**********README**********
Course: COMS E6998.006 – Search Engine Technology (Fall 2013)
UNI: yp2348
Name: Yash Parikh
Assignment 1: Building a Basic Information Retrieval System
Prof. Dragomir Radev
****

Programming Language: Python 2.7.3
Used an external text editor: Sublime Text 2
External Modules: Porter Stemmer, Levenshtein distance (for similarity using edit distance)
Python Modules: sys, os, time, shlex, re, cPickle as pickle, xml.etree.ElementTree
Note: NLTK has not been used

Assumptions: -

1. Since the Cranfield data set does not have unicode characters, cPickle is used since it is faster than pickle
2. Since the files in the Cranfield data set are small, the entire file is read while parsing instead of reading it line by line

Submission Contents: -

1. **yp2348_inverted_index.py** – Used to build the Inverted Index. Takes the path to the corpus as a command line argument (argv[1]).
   Example: python yp2348_inverted_index.py /home/cs6998/data/Cranfield_Collection/cranfieldDocs/
   Note: The path name should end with a slash (/)

2. **yp2348_query.py** – This is used to query the corpus using the inverted index that was built previously. It keeps accepting queries and giving results till "goodbye" is entered which makes the program exit.

3. **Porter.py** and **porter.pyc** – These are external files used for stemming.

Shebang added at the start of all files: #!/usr/bin/python

How to Run: -

1. **Run the inverted index file**: yp2348_inverted_index.py
   1. Command: python yp2348_inverted_index.py <path_to_corpus>
   2. Example: python yp2348_inverted_index.py /home/cs6998/data/Cranfield_Collection/cranfieldDocs/
   3. In the above example, the path to the corpus is a command line argument (argv[1])
   4. **NOTE:** The path to the corpus should have a trailing slash
   5. When it is run, it builds the index and creates and pickle file to store the index
   6. It also returns statistics such as time to build the index, size of the index file, number of documents indexed and the number of words indexed.
   7. **NOTE:** The document titles and authors have also been added to inverted indexed and are considered in the ranking.

2. **Run the query file**: yp2348_query.py
   1. Command: python yp2348_query.py
   2. The program will keep accepting queries till you enter "goodbye"
   3. Sample Inputs: -

1. !“boundary layer” jet
2. “destalling lift”
3. gradient “exit angle”

4. Sample Special Queries: -
    1. tf 1 destalling
    2. freq “boundary layer”
    3. title 5
    4. df layer
5. Sample Similarity Queries: -
    1. similar layer
    2. similar wing
6. It returns the documents matching the query in ranked order along with the snippet.
7. **NOTE:** If the document is displayed only because of a negation query, the snippet will contain the start of the document.
8. The snippet may display the title of a document if the query contains the document title
9. It also reports the total number of documents found and the time taken to execute the query.

Program Logic: -

Please refer to commenting in the code for detailed program logic.
The basic steps are shown below: -

I. Building the Index: -

1. Take the path to the corpus as a command line argument
2. Parse the corpus document by document (XML parsing)
3. Tokenize the file (including stemming and removal of stopwords)
4. Build the inverted index in a hash table using postings list for each word
5. Postings list is of the form [[doc_number,[position1, position2,...]], [...], ….]
6. Dump the inverted index into a 'pickle' file

II. Query the Index: -

1. Take the query from the user
2. Split the query into words (a phrase will be counted as a word and processed separately) and stem the query as well as remove stopwords
3. Load the inverted index from the pickle file
4. Get results of words – A
5. Get results of negation of words – B
6. Get results of phrases – C
7. Get results of negation of phrases – D
8. Result = (A U B U C U D)
9. A global variable is used for ranking which is called during the processing of A, B, C and D
10. Print the document number along with the snippet in a ranked fashion
11. Print the total number of documents in the result
12. Print the time taken to execute the query
13. Special queries and similarity queries are processed separately and return results independent of the above process
14. Similarity measure chosen is edit distance = 1

External Sources: -

1. http://hetland.org/coding/python/levenshtein.py – Levenshtein distance code for calculating similarity with edit distance = 1
2. https://pypi.python.org/pypi/stemming/1.0 – Used for getting the Porter stemmer external file.