Alan Verga
CS6998 Search Engine Fall 2011
Prof. Radev

De Gustibus Final Report

http://www.degustib.us

Contents

Motivation

While Information Retrieval often focuses on discovering the relevant, an interesting sub-field deals with discovering the novel. Typical Recommendation Systems use collaborative filtering methodologies, and as such suffer from traditional "cold-start" problems. Furthermore, such systems are often ill-suited when dealing with vague information needs. A prime example of such an information need is what to eat for lunch. Unlike movie ratings or product reviews, a user's preference for food is rarely static. A cheeseburger may be a favored food, but certainly such an "optimal" recommendation could not be repeatedly eaten everyday. Nor does identifying a penchant for Chinese cuisine infer that a user will desire varied Chinese dishes. "De Gustibus" attempts to address such uncertainty by implementing a browsing session employing implicit relevance feedback, designed to give novel and relevant food suggestions from restaurants in the Columbia University area. Employing K-means clustering and relevance feedback, "De Gustibus" rotates around clusters to maximize the novelty of suggestions and the diversity of results. Selections can then be used to refine browsing and discover similar offerings in the neighborhood.

Tools/Data

Several external tools and packages were used in development. The front and back-ends are run on a LAMP stack, with respective documentation of each component located in the submission directories.

Back-end

CPAN/Clairlib Perl Modules

- URI::URL
- WWW::Curl::Easy
- DBI
- HTML::Entities
- HTML::Strip
- XML::Simple
- XML::Parser::PerlSAX
- Clair::Utils::Stem

C++ / PHP

- Apache Thrift, http://incubator.apache.org/thrift/download/
- Boost 1.46.1 C++ Library, http://www.boost.org/

Front-end

- Google Maps API
- jQuery, http://www.jquery.com

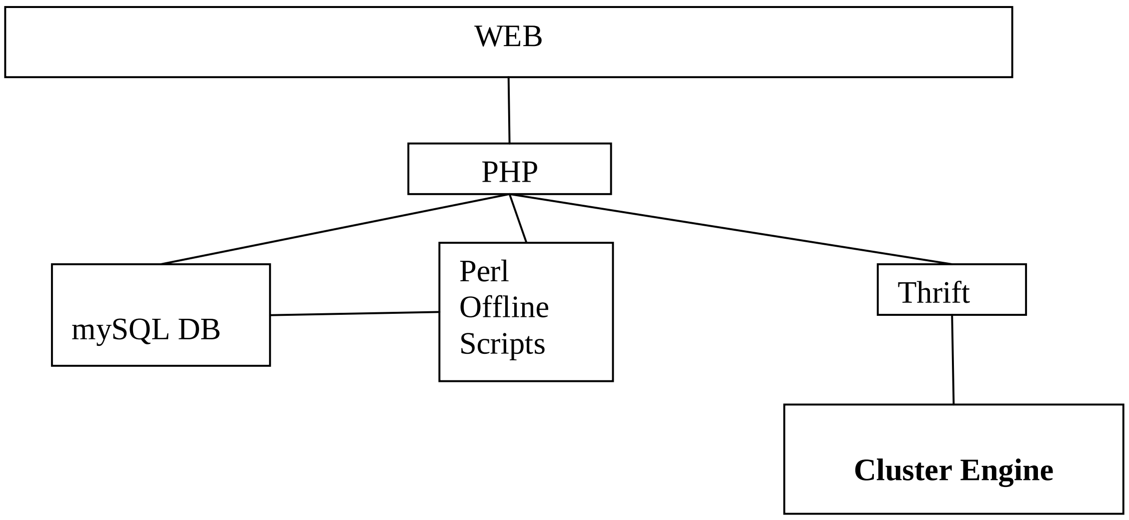
Data Source

Data was solely provided by the delivery.com API.
(https://www.delivery.com/create_api_account.php)

System Description

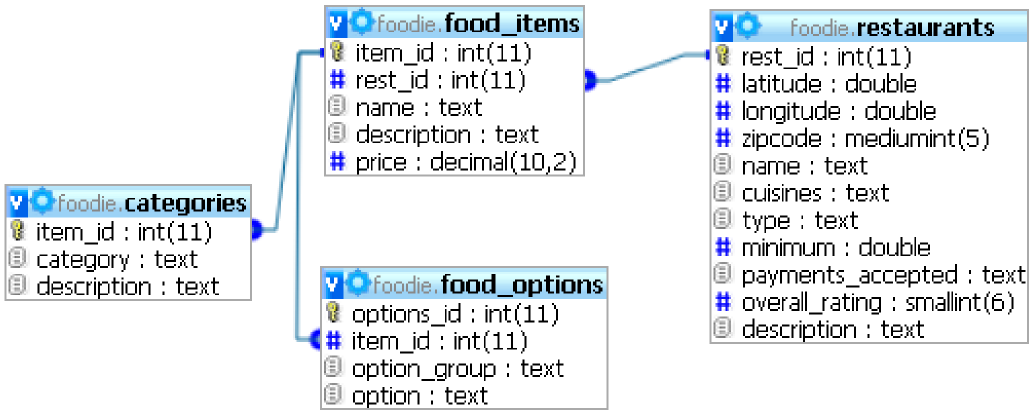
General Architecture

“De Gustibus” relies on Thrift to generate an inter-process RPC interface between PHP and C++, effectively turning the cluster engine into a miniature, local server. PHP client requests can be rapidly fulfilled, as similarity calculations via term-document / document-document matrices can be stored in-memory. Items are given an id, from which additional data can be accessed from the database before final presentation to the user. Perl scripts were used primarily for offline parsing and database maintenance. (See chart below.)



DB Schema

A basic schema for restaurant and menu information storage is shown below. Food_items and restaurants tables are self-explanatory; categories and food_options were secondary tables with meta-data that was not used in the current implementation.



Conceptual Implementation

Implementing "De Gustibus" involved three separate steps; crawling and processing data, implementation of the K-means clustering algorithm and cosine similarity calculations, and interfacing with a final presentation layer.

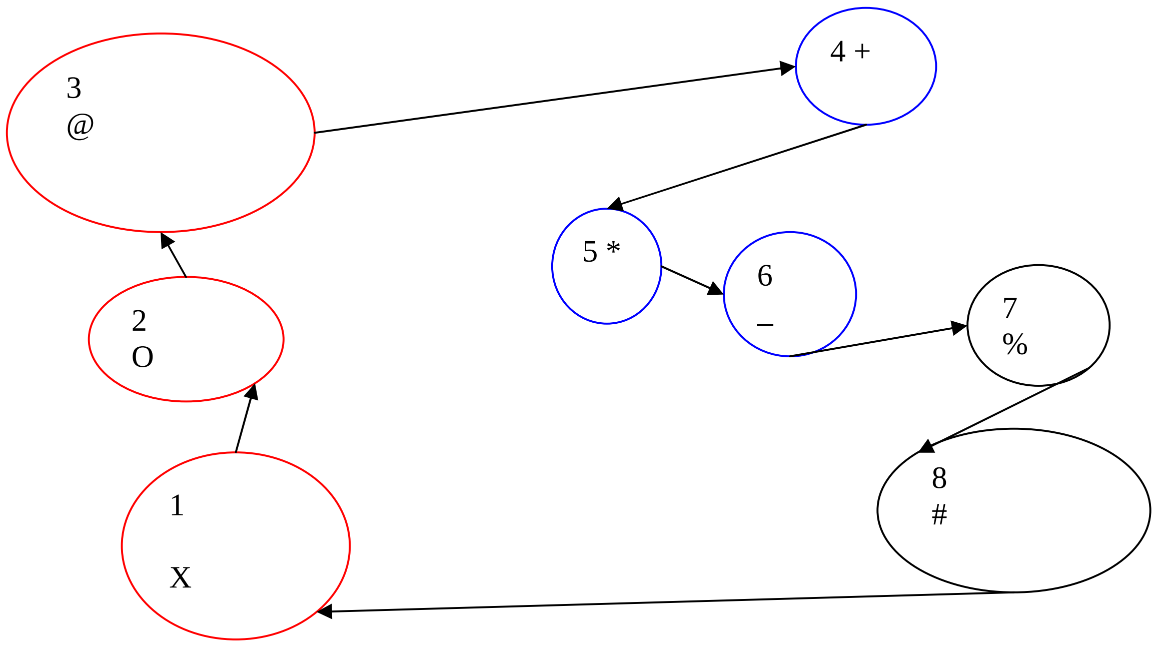
Crawling and Data Processing

Several perl scripts were created to download restaurant information from delivery.com, with restaurants limited to those in zip codes 10025 and 10027. Restaurant information was parsed to build secondary API request strings in order to download actual menu data. This data was parsed using an XML SAX parser, and imported into a mySQL database, according to the schema above. Using the data, a term-document matrix was created from relevant fields (excluding secondary information like "payments accepted.") and exported to a text file for the clustering engine to read. Terms were stemmed using the Clairlib stemmer, with exclusion of common stop words.

Clustering Engine

The "De Gustibus" experiment adopted K-means as the central algorithm in choosing items to present to users. K-means was hypothesized as a methodology to best cluster food items into self-similar groups, thereby creating groups dissimilar to one another. The initial item is selected from a random cluster, and clusters rotated one-by-one, with a single item presented to the user selected from each cluster.

The item in each cluster is selected based on maximum "dissimilarity" from what was already presented to the user. This was done by calculating the cosine similarity (find the most similar items from a list to what the user was already seen), and then negating the result. Thus, the most "negatively similar" item would be chosen among the cluster for presentation.



In the first iteration (red), a random cluster is selected (1), and a random item selected for presentation (X). The subsequent cluster is selected – cluster 2 – with the least similar item in cluster 2 (O) selected for presentation, given the similarity to what was chosen in cluster 1. The third cluster follows suit, with the item (@) selected based on the dissimilarity between the two items in the previous cluster. These three (X, O, @) items are then presented to the user. The process is subsequently repeated as the user requests more suggestions, and sees more items. The underlying assumption is that each request for more items is an implicit statement of non-relevance for the current items, unless noted otherwise (by clicking “similar” or “saving” the item.)

K-means was seeded randomly, using sqrt(N) menu items. This roughly translated to ~ 80 clusters of varying composition given the 10025 and 10027 zip-code restaurant set. Examination of the clusters showed many sparsely populated groupings, with a few dominated by a large number of items. Sparsely populated items tended toward unique restaurants and dishes consisting of language-specific terms; African (for example, “Tebsi”) and Thai (for example, “Pad Kee”) were noticeable standout clusters. Largely populated clusters contained common American food items – clearly honing in on “burger” and “sandwich” keywords. Chinese food items were the last noticeable distinction, although many clusters did contain multiple elements.

Functionality

Similar: users are given the option to see "similar" dishes. Dishes are consequently ranked in order of highest cosine similarity for presentation to the user. Items are added to a list of what the user has previously seen, so that subsequent cluster rotations are informed.

Save: The save functionality is for the user's benefit to track interesting food items, and does not influence similarity calculations.

Search: A basic keyword search tool provides a ranked list of matches in the term-document matrix. Terms are stemmed and ranked according to cosine similarity. This allows the user to search for specific items, cuisines, or restaurants. Matches are presented in rank order.

Presentation Layer

The presentation layer is dynamically generated in JavaScript based on a user profile consisting of items seen or items requested to be similar, with the appropriate Php scripts called to determine the results. The data for results are culled from the database, and elements rendered (geo-location coordinates to the Google map) and item data into the appropriate item "pane."

Evaluation

Evaluation of the system proved difficult as user preferences are difficult to gauge. A brief user survey was presented to testers to determine the success of "De Gustibus". The survey consisted of watching users interact with the system, and counting the number of times "saved", or "similar" requests were made, over all suggestion requests. This was compared with a session of randomly generated items – made unknown to the user. The underlying hypothesis was that a higher ratio of "saved" or "similar" clicks over all item requests, was indicative that requests were of use / interest to a user, and were also novel / diverse enough (had not yet been seen), that they would consider them as options for lunch. The session concluded when the user felt like they were done. Earlier sessions typically were longer as a brief explanation of the user interface was needed. General feedback was also elicited to gauge user sentiment.

	Tester	**duration**	**# saves**	**#similar**	**#requests**	**Ratio**
Degustibus	Tester 1	8:00	8	8	23	0.695652
Random	Tester 1	6:00	5	12	30	0.566667
Random	Tester 2	9:30	13	5	21	0.857143
Degustibus	Tester 2	7:00	9	9	26	0.692308
Random	Tester 3	7:00	2	1	15	0.2
Degustibus	Tester 3	3:00	3	6	12	0.75

Of the users, 2/3 thought “De Gustibus” gave helpful results, while 1/3 thought the Random session was more helpful. The level of “diversity” was thought to be equal by both, while all of them thought it was generally a useful tool.

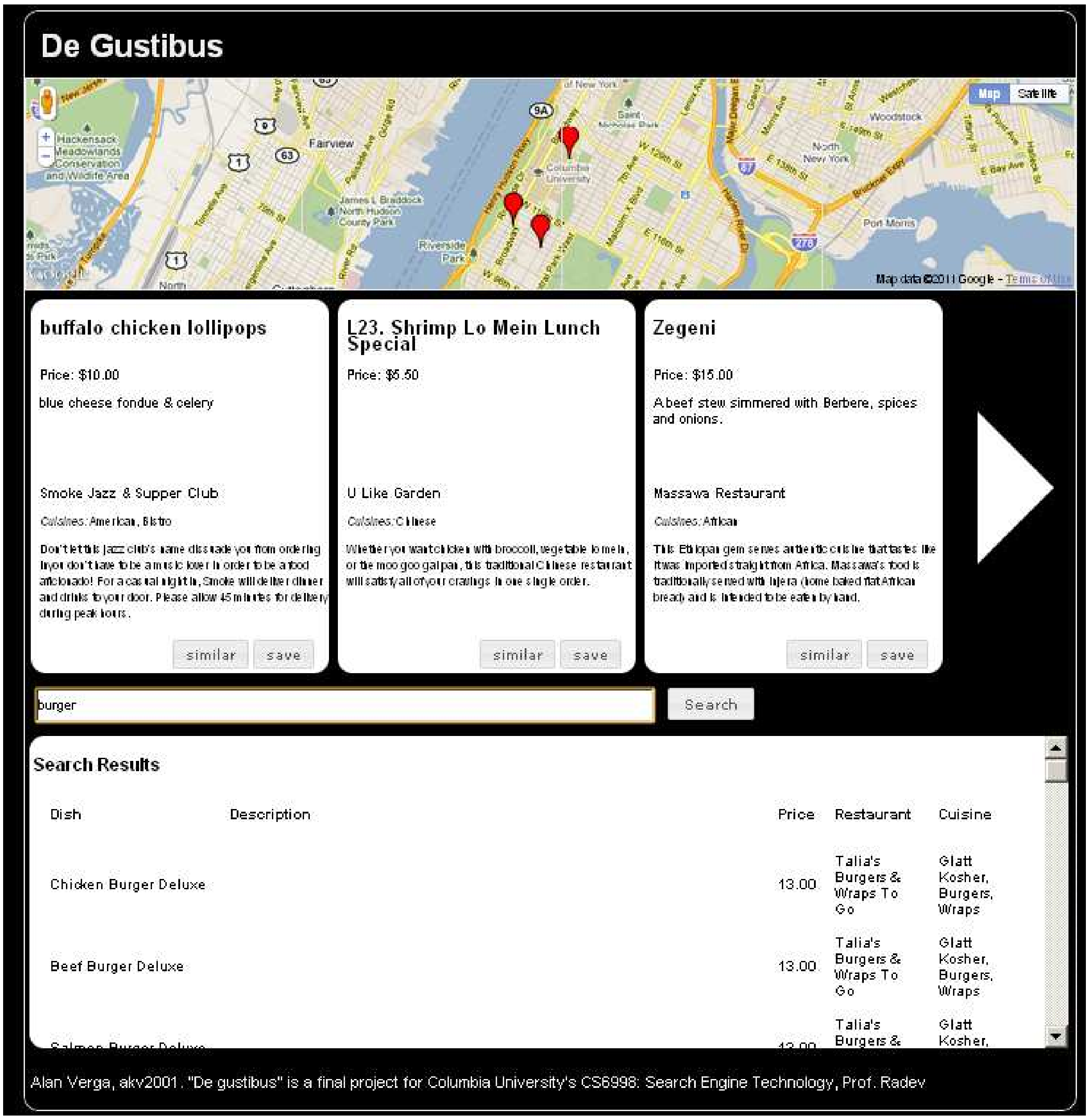
While admittedly not a scientific evaluation, it provides some color for evaluation. The data suggests that K-means provides better suggestions than random, though the margins are not significant between users, and can be shown to vary wildly. Thus, a larger sample size is needed for any serious validation, but some users may find it a helpful tool from time to time.

Improvements

There is any number of improvements that can be made to the system. Feedback ran from general user interface confusion, to uncertainty as to what to do with the results (“so do I go and eat them?”) From the brief testing, it is clear that a new methodology would better serve the notion of browsing for food items. A level of personalization was clearly desired, and could be served by introducing a brief “taste” survey. It is clear that a K-means is not a strong implementation of a content-based recommendation system, especially when variable taste preferences are involved. Particularly frustration was the lack of any clear metric to evaluate a “successful” recommendation, as a tester even preferred seeing more “random” recommendations rather than finding similar items. However, all agreed that it was an interesting tool with potential, and the ability to incorporate many classical IR techniques into a full system was a worthwhile experience.

Demo output

A working system is accessible at http://www.degustib.us



De Gustibus

buffalo chicken lollipops

Price: $10.00

blue cheese fondue & celery

Smoke Jazz & Supper Club

Cuisines: American, Bistro

Don't let this jazz club's name dissuade you from ordering inyou don't have to be a music lover in order to be a food aficionado! For a casual night in, Smoke will deliver dinner and drinks to your door. Please allow 45 minutes for delivery during peak hours.

similar save

L23. Shrimp Lo Mein Lunch Special

Price: $5.50

U Like Garden

Cuisines: Chinese

Whether you want chicken with broccoli, vegetable lo mein, or the moo goo gai pan, this traditional Chinese restaurant will satisfy all of your cravings in one single order.

similar save

Zegeni

Price: $15.00

A beef stew simmered with Berbere, spices and onions.

Massawa Restaurant

Cuisines: African

This Ethiopian gem serves authentic cuisine that tastes like it was imported straight from Africa. Massawa's food is traditionally served with injera (home baked flat African bread) and is intended to be eaten by hand.

similar save

burger

Search

Search Results

Dish	Description	Price	Restaurant	Cuisine
Chicken Burger Deluxe		13.00	Talia's Burgers & Wraps To Go	Glatt Kosher, Burgers, Wraps
Beef Burger Deluxe		13.00	Talia's Burgers & Wraps To Go	Glatt Kosher, Burgers, Wraps
Salmon Burger Deluxe		13.00	Talia's Burgers &	Glatt Kosher,

Alan Verga, akv2001. "De gustibus" is a final project for Columbia University's CS6998: Search Engine Technology, Prof. Radev

A screenshot of the website shows three items, and their locations on a Google map. Item names, prices, and brief descriptions are available, in addition to restaurant information (cuisine, description.) The large triangle is a button to get the next set of results.

Appendix File listing

File Listing
./degustibus/clustering/engine – contains c++ "cluster engine" source and executables.

ClusterEngine_server: server runtime needed for website to run
ClusterEngine_server.cpp: source
cluster.cpp: cluster class source
cluster.h: cluster header
cluster_index.*: test file for local verification. Not directly needed for website.
Cluster.thrift: IDL thrift specification

./doc: documentation
./gen-cpp: thrift generated server source for c++ code
./gen-php: php generated client source

./degustibus/crawl

README.txt: contains descriptions and order of crawling scripts
batch_import_menus.pl: imports menus to db
batch_term_doc_matrix2.pl: builds term doc matrix from multiple files to ./index
build_menu_queue.pl: builds list of menus to be downloaded from delivery.com
build_term_doc_matrix2.pl builds term doc matrix from single file
clean.pl: cleans database of extraneous items
get_menus.pl: downloads menus from delivery.com
get_restaurants.pl: downloads restaurants from delivery.com
import_menus_db.pl: imports menus to database
import_restaurants_db.pl : imports restaurants to database
zipcodes.xml: listing of zipcodes from which crawling can start from

./doc: documentation
./index: term-doc index output
./lib: packages for above scripts (see documentation)
./menu_queue: directory housing zipcode denoted lists of emnus
./menus: stores downloaded menus
./restaurants: stores downloaded restaurant files

./degustibus/www (requires active web server)

index.html: HTML file
PhpClient.php: server side php script to get items
Search.php: server side php script to search for terms
Stem.pl: basic stemmer to stem search terms

./css: contains style sheet for website
./gen-php: contains thrift libraries for PhpClient.php
./js: contains javascript for dynamic generation of site content
./lib: contains supporting perl packages for stemming
./thrift: contains libraries need for thrift

Compilation / Run-time

Cluster Engine: run ‘make’ to compile, run ./ClusterEngine_server binary and wait until “server active” is returned.

Assuming your web server is properly setup, simply login to the website, and start browsing.