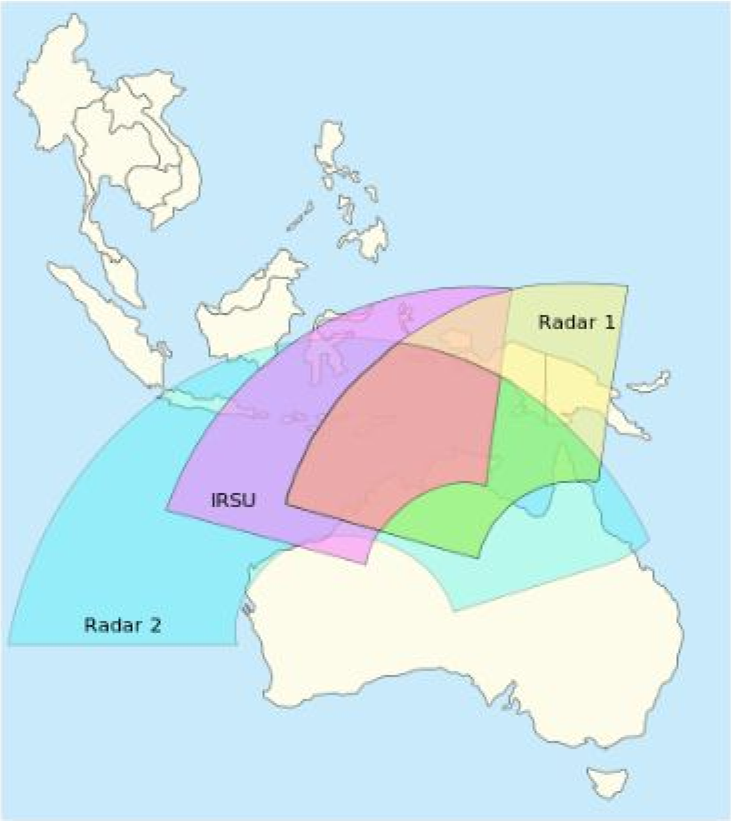
Jindalee

CSI4107 Search Engine Project

Vanilla System



The JORN area of operation.

[https://en.wikipedia.org/wiki/Jindalee_Operational_Radar_Network]

Winter 2020

Group SE 13

Tiffany Maynard 300047456
Jonathan Boerger 300098639

Project Dependencies

As outlined in the readme file, the following modules are required for the code to function:

- Numpy (pip install numpy)
- Natural Language Toolkit (pip install nltk)
- Beautiful Soup 4 (pip install bs4)
- Pandas (pip install pandas)

Modules

User Interface (gui.py)	
Who was in charge	Tiffany
Functionality	Allow a user to access the search engine capabilities
Assumptions	Assumes user will input Boolean operators in uppercase
Limitations	Does not allow looking at multiple full document results at once
Examples of what it does (input/output)	Opens a window for the user to enter a search query and then calls search and document retrieval modules to display results Displays spelling suggestions for misspelled query terms (number of suggestions TOP_N_SPELLING = 3 is set in config.py)
Problems Encountered	Minor: When the three spelling suggestions are very, very long the suggestions appear to overlap in th UI

Corpus pre-processing (corpus_preprocessing.py)	
Who was in charge	Jonathan
Functionality	Convert a collection of documents into a formatted corpus
Assumption	All courses which contain no course description are not applicable for the intent of the project.
Limitations	Information was extracted from the HTML page using xpath to navigate to specific html elements. Therefore, this module is only functional with HTML Only extracts information from English courses
Examples of what it does (input/output)	The modules take as input an HTML document, of the specified form, and extracts and places the relevant information (docID, courseID, title and description) into an XML file which forms the corpus for the search engine.
Problems Encountered	French courses are denoted by a (5 or 6 or 7) as the second digit of the course

	code, as such they were easy to identify and omit. However, there were also bilingual courses where in the course description there was both french and english text. Isolating and extracting the english text proved to be difficult. In particular where the text alternated between languages (English, French, English). In such cases it was not possible to create generalities to strip the French which necessitated manual manipulation of the HTML file.

Dictionary building and Inverted Index Construction (build_dictionary_and_index.py)	
Who was in charge	Jonathan
Functionality	Build a dictionary of terms to be indexed Associate dictionary terms to documents Associated bigraph index based on dictionary words
Assumptions	Only pre-processing terms are included in main inverted index
Limitations	If any one of the required indexes (in the form of csv files) are missing, then all indexes are recreated.
Examples of what it does (input/output)	Given a corpus and a set of linguistic processing parameters (e.g. do_stop_word_removal, do_stemming), check to see if files already exist for this corpus/parameter combination, and if not, creates a dictionary with the pre-processing applied. Uses the dictionary to create inverted index file (based on stemmed/lemmatized dictionary) and bigraph inverted index (based on spelling dictionary) saved as csv files.
Problems Encountered	Originally the inverted index was developed to exclude the TF-IDF values which were required for the VSM retrieval module. However, as the project progressed it was decided that the inverted index would incorporate that information, therefore the inverted index construction method needed to be redeveloped. Additionally, in the second evolution of the method, the inherent advantages of dictionaries were leveraged to make the process simpler and more efficient.

Supplemental Module - Linguistic Pre-processing (linguistic_preprocessing.py)	
Who was in charge	Jonathan
Functionality	This module applies linguistic pre-processing on text including: -contraction expansion -tokenizing -normalization (periods and hyphens done separately) -punctuation removal -case folding -stop word removal -stemming or lemmatization

	Also contains the function to split a word into its component bigraphs to allow for wildcard searching
Assumptions	Provided text is in English Assume that the user does not select both lemmatization and stemming as linguistic processing options All word groups separated by a slash are split into two distinct words
Limitations	Linguistic processing from module linguistic_processor.py does not allow changing the order the preprocessing steps are applied (e.g. normalize periods always happens before remove punctuation) Resolves hyphens by splitting the hyphenated word into multiple words There is no mechanism to restrict to only stemming or lemming
Examples of what it does (input/output)	(1)Takes in text and transforms according to provided linguistic processing parameters (LPP). For example "STATE-of-the-a.r.t.s-connections" returns ['state', 'art', 'connect'] when these LPP are set to True: do_normalize_hyphens, do_normalize_periods, do_case_fold, do_stop_word_removal, do_stemming (2) Takes in a word and returns a list of bigraphs. For example ‘cat’ returns [‘$c’,’ca’,’at’,’t$’].
Problems Encountered	Initially the module was developed with all LPP selected, however when testing different combination of LPP it became clear the linguistic processing did not function for all LPP combinations (for example even if the remove period parameter was not selected, but the remove punctuation one was, it would still remove periods as it was considered a punctuation mark). Compound terms separated by a slash were also not being split (e.g. ergodic/absorbing was initially not appearing as ergodic absorbing) Also, the bigraph splitter didn’t initially account for placement of wildcard characters (*), so this was updated.

Corpus Access (corpus_access.py)	
Who was in charge	Tiffany
Functionality	Access documents from the corpus
Assumptions	None
Limitations	Reads the corpus xml file every time documents need to be retrieved
Examples of what it does (input/output)	Takes in a search result as list of (doc_id, score) pairs, retrieves the document information from the corpus xml file, creates and outputs a list of Document objects
Problems Encountered	Needed to add a dummy score for boolean results to keep the input type consistent between boolean and vsm search results

Boolean model of information retrieval (boolean_search.py)	
Who was in charge	Jonathan
Functionality	Implement the boolean retrieval model
Assumptions	Boolean search is being operated by an experienced user and therefore all queries are properly formatted (proper use of brackets and operators).
Limitations	Reads the inverted index from the stored csv each time a query is performed The module only functions if the boolean operators are capitalized (ex AND)
Examples of what it does (input/output)	Takes a query string and corpus and processes the query string into a postfix format and then determines intersection and/or union of document ids returned for each term based on operators given in query string. Ultimately returns a list of document ids to be displayed as the search result
Problems Encountered	The most challenging portion of the boolean module was the conversion of the query from the infix form to the postfix form. I had no previous experience with postfix notation which caused trickle down issues which were difficult to spot when implementing the translation methods. Additionally when it came to integrating the various modules, a dummy relevance score needed to be added to the list of document ids being returned such that it conformed with the return format of the VSM to allow for a common corpus access method and GUI display regardless of the retrieval method.

Wildcard management with additional letter bigram indexing (wildcard_management.py)	
Who was in charge	Jonathan
Functionality	Resolves the wildcards as occurring anywhere in the query word.
Assumptions	This is assumed to work ONLY in the boolean model Provided word contains a wildcard
Limitations	Only works for words with one wildcard
Examples of what it does (input/output)	Given a word and a bigraph inverted index, returns a string of all words that meet the bigraph criteria formatted to contain an OR between each word so that this may later be added as criteria in the boolean retrieval module.
Problems Encountered	Originally the bigraph index was created using words that had already undergone linguistic pre-processing (LPP) and the wildcard word also underwent LPP prior to resolving the wildcard. However, this created an issue where the stemming or lemmatization modified the ending and produced an unintentional narrowing of results. For example the search for ps*logy became ps*logi and would thus no longer return the word psychology. To mitigate this issue, the bigraph index was created based on the spelling dictionary (all LPP steps except stemming and lemmatization).

Vector Space Model (vsm_weight.py and vsm_retrieval.py)	
Who was in charge	Tiffany
Functionality	Creates and includes term weights in the inverted index and implements the Vector Space Model for retrieval Uses cosine similarity* Uses tf-idf weight formula log(1+tf) x log(N/df)
Assumptions	Before calculating similarity, creates a shortlist of docs from the inverted index based only on those that have at least one search term from the query. This will need to be changed if we introduce relevance adjustments in the full system. *Calculated on the vectors formed only from subset of the terms in the query, so the normalized vectors are slightly different than what would be obtained using the full vocabulary. We learned in class that the actual value of the similarity measure was less important than their value relative to others.
Limitations	Reads the inverted index from the csv file every time a query is run
Examples of what it does (input/output)	Input a query string and a corpus and it returns a list of the the top-k document ids and their similarity scores. K is set in config.py (K_RETRIEVAL = 20)
Problems Encountered	Created a new inverted index routine to allow for more straightforward calculation and inclusion of tf-idf weights in the csv

Spelling correction with Weighted Edit Distance(spelling.py)	
Who was in charge	Tiffany
Functionality	Provide spelling suggestions of corrected words to the user
Assumptions	Assumes cost reductions for all transposed vowels. Additional rough-estimate cost reduction entries for consonants were included based on the top frequency entries in the file http://norvig.com/ngrams/count_1edit.txt Reduced edit distance search space with heuristic that assumes that the correct word will start with either the first or second letter of the input word.
Limitations	Does not account for transposition of whole groups of letters Suggestions in the case of multiple misspelled words are paired by frequency (i.e. closest match of word 1 with closed match of word 2
Examples of what it does (input/output)	Input a list of words to check for spelling and a corpus and it will output a list of suggested words for the words that are not found in the spelling dictionary.
Problems Encountered	Determined the need to create a spelling dictionary that includes words from the corpus before lemmatization or stemming in order to provide real word suggestions to the user rather than suggestions like “psychologi”.