# Lab1

成员：

- 周恩帅 PB17111561
- 张文岚 PB17111587

## 算法概述

首先我们需要对每个文档进行编号，生成一个文档id和路径的映射表`id_path_map.csv`

```
def gen_id_path_map(dataset_path, id_path_map):
    id_path_dict = {}
    num_files = 0
    for root, dirs, files in os.walk(dataset_path, topdown=False):
        for f in files:
            id_path_dict[num_files] = os.path.join(root.replace(dataset_path,
""), f)
            num_files += 1
    with open(id_path_map, "w+", newline="") as fp:
        w = csv.writer(fp)
        w.writerows(id_path_dict.items())
    return id_path_dict
```

之后每篇文档都以其id来指代。一共是517401篇文档。

### 预处理

数据集是十分庞大的，分词、词干提取等整个跑一遍得花50多分钟，这是难以接受的。所以我们采取分阶段的方式，每执行一步，都将结果保存到文件中。

经过分析，时间瓶颈在分词和词干提取上，因此我们先对文档做了预处理，将每篇文档分词、词干提取、删除停用词的结果保存在`output/doc_tokens`目录下，

这个预处理的过程是漫长的，因此我们考虑使用并行的方式。事实上，这个任务有很好的并行性，将所有文档分成8组，然后拆分到8个核上运行，性能最终有3倍左右的提升：

```
# 每个核上运行相同的代码
def preprocess(core_id, low_id, high_id, id_path_dict):
    print("Task%d starts..." % core_id)
    for iter in range(low_id, high_id):
        if iter >= util.docs_num:
            break
        if iter % 1000 == 0:
            print(iter)
        doc_id = iter
        doc_path = id_path_dict[doc_id]
        with open(os.path.join("..", "dataset", doc_path)) as doc_fp:
            # 对文档进行分词、词干提取等处理，后面会说到
            doc_str = util.doc2str(doc_fp)
            tokens = util.tokenize(doc_str)
            tokens = map(util.stem, tokens)
            tokens = filter(util.del_stop, tokens)
```

```
            path = os.path.join("..", "output", "doc_tokens", "%d.csv" % doc_id)
            write_tokens(path, tokens)
    print("Task%d ends..." % core_id)
```

下面是利用python多进程编写的并行程序：

```
cores = 8
core_payload = int(util.docs_num / cores) + 1
print("Start preprocessing")
start_time = time.time()
p = Pool(cores)
for i in range(cores):
    p.apply_async(
        preprocess, args=(i, i * core_payload, (i + 1) * core_payload,
id_path_dict)
    )
p.close()
p.join()
```

由于我的电脑只有8个核，因此性能还有很大提升空间

## 词条处理

本实验中，我们使用python的nlp工具包对邮件进行了分词、词干提取、删除停用词等处理，其中词干提取算法是Snowball算法，比较快速。

这里的停用词包括数字和特殊符号。

```
def doc2str(doc_fp):
    try:
        mail = email.parser.Parser().parse(doc_fp)
        mail_subject = mail.get("Subject")
        mail_body = mail.get_payload()
        return mail_subject + " " + mail_body
    except Exception as e:
        return "the"


def tokenize(doc_str):
    doc_str = doc_str.translate(del_tran_table)
    tokens = nltk.tokenize.word_tokenize(doc_str)
    return tokens


def stem(token):
    token = token.lower()
    return stemmer.stem(token)


def del_stop(token):
    return token not in stopwords
```

## 生成Top1000词条

为了减轻内存的压力，本实验只要求对词频前1000的词条进行检索，因此我们先把这前1000的词条找出来，顺便统计这些词条的文档频率：

```
 # 局部代码
    tokens_ttf = {}
 tokens_df = {}

 for iter in range(util.low_id, util.high_id):
     doc_id = iter
     doc_tokens_path = os.path.join("..", "output", "doc_tokens", "%d.csv" %
doc_id)
     with open(doc_tokens_path) as doc_fp:
         r = csv.reader(doc_fp)
         tokens_tf = dict((rows[0], int(rows[1])) for rows in r)
         append_ttf_df(tokens_tf, tokens_ttf, tokens_df)
 ttf_1000 = sorted(tokens_ttf.items(), key=lambda kv: (kv[1], kv[0]),
reverse=True)[
     :1000
 ]
 with open(os.path.join("..", "output", "ttf_1000.csv"), "w+", newline="") as
fp:
     w = csv.writer(fp)
     w.writerows(ttf_1000)

 tokens_1000 = list(map(lambda x: x[0], ttf_1000))
 df_1000 = list(map(lambda x: (x, tokens_df[x]), tokens_1000))
 with open(os.path.join("..", "output", "df_1000.csv"), "w+", newline="") as fp:
     w = csv.writer(fp)
     w.writerows(df_1000)
```

结果保存在`output/df_1000.csv`和`output/ttf_1000.csv`中，这里ttf指total term frequency，即总词项频率。

## 倒排表

从之前预处理步骤读取每个文档的词条，然后只生成top1000的词条的倒排索引：

```
with open(os.path.join("..", "output", "ttf_1000.csv")) as fp:
    r = csv.reader(fp)
    total_tf_1000 = list(r)
target_tokens = set(map(lambda x: x[0], total_tf_1000))
inverted_indices = {}
cost_time = [0, 0, 0, 0]
temp_time = [0, 0, 0, 0]
for iter in range(util.low_id, util.high_id):
    doc_id = iter
    doc_tokens_path = os.path.join("..", "output", "doc_tokens", "%d.csv" %
doc_id)
    with open(doc_tokens_path) as doc_fp:
        r = csv.reader(doc_fp)
        tokens = set(map(lambda x: x[0], list(r)))
        tokens = filter(token_filter, tokens)
        # add tokens of a certain doc into inverted index table
        append_tokens(tokens, doc_id, inverted_indices)
```

```
for key, value in inverted_indices.items():
    path = os.path.join("..", "output", "inverted_index_table", key + ".csv")
    with open(path, "w+", newline="") as fp:
        w = csv.writer(fp)
        for j in value:
            w.writerow([j])
```

结果保存到`output/inverted_index_table/{token}.csv`中，每个词条生成一个倒排索引文件，这样在查找时，就不用把所有的索引都读取进内存，而只需要读取要查询的词条的索引即可，这样速度快且节省内存。

## 布尔检索

为了识别布尔语义，我们需要写一个布尔语句解析器，即使用一种经典的的栈方法来计算bool表达式，参与bool运算的操作数是每个词条对应的索引集合，利用python的set数据结构，可以方便地对结果进行交并补等操作。

```
def bool_query(origin_query_str):
    query_str = (
        origin_query_str.lower()
        .replace("(", " ( ")
        .replace(")", " ) ")
        .replace("and", "&")
        .replace("or", "|")
        .replace("not", "!")
    )
    query_exp = query_str.split()
    query_exp.append("$")

    operand_stack = list()
    operator_stack = list("$")
    i = 0
    while True:
        element = query_exp[i]
        if element in util.operators_level:
            operator = operator_stack.pop()
            if util.operators_level[operator] < util.operators_level[element]:
                operator_stack.append(operator)
                operator_stack.append(element)
                i += 1
            else:
                if operator == "$":
                    break
                elif operator == "(":
                    if element != ")":
                        operator_stack.append(operator)
                        operator_stack.append(element)
                    i += 1
                elif operator == "!":
                    operand = operand_stack.pop()
                    result = util.operator_func[operator](operand)
                    operand_stack.append(result)
                else:
                    operand1 = operand_stack.pop()
                    operand2 = operand_stack.pop()
                    result = util.operator_func[operator](operand1, operand2)
```

```
                    operand_stack.append(result)
        else:
            operand = util.get_indices(element)  # set
            operand_stack.append(operand)
            i += 1
    return operand_stack.pop()
```

## 向量化表示

向量化表示和生成倒排索引类似，对每篇文档中的词项频率进行统计，进而算出每个词项的tf-idf值，从而算出该文档的向量。

每个文档的向量写入`output/doc_wordvec/{doc_id}.csv`中，该任务也可以并行。

```
def doc_vec(core_id, low_id, high_id):
    print("Task%d starts..." % core_id)
    with open(os.path.join("..", "output", "df_1000.csv")) as fp:
        r = csv.reader(fp)
        target_tokens_df = dict((rows[0], int(rows[1])) for rows in r)
        target_tokens_list = sorted(list(target_tokens_df.keys()))
    for iter in range(low_id, high_id):
        if iter >= util.docs_num:
            break
        if iter % 1000 == 0:
            print(iter)
        doc_id = iter
        doc_tokens_path = os.path.join("..", "output", "doc_tokens", "%d.csv" %
doc_id)
        with open(doc_tokens_path) as doc_fp:
            r = csv.reader(doc_fp)
            doc_tokens_tf = dict((rows[0], int(rows[1])) for rows in r)

            w_td = [0] * 1000
            j = 0
            for token in target_tokens_list:
                if token in doc_tokens_tf:
                    w_td[j] = (1 + math.log(doc_tokens_tf[token], 10)) *
math.log(
                        util.docs_num / target_tokens_df[token], 10
                    )
                j += 1

            path = os.path.join("..", "output", "doc_wordvec", "%d.csv" %
doc_id)
            with open(path, "w+", newline="") as fp:
                w = csv.writer(fp)
                w.writerow(w_td)
    print("Task%d ends..." % core_id)
```

## 语义检索

同样的，对于一个查询语句，我们也对其进行向量化，然后将该向量与每个文档向量求余弦相似度，最终返回前十个文档。

语义检索也可以很好地并行化。将50w+文档拆分到8核上，每个核计算属于它那一组文档和查询的余弦相似度，然后返回十个最相似的文档。

最终，我们从8个核上获得前80个相似文档，然后我们再在这80个文档里挑出前10相似的文档：

```
query_str = input("Query sentence: ")
query_vec = gen_query_vec(query_str)
print("Start searching")
start_time = time.time()
results = []
p = Pool(cores)
for i in range(cores):
    results.append(
        p.apply_async(
            query, args=(i, query_vec, i * core_payload, (i + 1) * core_payload)
        )
    )
p.close()
p.join()
docs_cos_top_80 = []
for res in results:
    docs_cos_top_80.extend(res.get())
docs_cos = get_top(dict(docs_cos_top_80), 10)
query_results = list(map(lambda x: (x[0], x[1], id_path_dict[x[0]]), docs_cos))
```

# 运行结果

5个查询词：market，energy，business，office，plans

由于结果比较多，截图只保留部分，但是速度还是非常快的，只是前期建立索引比较耗时。

## 布尔检索

输出格式为(文档id，文档路径)

- `market and energy and business and office and plans`

```
d_items\\335_'), (508281, 'williams-j\\deleted_items\\382_'), (508490, 'williams-j\\discussion_threads\\61_'), (508654, 'williams
-j\\inbox\\211_'), (508812, 'williams-j\\private_folders\\jays_stuff\\16_'), (509238, 'williams-w3\\bill_williams_iii\\307_'), (5
09425, 'williams-w3\\bill_williams_iii\\476_'), (509476, 'williams-w3\\bill_williams_iii\\521_'), (510297, 'williams-w3\\inbox\\8
6_'), (510304, 'williams-w3\\inbox\\92_'), (512007, 'williams-w3\\sent_items\\307_'), (512664, 'wolfe-j\\07_saved\\87_'), (513357
, 'wolfe-j\\inbox\\107_'), (513535, 'wolfe-j\\inbox\\269_'), (514151, 'ybarbo-p\\deleted_items\\114_'), (514170, 'ybarbo-p\\delet
ed_items\\136_'), (514171, 'ybarbo-p\\deleted_items\\137_'), (514172, 'ybarbo-p\\deleted_items\\138_'), (514201, 'ybarbo-p\\delet
ed_items\\40_'), (514546, 'ybarbo-p\\inbox\\181_'), (514597, 'ybarbo-p\\inbox\\227_'), (514601, 'ybarbo-p\\inbox\\230_'), (514626
, 'ybarbo-p\\inbox\\253_'), (514635, 'ybarbo-p\\inbox\\261_'), (514654, 'ybarbo-p\\inbox\\279_'), (514665, 'ybarbo-p\\inbox\\289_
'), (514673, 'ybarbo-p\\inbox\\296_'), (514842, 'ybarbo-p\\inbox\\450_'), (514965, 'ybarbo-p\\inbox\\587_'), (515541, 'zipper-a\\
deleted_items\\186_'), (516260, 'zipper-a\\inbox\\3_'), (517045, 'zufferli-j\\inbox\\38_'), (517182, 'zufferli-j\\sent_items\\211
_'), (517353, 'zufferli-j\\sent_items\\57_')]
```

- `market and not energy and business and office and plans`

```
), (500916, 'whalley-l\\all_documents\\431_'), (500934, 'whalley-l\\all_documents\\468_'), (501089, 'whalley-l\\all_documents\\64
1_'), (501251, 'whalley-l\\all_documents\\815_'), (501294, 'whalley-l\\all_documents\\858_'), (501402, 'whalley-l\\all_documents\
\986_'), (501773, 'whalley-l\\discussion_threads\\1215_'), (501898, 'whalley-l\\discussion_threads\\1331_'), (502103, 'whalley-l\
\discussion_threads\\280_'), (502192, 'whalley-l\\discussion_threads\\369_'), (502341, 'whalley-l\\discussion_threads\\535_'), (5
02492, 'whalley-l\\discussion_threads\\701_'), (502509, 'whalley-l\\discussion_threads\\736_'), (502518, 'whalley-l\\discussion_t
hreads\\753_'), (502560, 'whalley-l\\ecommerce\\2_'), (502606, 'whalley-l\\notes_inbox\\139_'), (502612, 'whalley-l\\notes_inbox\
\144_'), (502627, 'whalley-l\\notes_inbox\\158_'), (502771, 'whalley-l\\notes_inbox\\298_'), (502921, 'whalley-l\\notes_inbox\\44
9_'), (503042, 'whalley-l\\notes_inbox\\560_'), (503107, 'whalley-l\\notes_inbox\\61_'), (503228, 'whalley-l\\sent\\111_'), (5036
62, 'whalley-l\\_sent_mail\\92_'), (506421, 'white-s\\personal\\10_'), (507108, 'whitt-m\\inbox\\266_'), (507293, 'whitt-m\\inbox
\\437_'), (508120, 'williams-j\\deleted_items\\223_'), (508271, 'williams-j\\deleted_items\\372_'), (508655, 'williams-j\\inbox\\
212_'), (508860, 'williams-j\\sent_items\\124_'), (509829, 'williams-w3\\bill_williams_iii\\844_'), (512485, 'wolfe-j\\06_saved\\
13_'), (512676, 'wolfe-j\\07_saved\\98_'), (512712, 'wolfe-j\\09_saved\\10_'), (512744, 'wolfe-j\\09_saved\\40_'), (515299, 'zipp
er-a\\all_documents\\27_'), (516007, 'zipper-a\\discussion_threads\\22_'), (516155, 'zipper-a\\inbox\\149_'), (516574, 'zipper-a\
\sent_items\\275_'), (516752, 'zipper-a\\tq\\62_')]
```

- `market and not (energy or business) and office and plans`

```
126, 'taylor-m\\notes_inbox\\2549_'), (482724, 'taylor-m\\notes_inbox\\599_'), (482727, 'taylor-m\\notes_inbox\\600_'), (482728,
'taylor-m\\notes_inbox\\601_'), (482730, 'taylor-m\\notes_inbox\\603_'), (482948, 'taylor-m\\notes_inbox\\7_'), (482999, 'taylor-
m\\notes_inbox\\846_'), (483618, 'taylor-m\\sent\\1054_'), (483621, 'taylor-m\\sent\\1057_'), (485712, 'taylor-m\\sent_items\\366
_'), (485817, 'taylor-m\\sent_items\\47_'), (488393, 'thomas-p\\deleted_items\\277_'), (490229, 'tycholiz-b\\deleted_items\\36_')
, (490230, 'tycholiz-b\\deleted_items\\37_'), (490972, 'tycholiz-b\\sent_items\\191_'), (491295, 'tycholiz-b\\_sent_mail\\68_'),
(494911, 'watson-k\\e_mail_bin\\459_'), (495112, 'watson-k\\e_mail_bin\\661_'), (495248, 'watson-k\\inbox\\163_'), (495362, 'wats
on-k\\inbox\\266_'), (495379, 'watson-k\\inbox\\281_'), (496055, 'watson-k\\sent_items\\255_'), (496332, 'watson-k\\sent_items\\5
06_'), (496542, 'watson-k\\sent_items\\697_'), (496544, 'watson-k\\sent_items\\699_'), (496557, 'watson-k\\sent_items\\70_'), (49
6737, 'watson-k\\sent_items\\873_'), (497352, 'weldon-c\\deleted_items\\220_'), (498673, 'whalley-g\\all_documents\\319_'), (4991
45, 'whalley-g\\deleted_items\\41_'), (499374, 'whalley-g\\discussion_threads\\297_'), (500030, 'whalley-g\\notes_inbox\\45_'), (
500462, 'whalley-l\\all_documents\\1119_'), (500845, 'whalley-l\\all_documents\\319_'), (501758, 'whalley-l\\discussion_threads\\
1200_'), (502119, 'whalley-l\\discussion_threads\\297_'), (502933, 'whalley-l\\notes_inbox\\45_'), (503057, 'whalley-l\\notes_inb
ox\\575_'), (504732, 'white-s\\deleted_items\\723_'), (505264, 'white-s\\inbox\\111_'), (507469, 'whitt-m\\sent_items\\122_'), (5
15484, 'zipper-a\\deleted_items\\130_'), (515486, 'zipper-a\\deleted_items\\132_'), (515487, 'zipper-a\\deleted_items\\133_'), (5
16300, 'zipper-a\\inbox\\76_')]
```

- `market and (energy or business) and (office or plans)`

```
rbo-p\\inbox\\526_'), (514912, 'ybarbo-p\\inbox\\539_'), (514919, 'ybarbo-p\\inbox\\545_'), (514945, 'ybarbo-p\\inbox\\569_'), (5
14965, 'ybarbo-p\\inbox\\587_'), (514973, 'ybarbo-p\\inbox\\594_'), (514982, 'ybarbo-p\\inbox\\601_'), (515299, 'zipper-a\\all_do
cuments\\27_'), (515485, 'zipper-a\\deleted_items\\131_'), (515516, 'zipper-a\\deleted_items\\15_'), (515541, 'zipper-a\\deleted_
items\\186_'), (515613, 'zipper-a\\deleted_items\\251_'), (515625, 'zipper-a\\deleted_items\\262_'), (515638, 'zipper-a\\deleted_
items\\274_'), (515704, 'zipper-a\\deleted_items\\33_'), (515816, 'zipper-a\\deleted_items\\445_'), (515819, 'zipper-a\\deleted_i
tems\\448_'), (515830, 'zipper-a\\deleted_items\\45_'), (515856, 'zipper-a\\deleted_items\\483_'), (515861, 'zipper-a\\deleted_it
ems\\488_'), (515874, 'zipper-a\\deleted_items\\4_'), (515882, 'zipper-a\\deleted_items\\507_'), (515891, 'zipper-a\\deleted_item
s\\515_'), (515903, 'zipper-a\\deleted_items\\526_'), (515913, 'zipper-a\\deleted_items\\536_'), (515922, 'zipper-a\\deleted_item
s\\545_'), (515963, 'zipper-a\\deleted_items\\6_'), (516007, 'zipper-a\\discussion_threads\\22_'), (516105, 'zipper-a\\inbox\\101
_'), (516139, 'zipper-a\\inbox\\134_'), (516143, 'zipper-a\\inbox\\138_'), (516145, 'zipper-a\\inbox\\13_'), (516148, 'zipper-a\\
inbox\\142_'), (516155, 'zipper-a\\inbox\\149_'), (516164, 'zipper-a\\inbox\\157_'), (516172, 'zipper-a\\inbox\\165_'), (516260,
'zipper-a\\inbox\\3_'), (516377, 'zipper-a\\origination\\7_'), (516574, 'zipper-a\\sent_items\\275_'), (516619, 'zipper-a\\sent_i
tems\\343_'), (516643, 'zipper-a\\sent_items\\365_'), (516752, 'zipper-a\\tq\\62_'), (516833, 'zipper-a\\tss\\21_'), (516956, 'zu
fferli-j\\deleted_items\\87_'), (516963, 'zufferli-j\\deleted_items\\93_'), (517018, 'zufferli-j\\inbox\\13_'), (517045, 'zufferl
i-j\\inbox\\38_'), (517104, 'zufferli-j\\sent_items\\139_'), (517182, 'zufferli-j\\sent_items\\211_'), (517199, 'zufferli-j\\sent
_items\\227_'), (517229, 'zufferli-j\\sent_items\\254_'), (517353, 'zufferli-j\\sent_items\\57_')]
```

这个的结果非常之多

- `not (market or energy or business or office or plans)`

```
sovich-j\\all_documents\\29351_'), (62513, 'dasovich-j\\all_documents\\29352_'), (62515, 'dasovich-j\\all_documents\\29354_'), (6
2516, 'dasovich-j\\all_documents\\29355_'), (62518, 'dasovich-j\\all_documents\\29357_'), (62519, 'dasovich-j\\all_documents\\293
58_'), (62521, 'dasovich-j\\all_documents\\2935_'), (62522, 'dasovich-j\\all_documents\\29360_'), (62523, 'dasovich-j\\all_docume
nts\\29361_'), (62524, 'dasovich-j\\all_documents\\29362_'), (62526, 'dasovich-j\\all_documents\\29364_'), (62527, 'dasovich-j\\a
ll_documents\\29365_'), (62529, 'dasovich-j\\all_documents\\29368_'), (62531, 'dasovich-j\\all_documents\\29370_'), (62534, 'daso
vich-j\\all_documents\\29373_'), (62536, 'dasovich-j\\all_documents\\29375_'), (62537, 'dasovich-j\\all_documents\\29376_'), (625
38, 'dasovich-j\\all_documents\\29377_'), (62539, 'dasovich-j\\all_documents\\29378_'), (62540, 'dasovich-j\\all_documents\\29379
_'), (62542, 'dasovich-j\\all_documents\\29380_'), (62546, 'dasovich-j\\all_documents\\2938_'), (62547, 'dasovich-j\\all_document
s\\2939_'), (62549, 'dasovich-j\\all_documents\\2940_'), (62550, 'dasovich-j\\all_documents\\2941_'), (62551, 'dasovich-j\\all_do
cuments\\2942_'), (62552, 'dasovich-j\\all_documents\\2943_'), (62554, 'dasovich-j\\all_documents\\2945_'), (62555, 'dasovich-j\\
all_documents\\2946_'), (62556, 'dasovich-j\\all_documents\\2947_'), (62557, 'dasovich-j\\all_documents\\2948_'), (62558, 'dasovi
ch-j\\all_documents\\2949_'), (62560, 'dasovich-j\\all_documents\\2950_'), (62561, 'dasovich-j\\all_documents\\2951_'), (62562, '
dasovich-j\\all_documents\\2952_'), (62563, 'dasovich-j\\all_documents\\2953_'), (62564, 'dasovich-j\\all_documents\\2954_'), (62
565, 'dasovich-j\\all_documents\\2955_'), (62568, 'dasovich-j\\all_documents\\2958_'), (62569, 'dasovich-j\\Traceback (most recen
```

结果太多了，以至于短时间没跑完，被我kill掉了

# 语义检索

语义检索跑起来还是十分耗时的，因为它需要和50w+的文档求余弦相似度，计算和IO都是比较耗时的，因此这里只放两个运行结果示例。

输出格式（文档id，余弦相似度，文档路径）

- Query sentence: `enron wants to build an office`

```
Total time: 188.187573s
[(4636, 0.7927971736315429, 'arnold-j\\deleted_items\\55_'), (516465, 0.7306377147293412, 'zipper-a\\sent_items\\175_'), (277920,
 0.6914824019325179, 'lenhart-m\\sent\\1922_'), (274760, 0.6914824019325179, 'lenhart-m\\all_documents\\1923_'), (2571, 0.5758894
072560072, 'allen-p\\_sent_mail\\221_'), (1598, 0.5758894072560072, 'allen-p\\sent\\174_'), (131, 0.5758894072560072, 'allen-p\\a
ll_documents\\220_'), (335734, 0.5612631330325146, 'mclaughlin-e\\_sent_mail\\96_'), (335163, 0.5612631330325146, 'mclaughlin-e\\
sent\\195_'), (332939, 0.5612631330325146, 'mclaughlin-e\\all_documents\\745_')]
```

- Query sentence: `the president signed projects`

```
Total time: 196.705137s
[(278599, 0.5543012421366269, 'lenhart-m\\sent\\720_'), (275502, 0.5543012421366269, 'lenhart-m\\all_documents\\720_'), (46458, 0
.5543012421366269, 'campbell-l\\_sent_mail\\79_'), (45993, 0.5543012421366269, 'campbell-l\\sent\\301_'), (41435, 0.5543012421366
269, 'campbell-l\\all_documents\\755_'), (3010, 0.5543012421366269, 'allen-p\\_sent_mail\\79_'), (1769, 0.5543012421366269, 'alle
n-p\\sent\\32_'), (591, 0.5543012421366269, 'allen-p\\all_documents\\74_'), (498809, 0.5505624038365666, 'whalley-g\\calendar\\11
_'), (410426, 0.5505624038365666, 'scott-s\\_sent_mail\\347_')]
```