**CS6998: Search Engine Technology:   
Contextual Keyword Extraction Engine**

**By Anjishnu Kumar**

**UNI: ak3534**

**Project Report and API Documentation**

**Contextual Keyword Extraction Engine**

**Abstract:**

Keyword extraction is an extremely interesting topic in Information Retrieval- keywords are widely acknowledged to be extremely important in the field of text retrieval, and particularly while developing large scale modern search engines that limit the size of the inverted index used by the system.

Keyword extraction can be considered as closely related to the summarization problem, however there are some major differences: while summarization attempts to identify informational content of a segment of text and then return a portion of grammatical text, Keyword Extraction simply attempts to find short segments and tokens that contain the most important information, however they need not be grammatical.

However Keyword extraction has been a particularly challenging problem for a variety of reasons, including the lack of substantially sized standardized corpora.

Contextual keyword extraction is an extension to keyword extraction that finds keywords that are pertinent to a particular topic to enhance context specificity.

In this project we propose to build a system using modern NLP techniques such as Part of Speech Tagging, Brown Clustering and Rapid Automatic Keywords Extraction (RAKE) to use a small initial seed of keywords to generate more candidate keywords in a semi-supervised manner and expose the system as a JSON based web service.

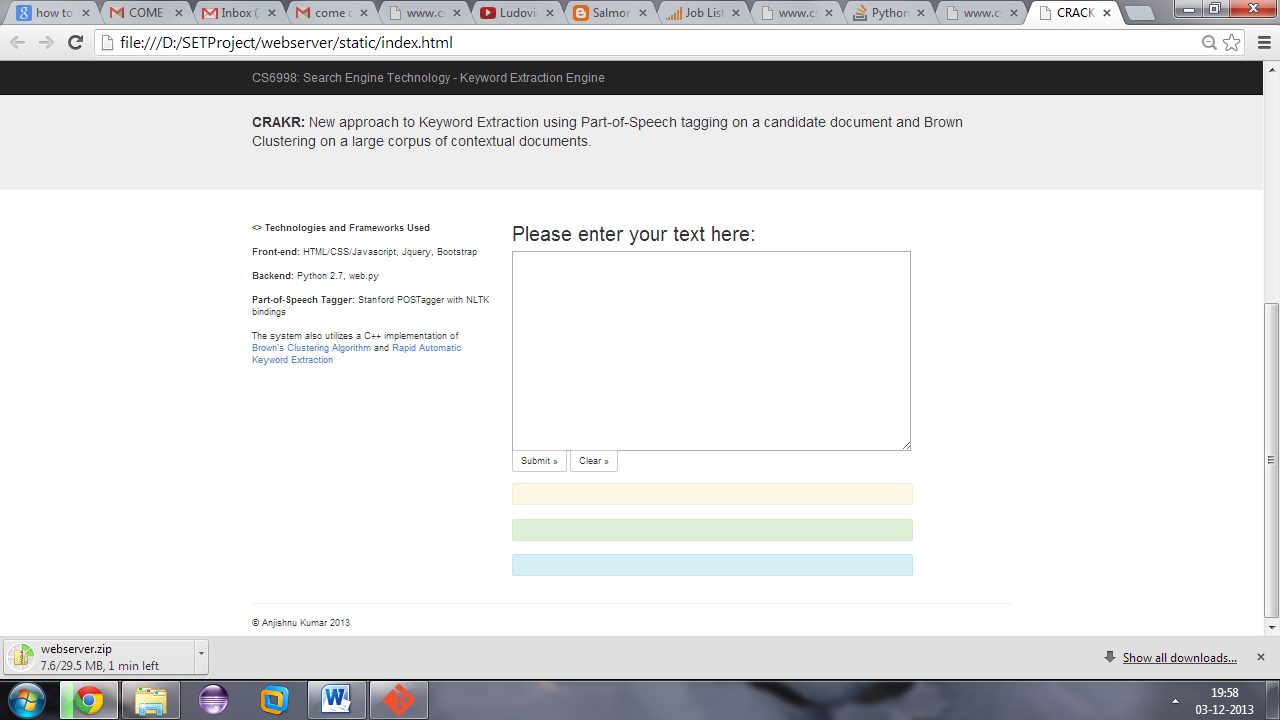
As a test case, the context we will be testing this engine in- is **online Technical Job Postings**. Using a corpus of job listings mined from the internet, hierarchical clusters have been formed using Brown Clustering.

Using an initial seed of job skill keywords, the Brown Clustering data will be used to find candidate keywords from the large corpus. This simulates real life behaviour in which small tagged datasets and much larger untagged datasets are available.

**Motivation:**

In today’s world, keyword extraction can become an incredibly powerful tool. The usage of keywords in social media has risen many-fold due to the use of the #hashtag, and the following years there will be a glut of data mapped to keywords being generated by the internet.

It is also important to note the rise of contextual search engines, such as highly customized job search engines, which are trying to facilitate more fine-grained matching of candidates and jobs that match their skillsets.

Accurate keyword extraction has many different uses, particularly if it can be applied as a means to identify contextually relevant information.

**<> Technologies and Frameworks Used  
  
Front-end:** HTML/CSS/Javascript, Jquery, Bootstrap   
  
**Backend:** Python 2.7, web.py   
  
**Part-of-Speech Tagger:** Stanford POSTagger with NLTK bindings   
  
The system also utilizes a C++ implementation of [Brown's Clustering Algorithm](http://www.cs.columbia.edu/~cs4705/lectures/brown.pdf) and [Rapid Automatic Keyword Extraction](http://sujitpal.blogspot.com/2013/03/implementing-rake-algorithm-with-nltk.html)

**RAKE: This system uses a customized implementation of RAKE built on the design by github user Aneesha. (**<https://github.com/aneesha/RAKE>)

**Brown Clustering: This system uses Percy Liang’s Brown Clustering Implementation**

**Data:**

Datasets for skills and job listings were provided by [www.Collegefeed.com](http://www.Collegefeed.com) and are confidential.

**Terminology Used:**

**NAIVE:** Naive selection of keywords that were already present in the seed corpus.

**RAKE:** Keywords extracted via the Rapid Automatic Keyword Extraction Algorithm

**CRAKR:** New approach to Keyword Extraction using Part-of-Speech tagging on a candidate document and Brown Clustering on a large corpus of contextual documents.

**Key Software Modules:  
  
serve.py – Main server file  
textprocess.py – contains code for textprocessing  
postagger.py – interface with the Stanford POS Tagger  
rake.py – A customized python implementation of the RAKE algorithm  
candygen.py – Contains the implementations of the Naïve keyword extraction and CRAKR algorithms.  
index.html/index\_helpers.js – Contains code for the Front end and GUI**

**Theoretical Background:**

**Brown Clustering:**

Brown clustering is a word clustering algorithm developed in 1992.

The algorithm proceeds as follows

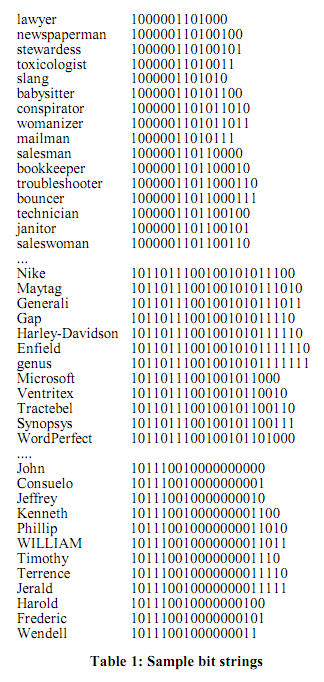
* In the beginning each word is considered its own, distinct, cluster
* Algorithm- Repeat:   
  Until only one cluster remains:
  + Merge two clusters based on some measure of quality (such, as smallest decrease in loss of mutual information on merging, or smallest reduction in the likelihood probability of the text based on some language model)
* A complete binary tree is thus obtained as the clusters are continuously merged.
* The tree is traversed. Each word is assigned a bit-string based on the clusters it was in and when they were merged. (For example, 0 every time we go left in the try and 1 every time we go right, or vice vera)
* The bitstring values can be interpreted as follows: The left-most bits are the most significant, and can be used as a similarity measure

**Advantages of Brown Clustering:**

The user can specify the number of clusters the algorithm attempts to find- thus allowing for a great deal of flexibility and fine tuning when dealing with increasingly large quantities of data. Takes advantage of increasingly accurate language models to improve over times.

**Disadvantages of Brown Clustering:**

Unable to model syntactic similarity, needs to be used in conjunction with a Part of Speech Tagger in order to ensure reasonable accuracy. Speed is low for large data sets.



This is a sample output after the Brown Clustering algorithm is run on a sample of text.

The numbers next to the words are known as bit-strings.

**Evaluation:**

**Evaluation of the system was done by comparing it to two other techniques of keyword extraction.**

1. **Naïve text selection from keywords in training set.**
2. **Comparison with RAKE- the non-context specific unsupervised keyword extraction algorithm.**

**Here is a sample.**

**Job Listing:**

*Our Lead SDET role requires a strong background in writing functional, integration and performance test automation for distributed service architectures. thePlatform’s media publishing system SDK requires RESTful services working in concert and requires in-depth technical understanding to test them effectively. Lead SDETs assist testers across several functional components concurrently, mentor mid and junior-level engineers, design and build test automation, and provide constructive reviews of test plans, test code and test process in an agile environment sharply focused on delivering customer value.*

*This position offers a remarkable opportunity to utilize your testing, design and coding skills on a broad range of technologies, and processes in a growing company with one of the healthiest development organizations you'll find anywhere.*

*Responsibilities:*

*Provide technical leadership for the design and implementation of test automation.*

*Attend product specification reviews and assist teams writing and reviewing test plans.*

*Write performance test plans and automation for integrated services and clusters.*

*Compose user stories for test with realistic time/effort estimates.*

*Develop and support test modules, and evangelize their use.*

*Keep abreast of industry developments and evolve thePlatform’s best practices.*

*Skills & Requirements*

*Requirements:*

*8+ years writing test plans and building automation with Java and TestNG or JUnit.*

*2+ years experience leading a team in an agile environment.*

*Expert test code troubleshooting, debugging and refactoring.*

*Proven experience with SOA, design patterns and web services technologies.*

*Recognize and test service, database and environment inter-dependencies.*

*Proficient with MySQL or Oracle, and sharing between datacenter topologies.*

*On point verbal and written communication skills.*

*Confidence and an easy-going personality.*

*Bachelor of Computer Science or a related discipline.*

**Results:**

**NAIVE: debugging, communication, testing, computer, design, troubleshooting, mysql, web, java, publishing, service, media, support, writing, environment, computer science, web services, test, junit, development, product, automation, leadership, services, a, database, science, test automation, agile, oracle, communication skills,**

**RAKE: requires, test, automation, technologies, theplatform’, integrated services, test plans, writing functional, agile environment, test code, assist teams writing, provide constructive reviews, web services technologies, reviewing test plans, provide technical leadership,**

**CRAKR: Lead SDET, background, functional, integration, performance test automation, service architectures, media, system SDK, RESTful services, concert, technical, Lead, testers, several functional components, engineers, design, test automation, constructive reviews, test plans, test code, test process, agile environment, customer value, remarkable, testing, design, coding skills, technologies, processes, company, development organizations, Responsibilities, technical leadership, design, implementation, test automation, product specification reviews, teams, test plans, performance test plans, automation, services, user stories, test, realistic, effort estimates, test modules, use, industry developments, best practices, Skills, Requirements Requirements, test plans, building automation, Java, JUnit, team, agile environment, test code troubleshooting, debugging, SOA, design patterns, web services technologies, service, database, environment, dependencies, MySQL , Oracle, sharing, datacenter topologies, point verbal, written communication skills, Confidence, easy, personality, Computer Science,**

**Notes:**

**Naïve extraction results in very few keywords being generated. RAKE also doesn’t manage to capture a large number of the context specific keywords (MySQL, Oracle, Java, JUnit) required in this case.**

**CRAKR outperforms both the Naïve and the RAKE algorithm on recall, but still has issues with noise and precision.**

**These can be improved on in further work by utilizing active and semi-supervised leaning approaches.**

**Pipeline:**

**Training:**

* **Collected a corpus of job listings –**
* **Appended all job listings to large file big.txt –**
* **preprocessed the large file (punctuation removal, adding of padding whitespaces around stops and commas, lowercasing etc)**
* **Used this as input to Percy Liang’s clustering algorithm.**
* **Use the output to generate clusters to explode the output from naïve selection.**

**System:**

* **Data entered into GUI > sent to the server via http POST request in JSON format.**
* **Server received the JSON request**
* **JSON is parsed into text**
* **RAKE Algorithm runs on text**
* **Naïve Extraction is run on the text- a small number of valid keywords are identified**
* **POS Tagging is performed on the algorithm.**
* **The keywords obtained in the Naïve Extraction are exploded using brown clusters.**
* **The possible keywords are reduced using POS tags – only tags starting with J and N (Adjectives and Noun-phrases) are retained. These tags were found to contain the maximum amount of information.**
* **The outputs are sent back to the**

**Potential Improvements:**

**The algorithm developed in this project has an impressive performance on certain data sets, especially in terms of recall. There are several possible improvements to the implementation.**

* **Encoding issues. The algorithm only works for standard utf-8 encoding, and throws a server error in case of exotic encoding. Later implementations should be extended to new encoding.**
* **Noisy data. Precision can be improved by reducing noise from the datasets via more sophisticated preprocessing.**
* **Speed: the algorithms described in this work can be sped up by recoding them in a faster language such as C or Java.**
* **Semi-supervised/Active learning: The current algorithm depends on manual corpus expansion being performed by the server side engineer- manually adding new text to the big.txt file and rerunning the process. This can easily be automated, but would lead to rapidly degrading performance over time as the amount of data in the corpus increases. The sweet spot has not yet been discovered.**
* **Active learning using online user input could be a way to disentangle the issues of noisy and large data by instead repurposing online web users to provide less, but higher quality data. This approach has not been explored in this work yet.**

**API Documentation:**

**Dependencies:**

* **Python 2.7**
  + **NLTK (**<http://nltk.org/install.html>)
  + **Web.py (**<http://webpy.org/install>)
* **Java Runtime Environment (Required for the Stanford POS-Tagger module)**
* **Any modern C compiler (Optional: Needed for the Brown Clustering training- but the trained file is already provided)**

**Launching the server:**

**Launching the server has been converted into a simple process.**

**$ python serve.py**

Running this script should activate the server and cause it to run on the localhost on port 8080.

You can now use the GUI interface on localhost:8080 to enter the sample text.

**Note:** RAKE is a context-free unsupervised Keyword Extraction Algorithm, and can be used effectively on any moderately sized document to generate keywords. The modified version of RAKE that is used in this system limits the maximum size of the output, so the RAKE algorithm will become less efficient with larger documents unless the limit parameter in rake.py in modified.

The other two keyword extraction approaches depend on a trained corpus to generate alternatives, therefore they are expected to result in poor performance if used with documents that are highly dissimilar to the case sample, for the best performance, it is recommended that the text used be relevant to the training corpus, otherwise the performance will be very poor.

Some example websites for technical recruiting:

<http://careers.stackoverflow.com/jobs/>

<http://www.collegefeed.com/cf/jobs?company=ebay&keyword=software+engineer&location=San+Jose%2C+CA>

**References:**

* **Rapid Automatic Keyword Extraction**  
  Rose, S., Engel, D., Cramer, N., & Cowley, W. (2010). Automatic Keyword Extraction from Individual Documents. In M. W. Berry & J. Kogan (Eds.), Text Mining: Theory and Applications: John Wiley & Sons.
* **Implementation of RAKE in Python:** <https://github.com/aneesha/RAKE>
* **Brown Clustering - Brown et al, 1992**
* **Brown Clustering: C++ Implementation, from Percy Liang’s Master’s Thesis, used as is.**

<https://github.com/percyliang/brown-cluster>