25 def hash_fingerprint(fingerprint):
26     """
27     使用哈希函数对指纹进行进一步的处理。
28     :param fingerprint: 要处理的指纹。
29     :return: 处理后的指纹（哈希值）。
30     """
31     # 创建哈希函数（可以选择MD5、SHA-1、SHA-256等哈希算法）
32     hash_function = hashlib.md5
33     # 计算指纹的哈希值
34     hashed_fingerprint = hash_function(fingerprint).hexdigest()
35     return hashed_fingerprint
36
37
38 def hamming_distance(fingerprint1, fingerprint2):
39     """
40     计算两个指纹之间的汉明距离。
41     :param fingerprint1: 第一个指纹。
42     :param fingerprint2: 第二个指纹。
43     :return: 汉明距离。
44     """
45     distance = sum(c1 != c2 for c1, c2 in zip(fingerprint1, fingerprint2))
46     return distance
47
48
49 def calculate_similarity(fingerprint1, fingerprint2):
50     """
51     计算两个指纹之间的相似度。
52     :param fingerprint1: 第一个指纹。
53     :param fingerprint2: 第二个指纹。
54     :return: 相似度。
55     """
56     hamming_dist = hamming_distance(fingerprint1, fingerprint2)
57     similarity = 1 - (hamming_dist / len(fingerprint1))
58     return similarity
59
60 def sort(file_hash1, file_hash2, dict1, dict2):
61     """
62     计算两个文件之间的总相似度。
63     :param file_hash1: 存储了第一个文件相关hash信息的块 FileHash

```

```

64 :param file_hash2: 存储了第二个文件相关hash信息的块 FileHash
65 :param dict1: 存储了第一个文件相关hash3信息的字典 dict
66 :param dict2: 存储了第二个文件相关hash3信息的字典 dict
67 :return: 相似度。
68 """
69 similarity1 = calculate_similarity(file_hash1.hash1, file_hash2.hash1)
70 similarity2 = calculate_similarity(file_hash1.hash2, file_hash2.hash2)
71 three = 0
72 three_all = 0
73 four = 0
74 op1 = 0
75 op2 = 0
76 for hash_item1 in dict1:
77     for hash_item2 in dict2:
78         if hash_item1 == hash_item2:
79             a = len(dict1[hash_item1])
80             b = len(dict2[hash_item2])
81             op1 = op1 + (a <= b and a or b)
82             op2 = op2 + (a >= b and a or b)
83             for tmp1 in dict1[hash_item1]:
84                 for tmp2 in dict2[hash_item2]:
85                     if (tmp1[0] == tmp2[0]).all():
86                         three += calculate_similarity(tmp1[1], tmp2[1])
87                         print(three)
88                         three_all += 1
89
90 if op2 != 0:
91     four = op1 / op2
92
93 if three_all != 0:
94     ro = (similarity1 * 3) + (similarity2 * 1) + 1.0 * (three * 4) /
three_all + four
95     ro = ro / 9
96 else:
97     ro = (similarity1 * 3) + (similarity2 * 1) + four
98     ro = ro / 5
99
100 print(similarity1)
101 print(similarity2)
102 print(three)
103 print(three_all)
104 print(four)
105 print("两文件的相似度为 :")
106 print(ro)
107
108 return ro

```

Tips 计算相似度的主要思想是用较少的关键信息确定两者相似情况，主要依赖于 提取出的文件标题、文件领域分类信息进行定义，测试结果如下

3. 对应接口

主要包括计算指定文件指纹并保存、判断指定目录下文件与已保存文件相似度、直接判定两文件相似度

```

1 def get_file_finger(file_dir, out_dir):
2     """

```

```

3         计算文件指纹并保存至相应目录
4         :param file_dir: 输入文件路径, 可为目录或单个文件 str
5         :param out_dir: 输出路径, 为目录 str
6         :return: 无
7         """
8         manager = Manager()
9         ans = manager.list()
10        log_base_dir = r'D:\OCR\OCR_test\OCR_test\data' # 结果输出路径
11        table_dir = r'D:\OCR\OCR_test\OCR_test\data' # 表格抽取结果输出路径(若无
        需抽取表格, 则不用填)
12
13        embedding_path = r'D:\OCR\OCR_test\OCR_test\configuration\100000-small-
        modi.txt' # 词嵌入文件路径, 对应100000-small-modi.txt文件
14        keywords_path =
        r'D:\OCR\OCR_test\OCR_test\configuration\domain_keywords.txt' # 领域关键词文
        件路径, domain_keywords.txt
15        intersection_path =
        r'D:\OCR\OCR_test\OCR_test\configuration\domain_mapping_trade_table.txt' #
        领域间关键词交集文件路径, intersection.txt
16
17        table_extract = False # 是否抽取表格
18        print_info = True # 是否输出每个文件的结果信息
19        os.makedirs(log_base_dir, exist_ok=True)
20        os.makedirs(table_dir, exist_ok=True)
21
22        embeddings = get_embedding_table(embedding_path) # 读入词嵌入, 对应
        100000-small-modi.txt文件
23        domain_keywords = get_domain_keywords(keywords_path) # 读入领域关键词,
        domain_keywords.txt
24        intersection = get_intersection(intersection_path) # 读入交集关键词,
        intersection.txt
25
26        ocr = paddleocr.PaddleOCR(use_angle_cls=True, lang="ch") # 初始化ocr模
        型
27        table_engine = PPStructure(table=False, ocr=False, show_log=True) # 初
        始化版面识别模型
28        uie_dict = {} # 初始化UIE模型
29        for domain in schemas_dict.keys():
30            uie_dict[domain] = {}
31            for schema_type in schemas_dict[domain].keys():
32                uie_dict[domain][schema_type] =
        Taskflow("information_extraction", model='uie-base',
33        schema=schemas_dict[domain][schema_type])
34        if os.path.isdir(file_dir):
35            # 多文件信息抽取
36            main_for_multiprocess(file_dir, log_base_dir, embeddings,
        domain_keywords, intersection, ocr, table_engine,
37            uie_dict, table_extract, table_dir,
        print_info)
38        else:
39            # 单个文件信息抽取
40            main(file_dir, log_base_dir, embeddings, domain_keywords,
        intersection, ocr, table_engine, uie_dict,

```

```

41         table_extract=table_extract, table_dir=table_dir,
print_info=True)
42
43     ans[0], dict1 = encryption(0, 0)
44     ans[1], dict2 = encryption(0, 1)
45     sort(ans[0], ans[1], dict1, dict2)
46
47
48 def check_file_finger(file_dir, check_file_dir, out_dir):
49     """
50         检查一批文件是否为敏感文件
51     :param file_dir: 输入文件路径, 可为目录或单个文件 str
52     :param check_file_dir: 输入已保存指纹的路径 str
53     :param out_dir: 输出路径, 为目录 str
54     :return: 是否为敏感文件 bool
55     """
56     manager = Manager()
57     ans = manager.list()
58     log_base_dir = r'D:\OCR\OCR_test\OCR_test\data' # 结果输出路径
59     table_dir = r'D:\OCR\OCR_test\OCR_test\data' # 表格抽取结果输出路径 (若无
        需抽取表格, 则不用填)
60
61     embedding_path = r'D:\OCR\OCR_test\OCR_test\configuration\100000-small-
modi.txt' # 词嵌入文件路径, 对应100000-small-modi.txt文件
62     keywords_path =
r'D:\OCR\OCR_test\OCR_test\configuration\domain_keywords.txt' # 领域关键词文
        件路径, domain_keywords.txt
63     intersection_path =
r'D:\OCR\OCR_test\OCR_test\configuration\domain_mapping_trade_table.txt' #
        领域间关键词交集文件路径, intersection.txt
64
65     table_extract = False # 是否抽取表格
66     print_info = True # 是否输出每个文件的结果信息
67     os.makedirs(log_base_dir, exist_ok=True)
68     os.makedirs(table_dir, exist_ok=True)
69
70     embeddings = get_embedding_table(embedding_path) # 读入词嵌入, 对应
100000-small-modi.txt文件
71     domain_keywords = get_domain_keywords(keywords_path) # 读入领域关键词,
domain_keywords.txt
72     intersection = get_intersection(intersection_path) # 读入交集关键词,
intersection.txt
73
74     ocr = paddleocr.PaddleOCR(use_angle_cls=True, lang="ch") # 初始化ocr模
        型
75     table_engine = PPStructure(table=False, ocr=False, show_log=True) # 初
        始化版面识别模型
76     uie_dict = {} # 初始化UIE模型
77     for domain in schemas_dict.keys():
78         uie_dict[domain] = {}
79         for schema_type in schemas_dict[domain].keys():
80             uie_dict[domain][schema_type] =
Taskflow("information_extraction", model='uie-base',
81
schema=schemas_dict[domain][schema_type])

```

```

82     if os.path.isdir(file_dir):
83         # 多文件信息抽取
84         main_for_multiprocess(file_dir, log_base_dir, embeddings,
domain_keywords, intersection, ocr, table_engine,
85                               uie_dict, table_extract, table_dir,
print_info)
86     else:
87         # 单个文件信息抽取
88         main(file_dir, log_base_dir, embeddings, domain_keywords,
intersection, ocr, table_engine, uie_dict,
89             table_extract=table_extract, table_dir=table_dir,
print_info=True)
90     # 读取 + 遍历对比
91     ans[0], dict1 = encryption(0, 0)
92     ans[1], dict2 = encryption(0, 1)
93     sort(ans[0], ans[1], dict1, dict2)
94
95
96 def file_file_check(file_dir):
97     """
98     计算两个文件相似度
99     :param file_dir: 输入文件路径, 应当为两个文件所处的目录 str
100    :return: 相似度 float
101    """
102    manager = Manager()
103    ans = manager.list()
104
105    file_dir = r'D:\OCR\OCR_test\OCR_test\data' # 文件路径/文件夹路径
106    log_base_dir = r'D:\OCR\OCR_test\OCR_test\data' # 结果输出路径
107    table_dir = r'D:\OCR\OCR_test\OCR_test\data' # 表格抽取结果输出路径 (若无
需抽取表格, 则不用填)
108
109    embedding_path = r'D:\OCR\OCR_test\OCR_test\configuration\100000-small-
modi.txt' # 词嵌入文件路径, 对应100000-small-modi.txt文件
110    keywords_path =
r'D:\OCR\OCR_test\OCR_test\configuration\domain_keywords.txt' # 领域关键词文
件路径, domain_keywords.txt
111    intersection_path =
r'D:\OCR\OCR_test\OCR_test\configuration\domain_mapping_trade_table.txt' #
领域间关键词交集文件路径, intersection.txt
112
113    table_extract = False # 是否抽取表格
114    print_info = True # 是否输出每个文件的结果信息
115
116    # 初始化
117    ## 目录创建
118    os.makedirs(log_base_dir, exist_ok=True)
119    os.makedirs(table_dir, exist_ok=True)
120
121    ## 词表读入
122    embeddings = get_embedding_table(embedding_path) # 读入词嵌入, 对应
100000-small-modi.txt文件
123    domain_keywords = get_domain_keywords(keywords_path) # 读入领域关键词,
domain_keywords.txt

```

```
124     intersection = get_intersection(intersection_path) # 读入交集关键词，
intersection.txt
125
126     ## 模型加载
127     ocr = paddleocr.PaddleOCR(use_angle_cls=True, lang="ch") # 初始化ocr模
型
128     table_engine = PPStructure(table=False, ocr=False, show_log=True) # 初
始化版面识别模型
129     uie_dict = {} # 初始化UIE模型
130     for domain in schemas_dict.keys():
131         uie_dict[domain] = {}
132         for schema_type in schemas_dict[domain].keys():
133             uie_dict[domain][schema_type] =
Taskflow("information_extraction", model='uie-base',
134
schema=schemas_dict[domain][schema_type])
135
136     main_for_multiprocess(file_dir, log_base_dir, embeddings,
domain_keywords, intersection, ocr, table_engine,
137                             uie_dict, table_extract, table_dir, print_info)
138     ans[0], dict1 = encryption(0, 0)
139     ans[1], dict2 = encryption(0, 1)
140     sort(ans[0], ans[1], dict1, dict2)
```