COMP 479 - Project 2

François LaBerge

October 23, 2020

1 Running the Program

See the "README.md" file packaged with the application.

2 Lossy Compressions Table

	(distinct) terms			nonpositional postings			tokens		
	number	Δ%	T%	number	Δ%	T%	number	Δ%	T%
unfiltered	60,972			1,783,275			2,871,145		
no numbers	58,759	-4	-4	1,621,515	-9	-9	2,644,815	-8	-8
case folding	42,720	-27	-30	1,498,356	-8	-16	2,644,815	-0	-8
30 stop words	42,690	-0	-30	1,291,577	-14	-28	1,953,585	-26	-32
150 stop words	42,570	-0	-30	1,153,316	-23	-35	1,754,517	-34	-39
stemming	31,843	-25	-48	1,095,928	-5	-39	1,754,517	-0	-39

3 Code Design

In this section we will go over the core parts of the code design

3.1 Creating the Index

3.1.1 Document Parsing

To parse the documents of Reuters, we reuse the code from project 1:

```
stream = stream_filters.tokenizer(
                        solutions.block_extractor(
                                solutions.block_document_segmenter(
                                        solutions.block_reader(path)
                                )
                        )
                )
```

The tokenizer for the has been replaced with a regular expression tokenizer to easily discard punctuation sings.

```
tokenizer = RegexpTokenizer('[A-Za-z]+|[0-9]+')
```

This regular expression will match with any string of strictly characters or strictly numbers.

3.1.2 Applying Filters & Lossy Compressions

To apply the lossy compression algorithms we use a list of lambdas that apply filters to the stream of token-docId pairs. The list is called "preprocessings". Filters are added through the command line interface.

```
args = parser.parse_args()

if args.removenumbers:
        preprocessings.append(stream_filters.remove_numbers)
if args.casefolding:
        preprocessings.append(stream_filters.case_folding)
if args.stopwords is not None:
        preprocessings.append(stream_filters.remove_stopwords)
        args.stopwords = open(args.stopwords, 'r').read()
if args.stemming:
        preprocessings.append(stream_filters.stem)
```

The filters are then applied with a simple for loop.

```
for filter in preprocessings:
        stream = filter(stream, stopwords)
```

The implementation of the filters is in "$ir/p2/stream_filters.py$"

3.1.3 Sorting and Removing Duplicates

The removal of duplicates is done with the python "set" function and the sorting is done with python built-in "sorted" function. The whole ordeal is dealt with in a single line

```
return sorted(set(stream), key=lambda element: (element[1], element[0]))
```

Notice that we first sort by term, then by docID.

3.1.4 Creating the naive index

A python dictionary is used to create the inverted index. The dictionary has the advantage of being easy to implement and most importantly has constant lookup times. In this case, the collection is small enough so that the expected issues of high memory usage and collisions are negligible here.

```
index = {}
for term in terms:
    if not term[1] in index:
        index[term[1]] = [term[0]]
    else:
        index[term[1]].append(term[0])

terms, postings = (count_terms(index), count_postings(index))
print('terms: {}\tpostings: {}\ttokens: {}'.format(terms, postings, tokens))

with open(output_index, 'wb') as f:
    pickle.dump(index, f, pickle.HIGHEST_PROTOCOL)
```

The index is serialized and saved on disk in python's "pickle" format to be easily reusable.

3.2 Querying the Index

Since we used a dictionnary as the datastructure for our inverted index, perform single term queries is trivial. We first load the dictionnary in memory.

```
        def load_index(index_file):
                with open(index_file, 'rb') as f:
                        return pickle.load(f)
```

we can then use the input terms to as the dictionnary indices to retrieve their postings list.

```
def query_term(terms, index_file, output_file):
    index = load_index(index_file)
    result = {}
    for term in terms:
        result[term] = index[term] if term in index else []
    print(json.dumps(result, indent=3), file=output_file)
```

4 Sample Queries

For our sample queries we chose the words "the", "man", and "carrot". For the regular index we get the following results.


```
{
   "the": [
      1,
      2,
      3,
          [... 21571 lines omitted]
      21578
   ],
   "man": [
      28,
      177,
      269,
      1410,
      [... 55 lines omitted]
      21498
   ],
   "carrot": []
}
```


Note that many lines were omitted in the snippet above for readablity, see "sampleQueriesNormal.json" and "sampleQueriesCompressed.json" for the full results.


```
{
   "the": [],
   "man": [
      486,
      3759,
      3815,
      8780,
      16562,
      19531,
      19983,
      21013
   ],
   "carrot": [
      9980
   ]
}
```


Let's begin our comparison from top to bottom. Unsurprisingly, the term "the" appears in almost every document in the corpus as we can see from the query on the uncompressed index. However, no results are found "the" in the compressed index since it is the first of stopwords to be removed.

The term "man" is a little more interesting, we can see that the numbers of terms were greatly reduced from the regular to the compressed index. This is strange, because casefolding is should have increased the length of the postings list if anything. Additionally, the smaller postings list is not a subset of the larger postings list which is even more strange. However, looking at the list of 150 stopwords, on can see that the word "man" is #43 of the most used. So then why was it not entirely removed? Running the query "man" on the index without stemming reveals the answers, as we get the result: []. Stemming is the culprit then.

Observing the document #486 we can see the word “manning” and “management”. This means that the PorterStemmer reintroduced a word from the stopwords list by stemming words starting with “man”.

We can see that the term “carrot” increased its postings list from 0 to 1 in the compressed index. Looking at the document #9980 one can read

> Named in the suit were <Carrot Components Corp¿, <ODU-Kontact GmbH and Co¿ and <KG and G and H Technologies Inc¿, it said.

As expected, the postings list of “Carrot” was merged with the one of “carrot”.