EPL660: Information Retrieval and Search Engines – Lab 1

University of Cyprus
Department of
Computer Science

Παύλος Αντωνίου

Γραφείο: B109, ΘΕΕ01

General Info

- Course website and material: https://www.cs.ucy.ac.cy/courses/EPL660

- Lab Instructor Information:
  - Email: paul.antoniou-AT-cs.ucy.ac.cy,
  - Office: B109 (Basement), Building ΘΕΕ01/FST01

Lab Content

- Tools for Information Retrieval and processing
  - Inverted index and Boolean retrieval model
    - NLTK Python library / Apache OpenNLP library
  - Apache Lucene: Full-text search library written in Java
  - Apache Hadoop: Distributed storage and processing of dataset of big data using MapReduce model
  - ElasticSearch: Distributed search engine based on Apache Lucene
  - Apache Spark: General engine for big data processing

Information Retrieval (IR)

- In information retrieval (IR), we are interested to extract information resources (e.g. documents) relevant to an information need (query)

- Huge amounts of data are now on-line
  - much of it is *unstructured text*
  - IBM reported in June 2012 that 90% of data available created in the past 2 yrs

Data Mining & Machine Learning

- Data mining (DM): discover the properties of large data sources
  - Big data: the data is so large that standard techniques (hardware, algorithms, etc.) cannot be used
  - Distributed storage & processing needed
  - Uses Machine Learning or other techniques (e.g. data visualization)
- Machine Learning (ML) is one source of tools used to solve problems in Information Retrieval
  - Design algorithms that can learn from experience and make predictions on new data (supervised learning)
  - Used to extract patterns from data (unsupervised learning) to learn more about the data

Boolean retrieval model

- The Boolean Model (BM) is arguably the simplest model to base an information retrieval system on
- First and most adopted
- Queries are Boolean expressions
  e.g., *Caesar AND Brutus*
- The search engine returns ALL documents that satisfy the Boolean expression.

Does Google use the BM?

Does Google use the BM?

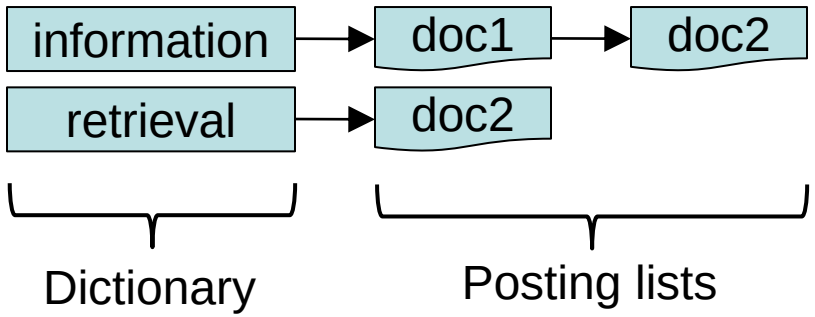
- On Google, the default interpretation of a query `[w1 w2 . . .wn]` is `w1 AND w2 AND ... AND wn`
- Cases you get hits that do not contain one of `wi` :
  - anchor text I like reading interesting books.
  - page contains variant of `wi` (morphology, spelling correction, synonym)
  - long queries (n large) – Google may ignore some terms
  - boolean expression generates very few hits
- Simple Boolean vs. Ranking of result set
  - Simple Boolean retrieval returns matching documents in no particular order
  - Google (and most well-designed Boolean engines) rank the result set – they rank good hits (according to some estimator of relevance) higher than bad hits.

Boolean Queries

- The Boolean retrieval model can answer any query that is a Boolean expression
  - Boolean queries are queries that use `AND`, `OR` and `NOT` to join query terms
  - It is precise: Returns documents that match condition
- Primary commercial retrieval tool for 3 decades
- Many professional searchers (e.g., lawyers) still like Boolean queries
  - You know exactly what you are getting

Indexing

- Searching terms within huge number of documents is a very slow process
- Idea behind indexing for information retrieval
  - build an inverted index to speed retrieval
    - a mapping from the terms to the respective documents containing them



  - building the index is slow, but it only needs to be built once
  - index can be built off-line, i.e., before queries have been seen

Doc retrieval using inverted index

- An inverted index maps terms to the documents that contain them
  - "inverts" the collection (which maps documents to the words they contain)
  - permits to answer boolean queries without visiting entire corpus
- An inverted index is slow to construct (requires visiting entire corpus)
  - but this only needs to be done once
  - can be used for any number of queries
  - can be done before any queries have been seen
- Usually the dictionary is kept in RAM, but the postings lists can be stored on hard disk

Inverted index construction

1. Collect the documents to be indexed:
   `Friends, Romans, countrymen.`
   `So let it be with Caesar.` . . .
2. Tokenize the text, turning each document into a list of tokens:
   `Friends` `Romans` `countrymen` `So` . . .
3. Do linguistic preprocessing (lowercasing, stop word removal, stemming, …), producing a list of normalized tokens, which are the indexing terms:
   `friend` `roman` `countryman` . . .
4. Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

Task (to be completed on next Lab)

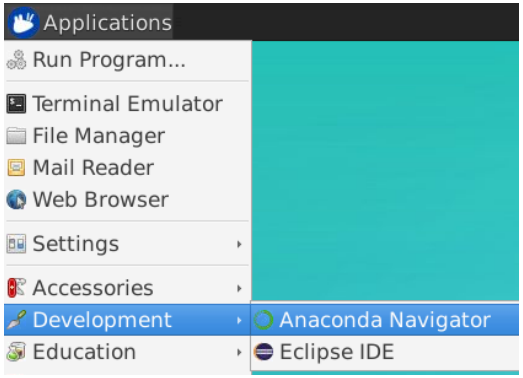
- Build a system that :
  - reads a dataset (corpus) of multiple text files
  - preprocess data
  - create dictionary and inverted index
  - use Boolean retrieval model to pose queries and get results
- Available tools:
  - Natural Language ToolKit (NLTK) (Python)
  - Apache OpenNLP (Java)

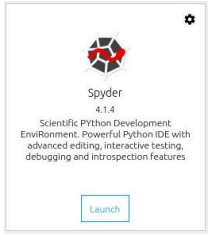
What is NLTK?

- Python interface to over 50 corpora and lexical resources
- Suite of libraries (models) for a variety of academic text processing tasks:
  - tokenization, stemming, tagging,
  - chunking, parsing, classification,
  - language modeling, logical semantics
- Pedagogical resources for teaching NLP theory in Python ...

Installation

- NLTK installation using pip (given that Python is already installed on your machine)
  - sudo pip install -U nltk
- Corpora and models installation
  - Open Python IDLE or Spyder (if you have Anaconda installed) and run the following commands in console
  - import nltk
  - nltk.download()
- Anaconda (Python 3.8, Spyder IDE, NLTK) already installed on the given VM





(Some) Modules in NLTK

Language Processing Task	NLTK module	Some functionalities
Accessing corpora	nltk.corpus	Standardized interfaces to corpora and lexicons
String processing	nltk.tokenize	Sentence and word tokenizers
	nltk.stem	Stemmers
Part-of-speech tagging	nltk.tag	Various part-of-speech taggers
Classification	nltk.classify	Decision tree, maximum entropy
	nltk.cluster	K-means
Chunking	nltk.chunk	Regular expressions, named entity tagging

Apache OpenNLP™

- Machine learning based Java toolkit for the processing of natural language text
- Supported NLP tasks
  - Tokenization / Sentence segmentation
  - Part-of-speech tagging
    - marks each word in a sentence with the word type (noun, verb, adjective, …)
  - Named entity recognition
  - Chunking
    - breaks sentence into groups (of words) containing sequential words of sentence, that belong to a noun group, verb group, etc
  - Lemmatization
    - remove any changes in form of the word like tense, gender, mood, etc. and return dictionary or base form of word
  - Language detection

Installation

- OpenNLP 1.9.3 already installed on VM and associated with Eclipse
- To install on your PC:
  - Binaries for OpenNLP 1.9.3 from https://mirror.library.ucy.ac.cy/apache/opennlp/opennlp-1.9.3/apache-opennlp-1.9.3-bin.tar.gz
  - Open Eclipse and add `lib/` folder into classpath
    - Window → Preferences → Java → Installed JREs
    - Double click on the java-8-openjdk-amd64
    - Add External JARs
    - Select apache-opennlp-1.9.3/lib/opennlp-tools-1.9.1.jar file
    - Press OK, Finish, OK

Hands on

- Run the NLTK examples shown in the next slides

- Download OpenNLP.zip eclipse project to experiment with SimpleTokenizer, WhitespaceTokenizer and DictionaryLemmatizer classes

NLTK: Corpora

- Task: Accessing corpora, stopwords
- NLTK module: `nltk.corpus`
- Functionality: interfaces to corpora, lexicons, stopwords
  - Load Gutenburg corpus from NLTK (full corpus here)

```
# import gutenberg corpus (collection)
from nltk.corpus import gutenberg, stopwords
# documents in gutenburg collection
all_filenames = gutenberg.fileids()
print(all_filenames)
```

```
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', 'bible-
kjv.txt', 'blake-poems.txt', 'bryant-stories.txt', 'burgess-busterbrown.txt',
'carroll-alice.txt', 'chesterton-ball.txt', 'chesterton-brown.txt',
'chesterton-thursday.txt', 'edgeworth-parents.txt', 'melville-moby_dick.txt',
'milton-paradise.txt', 'shakespeare-caesar.txt', 'shakespeare-hamlet.txt',
'shakespeare-macbeth.txt', 'whitman-leaves.txt']
```

NLTK: Corpora

```
# full text of shakespeare-hamlet.txt document
hamlet = gutenberg.raw('shakespeare-hamlet.txt')
# list of words in hamlet document
hamlet_words = gutenberg.words('shakespeare-
hamlet.txt')
# list of stop words
stopwords = stopwords.words('english')
print(stopwords[0:30])
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
"you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself',
'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers',
'herself', 'it', "it's", 'its', 'itself']
```

NLTK: Text Processing

- Task: Text processing
- Modules: `nltk.tokenize`, `nltk.stem`
- Functionality: word tokenizers, sentence tokenizers, stemmers, n-grams splitters

```
from nltk import word_tokenize, sent_tokenize
text1 = word_tokenize("The quick brown fox jumps over the lazy dog")
print(text1)
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
text2 = sent_tokenize("The quick brown fox jumps over the lazy dog. What a lazy dog!")
print(text2)
['The quick brown fox jumps over the lazy dog.', 'What a lazy dog!']
```

NLTK: Text Processing

```
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
print(stemmer.stem('processing'))
```

```
process
```

Stemming: the automated process which produces a base string

```
sentence = "This is my sentence and I want to ngramize it."
n = 6
w_6grams = ngrams(sentence.split(), n)
for grams in w_6grams:
    print(grams)
```

Ngramize: the process which produces a contiguous sequence of n items from a given sample of text or speech

```
('This', 'is', 'my', 'sentence', 'and', 'I')
('is', 'my', 'sentence', 'and', 'I', 'want')
('my', 'sentence', 'and', 'I', 'want', 'to')
('sentence', 'and', 'I', 'want', 'to', 'ngramize')
('and', 'I', 'want', 'to', 'ngramize', 'it.')
```

NLTK: Exploring corpora

- When starting to explore a corpus you may ask:
  - How many total words does the corpus have?
  - How many unique words does the corpus have?
  - What are the counts for the 10 most frequent words?
- Task: Corpus frequency distribution
- NLTK module: `nltk.FreqDist`

```
from nltk import FreqDist
from nltk.tokenize import word_tokenize
data = "I like this course and I hope to pass this semester."
words = word_tokenize(data)
fdist = FreqDist(words)
print(fdist.N()) # Prints total number of tokens
print(fdist.most_common(2)) # Prints 2 most common tokens
```

OpenNLP: Simple Whitespace tokenizer

```
import opennlp.tools.tokenize.WhitespaceTokenizer;

public class WhitespaceTokenizerExample {
        public static void main(String args[]) {

                String sentence = "Hi. How are you? Welcome to EPL660. " +
"We are learning natural language processing tools using java language.";

                // Instantiating whitespaceTokenizer class
                WhitespaceTokenizer whitespaceTokenizer =
WhitespaceTokenizer.INSTANCE;

                // Tokenizing the given paragraph
                String tokens[] = whitespaceTokenizer.tokenize(sentence);

                // Printing the tokens
                for (String token : tokens)
                        System.out.println(token);
        }
}
```

More info

- Java PorterStemmer implementation: https://tartarus.org/martin/PorterStemmer/java.txt
- OpenNLP official tutorial and examples: https://www.tutorialkart.com/opennlp/apache-opennlp-tutorial/
- OpenNLP tutorial and examples: https://www.tutorialspoint.com/opennlp/opennlp_tutorial.pdf

Next Week's Lab

- Write simple Python programs using NLTK following the steps towards creating an inverted index
- Programs must be submitted to Moodle