文件指纹说明文档

1.文件指纹数据结构

```python
class FileHash:
    def __init__(self):
        # domain_pred_res: 领域预测结果，str
        # domain_pred_match_num: 领域预测结果的置信度，float
        # words: 文件名 / 标题关键词提取结果，set 其中不包括跨领域词
        # domain_matched: 各领域下匹配到的文本关键词，dict {领域: [文本关键词, 匹配的最佳领域词, 相似度]} 其中包括了非预测领域的结果
        # use_other 是否使用了other
        # file_key_word 存储schema和对应schema的结果 为string:list list中为提取出的信息
        self.domain_pred = ""
        self.domain_pred_match_num = 0.0
        self.words = set()
        self.domain_matched = {}
        self.use_other = False
        self.file_key_word = dict()

        self.hash1 = generate_ngram_lsh_fingerprint("hash1")
        self.hash2 = generate_ngram_lsh_fingerprint("hash2")
        self.hash3 = {}
        # hash3的具体结构为 {领域:[预测关键词, 已提取关键词]}
        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多进程变量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nparray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

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

```
    """
    # 创建MinHash对象
    minhash = MinHash(num_perm=num_perm)
    # 创建哈希函数（可以选择MD5、SHA-1等哈希算法）
    hash_function = hashlib.md5
    # 生成N-gram序列
    ngram_sequence = ngrams(text, n)
    # 添加N-gram序列到MinHash中
    for ngram in ngram_sequence:
        for item in ngram:
            minhash.update(hash_function(item.encode()).digest())
    # 返回指纹的哈希摘要
    fingerprint_digest = minhash.digest()
    return fingerprint_digest


def hash_fingerprint(fingerprint):
    """
    使用哈希函数对指纹进行进一步的处理。
    :param fingerprint: 要处理的指纹。
    :return: 处理后的指纹（哈希值）。
    """
    # 创建哈希函数（可以选择MD5、SHA-1、SHA-256等哈希算法）
    hash_function = hashlib.md5
    # 计算指纹的哈希值
    hashed_fingerprint = hash_function(fingerprint).hexdigest()
    return hashed_fingerprint


def hamming_distance(fingerprint1, fingerprint2):
    """
    计算两个指纹之间的汉明距离。
    :param fingerprint1: 第一个指纹。
    :param fingerprint2: 第二个指纹。
    :return: 汉明距离。
    """
    distance = sum(c1 != c2 for c1, c2 in zip(fingerprint1, fingerprint2))
    return distance


def calculate_similarity(fingerprint1, fingerprint2):
    """
    计算两个指纹之间的相似度。
    :param fingerprint1: 第一个指纹。
    :param fingerprint2: 第二个指纹。
    :return: 相似度。
    """
    hamming_dist = hamming_distance(fingerprint1, fingerprint2)
    similarity = 1 - (hamming_dist / len(fingerprint1))
    return similarity

def sort(file_hash1, file_hash2, dict1, dict2):
    """
    计算两个文件之间的总相似度。
    :param file_hash1: 存储了第一个文件相关hash信息的块 FileHash
```

```
    :param file_hash2: 存储了第二个文件相关hash信息的块 FileHash
    :param dict1: 存储了第一个文件相关hash3信息的字典 dict
    :param dict1: 存储了第二个文件相关hash3信息的字典 dict
    :return: 相似度。
    """
    similarity1 = calculate_similarity(file_hash1.hash1, file_hash2.hash1)
    similarity2 = calculate_similarity(file_hash1.hash2, file_hash2.hash2)
    three = 0
    three_all = 0
    four = 0
    op1 = 0
    op2 = 0
    for hash_item1 in dict1:
        for hash_item2 in dict2:
            if hash_item1 == hash_item2:
                a = len(dict1[hash_item1])
                b = len(dict2[hash_item2])
                op1 = op1 + (a <= b and a or b)
                op2 = op2 + (a >= b and a or b)
                for tmp1 in dict1[hash_item1]:
                    for tmp2 in dict2[hash_item2]:
                        if (tmp1[0] == tmp2[0]).all():
                            three += calculate_similarity(tmp1[1], tmp2[1])
                            print(three)
                            three_all += 1
    if op2 != 0:
        four = op1 / op2

    if three_all != 0:
        ro = (similarity1 * 3) + (similarity2 * 1) + 1.0 * (three * 4) /
three_all + four
        ro = ro / 9
    else:
        ro = (similarity1 * 3) + (similarity2 * 1) + four
        ro = ro / 5
    print(similarity1)
    print(similarity2)
    print(three)
    print(three_all)
    print(four)
    print("两文件的相似度为 :")
    print(ro)

    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

```
def get_file_finger(file_dir, out_dir):
    """

```

```
        计算文件指纹并保存至相应目录
    :param file_dir: 输入文件路径，可为目录或单个文件 str
    :param out_dir: 输出路径，为目录 str
    :return: 无
    """
    manager = Manager()
    ans = manager.list()
    log_base_dir = r'D:\OCR\OCR_test\OCR_test\data'  # 结果输出路径
    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）

    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
    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
    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

    table_extract = False  # 是否抽取表格
    print_info = True  # 是否输出每个文件的结果信息
    os.makedirs(log_base_dir, exist_ok=True)
    os.makedirs(table_dir, exist_ok=True)

    embeddings = get_embedding_table(embedding_path)  # 读入词嵌入，对应100000-small-modi.txt文件
    domain_keywords = get_domain_keywords(keywords_path)  # 读入领域关键词，domain_keywords.txt
    intersection = get_intersection(intersection_path)  # 读入交集关键词，intersection.txt

    ocr = paddleocr.PaddleOCR(use_angle_cls=True, lang="ch")  # 初始化ocr模型
    table_engine = PPStructure(table=False, ocr=False, show_log=True)  # 初始化版面识别模型
    uie_dict = {}  # 初始化UIE模型
    for domain in schemas_dict.keys():
        uie_dict[domain] = {}
        for schema_type in schemas_dict[domain].keys():
            uie_dict[domain][schema_type] = Taskflow("information_extraction", model='uie-base',

schema=schemas_dict[domain][schema_type])
    if os.path.isdir(file_dir):
        # 多文件信息抽取
        main_for_multiprocess(file_dir, log_base_dir, embeddings, domain_keywords, intersection, ocr, table_engine,
                              uie_dict, table_extract, table_dir, print_info)
    else:
        # 单个文件信息抽取
        main(file_dir, log_base_dir, embeddings, domain_keywords, intersection, ocr, table_engine, uie_dict,
```

```
            table_extract=table_extract, table_dir=table_dir,
print_info=True)

    ans[0], dict1 = encryption(0, 0)
    ans[1], dict2 = encryption(0, 1)
    sort(ans[0], ans[1], dict1, dict2)


def check_file_finger(file_dir, check_file_dir, out_dir):
    """
        检查一批文件是否为敏感文件
    :param file_dir: 输入文件路径，可为目录或单个文件 str
    :param check_file_dir: 输入已保存指纹的路径 str
    :param out_dir: 输出路径，为目录 str
    :return: 是否为敏感文件 bool
    """
    manager = Manager()
    ans = manager.list()
    log_base_dir = r'D:\OCR\OCR_test\OCR_test\data'  # 结果输出路径
    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）

    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
    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
    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

    table_extract = False  # 是否抽取表格
    print_info = True  # 是否输出每个文件的结果信息
    os.makedirs(log_base_dir, exist_ok=True)
    os.makedirs(table_dir, exist_ok=True)

    embeddings = get_embedding_table(embedding_path)  # 读入词嵌入，对应
100000-small-modi.txt文件
    domain_keywords = get_domain_keywords(keywords_path)  # 读入领域关键词，
domain_keywords.txt
    intersection = get_intersection(intersection_path)  # 读入交集关键词，
intersection.txt

    ocr = paddleocr.PaddleOCR(use_angle_cls=True, lang="ch")  # 初始化ocr模
型
    table_engine = PPStructure(table=False, ocr=False, show_log=True)  # 初
始化版面识别模型
    uie_dict = {}  # 初始化UIE模型
    for domain in schemas_dict.keys():
        uie_dict[domain] = {}
        for schema_type in schemas_dict[domain].keys():
            uie_dict[domain][schema_type] =
Taskflow("information_extraction", model='uie-base',

schema=schemas_dict[domain][schema_type])
```

```
    if os.path.isdir(file_dir):
        # 多文件信息抽取
        main_for_multiprocess(file_dir, log_base_dir, embeddings, domain_keywords, intersection, ocr, table_engine,
                              uie_dict, table_extract, table_dir, print_info)
    else:
        # 单个文件信息抽取
        main(file_dir, log_base_dir, embeddings, domain_keywords, intersection, ocr, table_engine, uie_dict,
             table_extract=table_extract, table_dir=table_dir, print_info=True)
    # 读取 + 遍历对比
    ans[0], dict1 = encryption(0, 0)
    ans[1], dict2 = encryption(0, 1)
    sort(ans[0], ans[1], dict1, dict2)


def file_file_check(file_dir):
    """
        计算两个文件相似度
    :param file_dir: 输入文件路径，应当为两个文件所处的目录 str
    :return: 相似度 float
    """
    manager = Manager()
    ans = manager.list()

    file_dir = r'D:\OCR\OCR_test\OCR_test\data'  # 文件路径/文件夹路径
    log_base_dir = r'D:\OCR\OCR_test\OCR_test\data'  # 结果输出路径
    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）

    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
    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
    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

    table_extract = False  # 是否抽取表格
    print_info = True  # 是否输出每个文件的结果信息

    # 初始化
    ## 目录创建
    os.makedirs(log_base_dir, exist_ok=True)
    os.makedirs(table_dir, exist_ok=True)

    ## 词表读入
    embeddings = get_embedding_table(embedding_path)  # 读入词嵌入，对应100000-small-modi.txt文件
    domain_keywords = get_domain_keywords(keywords_path)  # 读入领域关键词，domain_keywords.txt
```

```
    intersection = get_intersection(intersection_path)  # 读入交集关键词，intersection.txt

    ## 模型加载
    ocr = paddleocr.PaddleOCR(use_angle_cls=True, lang="ch")  # 初始化ocr模型
    table_engine = PPStructure(table=False, ocr=False, show_log=True)  # 初始化版面识别模型
    uie_dict = {}  # 初始化UIE模型
    for domain in schemas_dict.keys():
        uie_dict[domain] = {}
        for schema_type in schemas_dict[domain].keys():
            uie_dict[domain][schema_type] = Taskflow("information_extraction", model='uie-base', schema=schemas_dict[domain][schema_type])

    main_for_multiprocess(file_dir, log_base_dir, embeddings, domain_keywords, intersection, ocr, table_engine,
                          uie_dict, table_extract, table_dir, print_info)
    ans[0], dict1 = encryption(0, 0)
    ans[1], dict2 = encryption(0, 1)
    sort(ans[0], ans[1], dict1, dict2)
```