COL764 , Assignment1

Mihir Gupta

Language used: Python3, OS – Windows10

1)invidx.py

Libraries imported : BeautifulSoup, nltk,nltk.corplus , os,pickle,sys,string

(lxml was also used in BeautifulSoup and may need to be downloaded)

Libraries installation

pip install bs4

pip install nltk

pip install lxml

pip install heapdict

Executable launching

python invidx.py <coll-path> <indexfile>

where,

coll-path specifies the directory containing the files containing documents of the collection,and

indexfile is the name of the generated index postings.

All tagged files in the folder containing the files are parsed using BeautifulSoup library as a xml document. For this to be done a dummy start tag was also inserted at the beginning of the contents read from each file along with the corresponding closing tag in the end. Tagged words from the Location,Person,Organisation tags in the text tags for each document were stored in a dictionary of dictionary. This consisted of words as keys and dictionaries of documents and count of that word in the documents was stored and updated for each word. Data of words and their labels was also stored in a separate dictionary. Stop words, punctuations and some other redundant words like "''","'d'","'s","-lrb-","-rrb-","``" were excluded. First thing dumped in the .dict file was the number of documents(N) . After that arrays consisting of keyword ,f(number of documents having that keyword),offset to postings list in .idx file and tag were dumped to .dict file in binary mode using pickle for each word and simultaneously postings lists consisting of a dictionary of the documents in which word occurs and their counts was dumped in binary mode using pickle from the data of the 2 dictionaries created while parsing the input file .

2)**vecsearch.py**

Libraries imported :

```
import nltk
from nltk.tag.stanford import StanfordNERTagger
from nltk.stem import PorterStemmer
import heapdict
import sys
import math
import pickle
```

It is assumed in the code that the files "stanford-ner-4.0.0.jar", "english.all.3class.distsim.crf.ser.gz" for using the StanfordNERTagger are present in the same directory as vecsearch.py .

Note : Query file is assumed to be precisely of the same format as the topics file that has been given to us.

Executable launching

python vecsearch.py --query <queryfile> --cutoff <k> --output <resultfile> --index <indexfile> --dict <dictfile>

command-line arguments:

--query queryfile a file containing keyword queries, with each line corresponding to a query

--cutoff k the number k (default 10) which specifies how many top-scoring results have to be returned for each query

--output resultfile the output file named resultfile which is generated by your program, which contains the document ids of all documents that have top-k (k as specified) highest-scores and their scores in each line (note that the output could contain more than k documents). Results for each query have to be separated by 2 newlines.

--index indexfile the index file generated by invidx cons program above

--dict dictfile the dictionary file generated by the invidx cons program above

(k is by default set to 10)

First total number of documents is loaded from the .dict file . Then a dictionary with key as word and array consisting of f,offset in .idx file and tags is created from the data of the.dict file and a

dictionary consisting of words as keys and postings list as values are created using the data from the .idx using seek functions with stored offsets for data of each key .

For every word , idf is calculated as N/log2(f) where f is the number of documents in which word occurs and N is the total number of documents and stored

Data for each word is traversed and a dictionary of document as keys and values as dictionary of

Word and tf values are stored . Where tf = 1 + log(fij) where fij is the frequency of word in that particular document.

Next mod of each document vector is calculated as sum of squares of product of tf and idf values for each word that occurs in the document , and finally its square root is stored .

Using porter stemmer, stem of each word is stored and an inverted dictionary of stem and list of words that have that stem is also stored.

A trie is implemented in the standard way along with a function that returns list of word that have a given prefix.

Contents of the query file are analysed line by line and if the first word on splitting is the num tag , then the third word is stored as the query id.

If the first word is title tag , then the query starts from the 3rd word . Each word in the query is tagged by the StanfordNer tagger as Organization,Person,Location or others .

Stem of each word in the query is also stored.

Next each word in query is checked if it is present in the dictionary then it is added to the list of potential matches or if stem of a query word matches with a stem in the list of stems,then the corresponding in dictionary having that stem are added to the list of potential matches if not already present.

If a word in query contains '*' then a prefix search is done on the substring of the word before "*" and the words returned are added to list of potential matches.

For each word in potential matches we check whether the tags match to the query word,however If query word is tagged as 'other' than it matches with all tags . Matched words are then stored in a list. For every matched word we store and update its count for the given query.

If there is no matched word, then query has a null vector and Cosine is assumed to be 0 with all documents.

Else for each matched word in the dictionary , we calculate the corresponding weight of the query vector by calculating tf and multiplying with idf and store it and use it to calculate its product with the correspond weight of the document vector of each document to calculate dot product and also the mod of the query vector .

Finally cosine with each document vector is calculated and stored in a priority queue using heapdict in python. Documents with the top k cosine are added to the result file as retrieved from the heapdict.

3**) printdict.py**

Executable launching

python printdict.py <indexfile.dict>

File reads the data of the file indexfile.dict using pickle . First entry that is the number of documents is ignored. Next the arrays storing data for each key word(indexterm) are read and for each key word

<indexterm>:<df>:<offset -to-its-postingslist-in-idx-files>

is printed till end of file is reached.