605.744: Information Retrieval
Programming Assignment #1: Corpus Statistics

Sabbir Ahmed

September 6, 2022

1 Introduction

This paper describes a collection of scripts written to compute the term-frequency, document-frequency, and other statistics from a pre-generated corpus.

2 Technical Background

All of the source code is in Python 3.10. The program is split into several modules and follows an object oriented structure. The following is the directory structure of the source code:

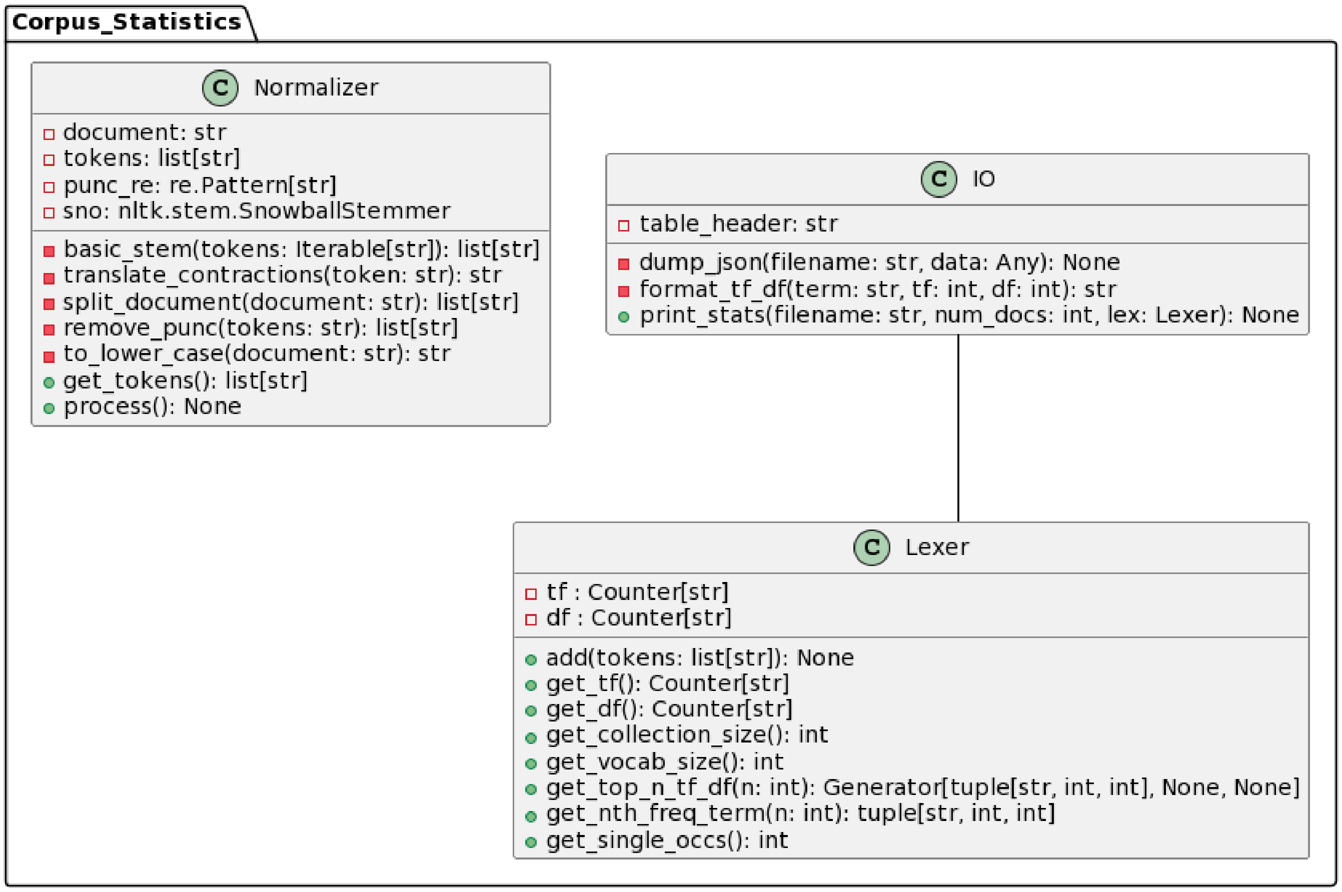
Figure 1: Directory Hierarchy of Assignment 1

```
.
├── data
│   ├── headlines_df.json
│   ├── headlines_stats.txt
│   ├── headlines_tf.json
│   ├── yelp_df.json
│   ├── yelp_stats.txt
│   └── yelp_tf.json
├── run.py
├── src
│   ├── __init__.py
│   ├── io.py
│   ├── lexer.py
│   └── normalize.py
```

The source code for all of the files are attached in Appendix A.

The total number of non-empty lines of code for the program comes to 200, with an average execution time of 40 seconds to process both of the sample files.

Figure 2: UML of Corpus Statistics



2.1 Driver

The driver script for the program is `run.py`. The file is responsible for opening up the input files, reading in all of the documents contained in those files, processing them into the lexicon, and generating and saving statistics on them. Since the sample files are relatively small, the entire content may be loaded into the memory of a modern machine. However, to account for possible larger files, the lines containing the documents are saved in memory, normalized, and added to the lexicon before moving to the next document.

This method of reading the files imposes a limitation, where every document is assumed to be on the 2nd line out of 4 (line_num % 4 == 1). The algorithm will need to be modified if the files in the future have different formats.

2.2 `normalize.Normalizer`

The `Normalizer` class normalizes the documents. The class is instantiated once and it provides a method to set the document for normalization. The normalization pipeline includes the following processes, in order:

1. translating the entire document to lower case. The document is unconditionally transformed into lower case, which can lead to confusion with words that are considered proper nouns.

2. splitting the document on whitespace into tokens.
3. expanding word contractions into their components. The contractions are pre-generated into a Python dict in `normalize.CONTRACTIONS` for constant-time look-ups.
4. splitting the tokens on all punctuations except "'". The "'" is preserved for the stemmer to process later. Not splitting on this character also avoids including incorrect tokens into the lexicon. For example, "food's" will get tokenized into ["food", "s"].
5. rejoining and splitting the tokens to remove empty tokens.
6. stemming tokens with nltk.stem.SnowballStemmer.stem(word).

The tokens are available as a list of string to be added into the lexicon before being overwritten by the next document.

2.2.1 External Libraries

The external library `nltk` was used for its stemming capabilities. The stemmers `PorterStemmer` and `SnowballStemmer` were tested over a sample of the documents, and the latter algorithm appeared to yield subjectively better-looking results.

2.3 `lexer.Lexer`

The `Lexer` class utilizes 2 `collections.Counter` objects to store the term-frequency and document-frequency of the lexicon. The bag-type container is a part of the standard library and match in performance compared to default `dicts`, but with additional methods such as `Counter.total()`, `Counter.most_common(n)`, etc. These methods are helpful in generating statistics required in the assignment.

3 Statistics and Observations

Since stopwords were not removed, they were prominent in the 100 most frequent terms in both of the files. "the" and "to" were present in the top 5 most frequent terms in both of the sets of documents. In the Yelp corpus, some of the most commonly occurring words were related to food and restaurant services. This theme is expected due to the source of the documents containing reviews of mostly food service establishments and its quality. The Headlines corpus did not appear to circulate a theme, although some of the frequent words appeared to be numerical or date values. This can be expected due to the source of the documents mostly discussing events with timelines. The numerical values can indicate money, percentage, weather, length of service, etc.

Both of the corpora had an almost identical percentage of singly-occurring terms (48.8%). These are terms that appeared in only one document.

The outputs are attached in Appendix B.

A Source Code

Code Listing 1: ./src/io.py

```
import json
from typing import Any
```

```python
from .lexer import Lexer


class IO:

    __table_header: str = (
        f"{'Word':<12} | {'TF':<6} | {'DF':<6}\n----------------------------"
    )

    @staticmethod
    def __dump_json(filename: str, data: Any) -> None:

        with open(f"data/{filename}.json", "w") as fp:
            json.dump(data, fp)

    @staticmethod
    def __format_tf_df(term: str, tf: int, df: int) -> str:

        return f"{term:<12} | {tf:<6} | {df:<6}"

    @staticmethod
    def print_stats(filename: str, num_docs: int, lex: Lexer) -> None:

        print("Processed", num_docs, "documents.")
        IO.__dump_json(filename + "_tf", lex.get_tf())
        IO.__dump_json(filename + "_df", lex.get_df())

        with open(f"data/{filename}_stats.txt", "w") as fp:

            print("----------------------------", file=fp)
            print(num_docs, "documents.", file=fp)

            print("----------------------------", file=fp)
            print("Collections size:", lex.get_collection_size(), file=fp)
            print("Vocabulary size:", lex.get_vocab_size(), file=fp)
            print("\n----------------------------", file=fp)

            print("Top 100 most frequent words:", file=fp)
            print(IO.__table_header, file=fp)
            for term in lex.get_top_n_tf_df(100):
                print(IO.__format_tf_df(*term), file=fp)

            print("\n----------------------------", file=fp)
            print("500th word:", file=fp)
            print(IO.__table_header, file=fp)
            print(IO.__format_tf_df(*lex.get_nth_freq_term(500)), file=fp)

            print("\n----------------------------", file=fp)
            print("1000th word:", file=fp)
            print(IO.__table_header, file=fp)
            print(IO.__format_tf_df(*lex.get_nth_freq_term(1000)), file=fp)

            print("\n----------------------------", file=fp)
            print("5000th word:", file=fp)
            print(IO.__table_header, file=fp)
            print(IO.__format_tf_df(*lex.get_nth_freq_term(5000)), file=fp)

            print("\n----------------------------", file=fp)
```

```
            single_occs: int = lex.get_single_occs()
            print(
                "Number of words that occur in exactly one document:", file=fp
            )
            print(
                single_occs,
                f"({round(single_occs / lex.get_vocab_size() * 100, 2)}%)",
                file=fp,
            )
```

Code Listing 2: ./src/normalize.py

```
import re
from typing import Iterable

import nltk

CONTRACTIONS: dict[str, str] = {
    "aren't": "are not",
    "ain't": "is not",
    "can't": "cannot",
    "couldn't": "could not",
    "didn't": "did not",
    "doesn't": "does not",
    "don't": "do not",
    "hadn't": "had not",
    "hasn't": "has not",
    "haven't": "have not",
    "he'd": "he had",
    "he'll": "he will",
    "he's": "he is",
    "i'd": "i had",
    "i'll": "i will",
    "i'm": "i am",
    "i've": "i have",
    "isn't": "is not",
    "it's": "it is",
    "let's": "let us",
    "mightn't": "might not",
    "mustn't": "must not",
    "shan't": "shall not",
    "she'd": "she had",
    "she'll": "she will",
    "she's": "she is",
    "shouldn't": "should not",
    "that's": "that is",
    "there's": "there is",
    "they'd": "they had",
    "they'll": "they will",
    "they're": "they are",
    "they've": "they have",
    "wasn't": "was not",
    "we'd": "we had",
    "we're": "we are",
    "we've": "we have",
    "weren't": "were not",
    "what'll": "what will",
    "what're": "what are",
    "what's": "what is",
```

```python
    "what've": "what have",
    "where's": "where is",
    "who'd": "who had",
    "who'll": "who will",
    "who're": "who are",
    "who's": "who is",
    "who've": "who have",
    "won't": "will not",
    "wouldn't": "would not",
    "you'd": "you had",
    "you'll": "you will",
    "you're": "you are",
    "you've": "you have",
}


class Normalizer:
    def __init__(self) -> None:

        self.__document: str = ""
        self.__tokens: list[str] = []

        self.__punc_re: re.Pattern[str] = re.compile(
            '[!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~]'
        )
        self.__sno: nltk.stem.SnowballStemmer = nltk.stem.SnowballStemmer(
            "english"
        )

    def set_document(self, document: str) -> None:

        self.__document = document[:-1]

    def __basic_stem(self, tokens: Iterable[str]) -> list[str]:

        return [self.__sno.stem(token) for token in tokens]

    def __translate_contractions(self, token: str) -> str:

        return CONTRACTIONS.get(token, token)

    def __split_document(self, document: str) -> list[str]:

        return document.split(" ")

    def __remove_punc(self, tokens: str) -> list[str]:

        return self.__punc_re.split(tokens)

    def __to_lower_case(self, document: str) -> str:

        return document.lower()

    def get_tokens(self) -> list[str]:

        return self.__tokens

    def process(self) -> None:
```

```
        self.__document = self.__to_lower_case(self.__document)
        self.__tokens = self.__split_document(self.__document)

        temp_str: str = ""
        for token in self.__tokens:
            temp_str += self.__translate_contractions(token) + " "

        no_puncs: list[str] = self.__remove_punc(temp_str)
        no_empty: Iterable[str] = filter(
            None, self.__split_document(" ".join(no_puncs))
        )
        self.__tokens = self.__basic_stem(no_empty)
```

Code Listing 3: ./src/lexer.py

```
from collections import Counter
from typing import Generator


class Lexer:
    def __init__(self) -> None:

        self.__tf: Counter[str] = Counter()
        self.__df: Counter[str] = Counter()

    def add(self, tokens: list[str]) -> None:

        self.__tf.update(tokens)
        self.__df.update(set(tokens))

    def get_tf(self) -> Counter[str]:

        return self.__tf

    def get_df(self) -> Counter[str]:

        return self.__df

    def get_collection_size(self) -> int:

        return self.__tf.total()

    def get_vocab_size(self) -> int:

        return len(self.__tf)

    def get_top_n_tf_df(
        self, n: int
    ) -> Generator[tuple[str, int, int], None, None]:

        top_n_tf = self.__tf.most_common(n)
        for tf in top_n_tf:
            term, freq = tf
            yield term, freq, self.__df[term]

    def get_nth_freq_term(self, n: int) -> tuple[str, int, int]:

        term, freq = self.__tf.most_common(n)[-1]
        return term, freq, self.__df[term]
```

```python
    def get_single_occs(self) -> int:

        single_occs: int = 0
        for df in self.__df.values():
            if df == 1:
                single_occs += 1

        return single_occs
```

Code Listing 4: ./run.py

```python
from src.io import IO
from src.normalize import Normalizer
from src.lexer import Lexer


if __name__ == "__main__":

    def process_document(filename: str) -> None:

        prep = Normalizer()
        lex = Lexer()
        io = IO()

        line_num = 0
        num_docs = 0

        with open(filename) as fp:
            for line in fp:
                match line_num % 4:
                    case 0:
                        num_docs += 1
                    case 1:
                        prep.set_document(line)
                        prep.process()
                        lex.add(prep.get_tokens())
                    case _:
                        pass
                line_num += 1

        io.print_stats(filename[:-4], num_docs, lex)

    process_document("yelp.txt")
    process_document("headlines.txt")
```

B Outputs

Code Listing 5: Statistics of 'yelp.txt'

```
______________________________
8892 documents.
______________________________
 Collections size: 1294386
 Vocabulary size: 22645

______________________________
```

Top 100 most frequent words:

Word	TF	DF
the	65479	8531
and	41571	8281
i	38650	7676
a	33661	7895
to	28121	7434
was	23336	6044
it	22373	6900
is	21336	6723
of	19586	6503
not	16170	6320
for	14816	6276
in	14048	5993
that	12070	5120
but	11175	5617
have	10608	5283
with	10448	4990
my	10259	4878
this	10201	5402
we	10182	3442
you	9908	4314
they	9429	4567
on	8786	4715
food	8151	4844
had	8051	4292
place	7300	4407
are	7235	4029
were	7068	3468
good	6936	4186
so	6552	3799
at	6419	3858
be	6346	3930
order	5928	3277
as	5580	3015
like	5453	3437
there	5328	3342
just	4589	3041
go	4574	3240
if	4528	3089
get	4443	3016
time	4426	3001
out	4358	2981
do	4264	2888
all	4255	2871
one	4132	2818
here	4129	3007
veri	4089	2706
me	4040	2563
will	3865	2930
our	3778	1955
great	3741	2660
or	3721	2565

```
their          | 3710     | 2344
servic         | 3614     | 2914
when           | 3577     | 2549
from           | 3462     | 2521
would          | 3420     | 2362
restaur        | 3265     | 2126
up             | 3214     | 2298
am             | 3144     | 2270
realli         | 3122     | 2159
tri            | 3101     | 2304
which          | 3095     | 2130
back           | 3038     | 2331
what           | 3038     | 2233
some           | 3023     | 2169
did            | 3006     | 2083
about          | 2992     | 2205
been           | 2780     | 2127
an             | 2764     | 2112
no             | 2719     | 1943
chicken        | 2594     | 1573
friend         | 2507     | 1976
onli           | 2489     | 1999
can            | 2450     | 1874
eat            | 2443     | 1876
other          | 2428     | 1928
more           | 2412     | 1875
your           | 2384     | 1697
even           | 2364     | 1830
pizza          | 2349     | 873
come           | 2343     | 1839
us             | 2311     | 1436
also           | 2264     | 1750
by             | 2259     | 1765
too            | 2230     | 1781
becaus         | 2173     | 1679
got            | 2101     | 1545
he             | 2082     | 1031
fri            | 2072     | 1301
look           | 2068     | 1604
want           | 2035     | 1573
make           | 2029     | 1586
menu           | 2027     | 1522
nice           | 2023     | 1574
love           | 1994     | 1566
tast           | 1993     | 1482
littl          | 1986     | 1531
well           | 1980     | 1559
price          | 1974     | 1621
she            | 1938     | 899

--------------------------------
500th word:
Word           | TF       | DF
--------------------------------
```

```
game            | 345     | 246

________________________________
1000th word:
Word            | TF      | DF
________________________________
somewhat        | 127     | 119

________________________________
5000th word:
Word            | TF      | DF
________________________________
descent         | 8       | 8

________________________________
Number of words that occur in exactly one document:
11074 (48.9%)
```

Code Listing 6: Statistics of 'headlines.txt'

```
________________________________
500000 documents.
________________________________
Collections size: 4664486
Vocabulary size: 121247

________________________________
Top 100 most frequent words:
Word            | TF      | DF
________________________________
to              | 119686  | 110472
in              | 89522   | 84809
the             | 84792   | 73259
of              | 76477   | 70353
for             | 67764   | 65927
and             | 55814   | 51591
on              | 42675   | 41606
a               | 39708   | 36700
at              | 31899   | 31368
with            | 31840   | 31345
new             | 27029   | 26514
2015            | 24141   | 23701
is              | 21299   | 20479
by              | 17562   | 16983
as              | 16728   | 16122
from            | 16194   | 16054
market          | 14811   | 12927
after           | 13023   | 12975
it              | 12180   | 11716
announc         | 11090   | 11065
be              | 10942   | 10818
up              | 10800   | 10654
over            | 10006   | 9960
say             | 9871    | 9824
not             | 9639    | 9476
```

year	9436	9244
report	8916	8754
will	8802	8722
1	8648	8225
day	8560	8397
global	8268	8176
man	8210	8125
2	8029	7774
you	7987	7440
open	7930	7875
us	7864	7765
get	7837	7784
more	7682	7577
out	7639	7584
first	7589	7531
your	7443	7142
are	7358	7221
world	7263	7155
septemb	7257	7199
win	7223	7152
week	6876	6749
3	6808	6612
how	6781	6734
s	6769	6643
polic	6580	6519
this	6501	6444
launch	6426	6414
make	6398	6366
5	6268	6106
show	6138	6058
share	5886	5763
school	5817	5645
about	5741	5686
take	5725	5714
state	5712	5572
servic	5694	5565
u	5648	5560
back	5609	5536
home	5590	5476
video	5565	5495
top	5563	5500
plan	5515	5452
busi	5489	5329
one	5474	5284
time	5465	5377
industri	5403	5281
no	5290	5067
off	5285	5214
game	5243	5092
china	5216	5109
10	5120	5056
inc	5062	4493
that	5048	4989
4	5011	4919

has	5005	4967
help	4991	4949
call	4953	4925
an	4952	4892
have	4929	4860
set	4905	4889
citi	4891	4776
what	4887	4817
group	4880	4790
stock	4777	4708
into	4771	4760
live	4756	4673
review	4718	4691
updat	4710	4689
i	4667	4070
research	4666	4556
against	4662	4647
two	4645	4582
rate	4643	4551
fall	4635	4613
kill	4614	4597

500th word:

Word	TF	DF
st	1524	1501

1000th word:

Word	TF	DF
construct	831	822

5000th word:

Word	TF	DF
laud	99	99

Number of words that occur in exactly one document:
59030 (48.69%)