# CS 132 (Spring 2017) Programming assignment 3

**Due Tuesday Feb. 28, 12:30 pm**

**Objective: Build a simple Boolean Information Retrieval System**
**Corpus: Wikipedia movie corpus**

**Resources: python 2.7, flask (http://flask.pocoo.org/), nltk (http://www.nltk.org/)**

**Summary: Implement a simple Boolean information retrieval system supporting conjunctive ("AND") queries over terms, and apply it to your wikipedia movie corpus. The system should consist of (1) an indexing module that constructs an inverted index, with term dictionary and postings lists for a corpus and (2) a run-time module that implements a Web UI for searching the corpus, returning a result list and presenting selected documents. Use all naming conventions mentioned below.**

**Corpus details:** Index and query over title and text only. Treat these as a single "field" for the purposes of indexing and retrieval. You will need to tokenize the text and normalize terms. Use stemming and a stop word list (and indicate in your readme.pdf file what stemmer you used and include your list of stop words.

**Queries:** Should be *conjunctive*, i.e., a document must match ALL (non stopword) terms in the query. If a term in the query is a stop word, your results page should indicate it by saying "Ignoring term: <stopword>" and process the query without it. If a term in the query is not in your dictionary, your results page should indicate it by saying "Unknown search term: <term>" and return 0 results.

**Interface**: Use the (flask) web framework to build a web interface with three page types: query, SERP, target article. The query page should include a query box. The search result page (SERP) should include an editable query box and show *total number of hits* and *unordered results*, 10 at a time. Each result should include title and a few lines of text in a readable format (not raw json). A *Next button* should display the next 10 results. Clicking on a title should display the full target document (title, text, and structured fields). Your abstracts do not need to contain snippets containing query terms.

**NOTE:** You should implement posting list intersection yourself and optimize by intersecting the shortest lists first. Do not use any packages or python functions that might do this for you.

**Data structure:** Store your index as a shelf file (class shelve). Make sure the *writeback* parameter is set to false to save time/space. Include in the readme.pdf the expected building time on your machine. You will also need a data structure to store "doc-data", the information needed to present an article to the user. Create your index files offline and open them at run-time.

**Testing:** Develop and test your system using a small database of ~10 hand-made documents before scaling up to your full corpus. Your hand-made documents should contain examples of things you want to test (tokenization, stemming or lemmatization, stop words, variants of the same term, etc.) Include your test corpus as a json file (test_corpus.json) and explain the tests you ran to verify that your system is working in your readme.pdf file.

**Extra credit:** Support querying over the following structured fields: director, starring, location. Your UI should be extended to include optional input for each of these fields. The results should contain all documents that match the general text query *and* all the optional fields. Any field that is empty (including the text field) can be ignored. But if all fields are empty, return 0 results and a message.

**Deliverables:**
Code: Two main python modules. One will create your index (as a shelve file) and doc-data. The second will invoke the flask UI and allow users to query the index. The command line calls for running these modules should be:

```
python boolean_index.py
```

python boolean_query.py

**Readme**: Follow the instructions included in Assignment 2 regarding what should be included in the readme.pdf file. with running instruction, details about the files in the folder, all text normalization details, shelve timing, packages used, test queries examples, and any other thoughts or concerns you like to share.

**Data**: two corpus files (test_corpus.json, films_corpus.json).

**Submission:**

1. **Upload all files (Including readme.pdf) into latte as a single zip file named "<lastname>_<first initial>_boolean".**
2. **Upload readme.pdf separately into the latte assignment titled "readme". (This is for convenience reviewing on latte.)**