CS410 Text Information Systems
Course Project – Final Documentation

Project Title: Text Retrieval Engine for Construction Equipment
Group Name: Kaz Lone Star
Member: Kazuhei Sasaki (ksasaki2@illinois.edu)

Overview: My employer receives ERP data from major equipment rental companies, aggregates the data, and provides benchmark insights for the industry. One of the biggest challenges in this business model is how to normalize equipment information: the benchmarks are calculated based on client-provided make/model information, but different companies have different representations and conventions of equipment information. In addition, there are countless variations of minor trims in the world of construction equipment. For example, Caterpillar 336 and 336GC are essentially treated as the same model, although there could be a very minor difference. My proposed system accepts a set of make/model as a query, calculates similarities with the document database, and returns Top N matching results depending on the parameter and score. In a nutshell, the system is a VSM-based (or similarity-based) model using letter-level n-gram vectors as features.

Code Documentation:

1. /data
   - consolidated_docs.json
     Equipment data collected from constructionequipmentguide.com using scraper.py. There are 17,464 unique make-model pairs.
   - validation_labeled.txt
     Evaluation dataset having 50 make-model pairs for each of the 10 categories*, retrieved from the real ERP data: 500 make-model pairs in total. I manually labeled them for the purposes of evaluation.
     * Tractors / Excavators / Wheel Loaders / Dozers / Articulating Boom Lifts / Skid Steer Loaders / Forklift Trucks / Generators / Scissor Lifts / Backhoe Loaders
2. /src
   - evaluation.py
     A script used for evaluation. It imports utilfuncs and searchfuncs, and for each make-model pair in the dataset, run the search engine and compare to the manually labeled answer (i.e. calculates the rank against the "ground truth"). It calculates Mean Reciprocal Rank for 12 different strategies.
   - scraper.py
     A script to retrieve the source data from constructionequipmentguide.com. Please be mindful when you run this script so as not to overload their web server.
   - searchfuncs.py
     This library consists of two main functions: search_result() and search_result_match_rank(). The former is the main search function. The latter is primarily for evaluation purposes, taking both a query and answer string (or the ground truth) and returning the rank of the estimation.

query	Query, i.e. pair of make and model in one string
n	N or ngram. Default is 3
similarity_func	Similarity function to be used. Default is cosine_similarity
top	Number of records to be returned. Default is 10

cutoff	Cutoff value on the similarity. If supplied, any records having similarities lower than the cutoff will be dropped from the result set. Default is 0.0

- utilfuncs.py
  A set of utility functions to calculate similarities. ngram() takes string and returns its letter-level n-gram list. Any similarity is calculated based on this n-gram list: for example, cosine_similarity() calculates and returns cosine similarity between two lists.
- search_api.py
  A script to implement Web API using FastAPI framework. search_api() receives make, model, top, and cutoff as input parameters, passes them over to search_result(), and returns the results in JSON format.

3. Other
   - index.html
     HTML file for the Web Demo page
   - app.js
     JavaScript file that accesses and retrieves data from the API. This also displays the results in a table format.

Usage: You can take advantage of the system by:

(1) Accessing the Web Demo page
- URL: https://storage.googleapis.com/my-front/index.html
- Put in Make and Model to be searched
- Change Top and Cutoff as needed (optional)

(2) Querying the Web API
- Documentation: https://my-api-y7vscpfafa-ue.a.run.app/docs
- API Endpoint: https://my-api-y7vscpfafa-ue.a.run.app/
- Use POST method

Please check out the video tutorial as well.

Evaluation: I used Mean Reciprocal Rank (Mean AP) as an evaluation method for this task. Using the hand-labeled evaluation dataset (actual) and the system response (estimated), I calculated Reciprocal Rank of each record and averaged it over the entire dataset. There are a couple of important considerations to be noted:

- I set a cutoff threshold on each similarity function. Cutoff means anything below the threshold is not included in the estimation result set.
- For samples that did not match anything due to the low similarity score,
  (1) If the samples were 'actually' not supposed to match anything (i.e. they did not exist in the scraped document set), then give #1 rank.
  (2) If the samples were 'actually' supposed to match something (i.e. they did exist in the scraped document set), then give #10 rank.
  There could be some controversy over this approach. In the future, we could use different indicators such as precision/recall/f-score to evaluate no-match records.

CS410 Text Information Systems
Course Project – Final Documentation

Below table summarizes Mean Reciprocal Rank of different strategies. As you can see, EnsembleRank with cutoff of 0.1 marked the highest score. EnsembleRank is a combination (simple average) of all strategies (i.e. Cosine, Jaccard, Euclidean and Levenshtein). CosineRank_0.1 ranked the 4th, but the score difference is marginal. EnsembleRank is extremely slow compared to CosineRank. Due to the speed, I still decided to use CosineRank as the similarity function for the API with the cutoff being defaulted to 0.1.

Strategy	Mean AP
EnsembleRank_0.1:	0.9289
EnsembleRank_0.2:	0.9284
JaccardRank_0.1:	0.9254
CosineRank_0.1:	0.9247
EnsembleRank_0.3:	0.9232
CosineRank_0.2:	0.9211
CosineRank_0.3:	0.8952
EuclideanRank_0.1:	0.8908
EuclideanRank_0.2:	0.8908
EuclideanRank_0.3:	0.8908
JaccardRank_0.2:	0.8827
JaccardRank_0.3:	0.8696
LevenshteinRank_0.1:	0.8432
LevenshteinRank_0.2:	0.8342
LevenshteinRank_0.3:	0.8109

Future Extensions: Although I worked on this task as a course project for CS410, I would like to continue to develop and improve this system as there is real business demand for this. Here are some ideas for the future:

- Incorporate machine learning – I employed a simple, similarity-based method this time. I would be curious to see if the usage of machine learning technologies, including Large Language Models (LLM), would enhance the performance of this retrieval task.
- Feedback system – I wanted to add feedback mechanism such as Rocchio Feedback. Ideally, the webpage can have upvote/downvote icons next to the results, so that the users can help improve the results. Unfortunately, I did not have time to implement it.

Estimated Work Time:

1. Scrape from constructionequipmentguide.com (5h) → **Completed (5h)**
2. Build an evaluation dataset, such as hand-labeling (5h) → **Completed (8h)**
3. Build prototype retrieval systems on Jupyter Notebook (8h) → **Completed (8h)**
4. Build evaluation strategies (2h) → **Completed (3h)**
5. Finalize the backend algorithms for the retrieval (5h) → **Completed (5h)**
6. Develop API using FastAPI (5h) → **Completed (5h)**
7. (Optional) Develop frontend webpage using the API (10h) → **Completed (10h incl. deployment)**
8. (Optional) Implement user feedback (10h) → **Not doing this time**
9. **Documentation & presentation → Completed (5h)**

Total: 30-50h → **Actual 49h**