Final Project Report
COEN 272 Web Search and Information Retrieval

Search Engine for Mobiles Devices

Instructor: Ben Steichen

Team:
Ziyi Yang
Shilpita Roy
Mansi Iyengar
Raymond Christopher

Contents

Abstract

A search engine is the practical application of information retrieval techniques to large-scale text collections. We wanted a central location for users to look up mobile devices and their specifications such as camera quality, display size, and price easily. A text corpus is created by crawling all the sites with information of mobile devices. Search engine was implemented, using technology such as Ajax-Solr for the front end and Apache Solr for the search engine platform.
Facets and classifications based on key features of cellphone was done during the crawling. User study and evaluation of the basic search engine and search engine with facet classifications were performed and logging information was for each user is stored. SUS and NASA TLX questionnaire is used to gather user reviews.

Introduction

Searching and communication are the most common use of computers and the web. For an elegant web search and information retrieval, the search engine should fetch the most relevant, fresh and effective information. The building blocks of a Search engine architecture are Text acquisition, Text processing and Indexing.

Users often buy their mobile devices based on their popularity or hype without actually knowing what the actual tech specifications are for that device. Apple iPhones, ipads and Samsung Galaxies are a common sight in our everyday lives, but there are countless other brands on the market such as Sony, Microsoft, and Motorola. We chose our search engine domain as mobile devices so that users can understand the specifications of their devices and possibly explore other options when it's time to upgrade instead of sticking with the same brand of their previous devices. Our user base are people that are looking to upgrade their devices or are just curious about what of kind of mobiles devices there are on the market. Our domain also targets users that want to compare the specifications of different devices to see which is better or fit their needs.

We have used the Apache Solr as the framework to build our search engine. The widgets and core code was developed in JavaScript. Json was used to record the user logging information. Solr is highly reliable, scalable and fault tolerant, providing distributed indexing, replication and load-balanced querying, automated failover and recovery, centralized configuration and more.
The search engine for devices was built in the following parts:

1. Crawler – to crawl and download the websites with information regarding the devices
2. Noise removal- to remove the noise (ads, navigation menu etc.) to avoid error in retrieval
3. Indexing/ Stemming/Facets – the crawled pages are indexed and stemmed for facets and classification
4. Logging – User session details, queries, clicks etc. are recorded.

Implementation

Crawler:

To crawl the web, we identified the websites that contain the latest and all inclusive details of the devices. We tried to find out the go-to sites that contains specification, user reviews and prices for all mobile devices. Gsmarena.com, phonearena.com, and phonescoop.com are websites that contain information about mobile devices such as reviews, news, and specifications. The specifications page of these websites contain information on the phone such as the display size, camera quality, system memory, and price. We only care

about the specifications page of these websites so our crawler only saved those pages. We decided to label the documents with facet information during crawl time.

The web crawler is built on Java using Jsoup jar. It takes a seed from the csv file as input. The file contains the site url, limit of links to crawl and domain. The downloaded pages and documents are stored as repository for the search engine. The crawler follows the politeness policy and respects the ROBOT.txt files of the sites it is crawling. We crawled total of 3 sites with 5,512 pages.

Seed: www.phonearena.com/phones/Apple, 5000, www.phonearena.com/phones/Apple

Noise Removal:

The pages that were crawled being commercial sites contained a lot of noise data. These noise data were navigation information, related stories, user comments, advertisements and breadcrumbs. Since the indexing and then the page retrieval for search was based on the entire document, these noise data added surplus word counts and resulted in inaccurate information retrieval. It became necessary to perform text processing to strip the pages of the noise.

Since the indexing and facet building was based on the meta data in the html pages. The text processing algorithm was designed to keep the head of each html intact. The id class of the noise data was recognized and removed from the html pages.

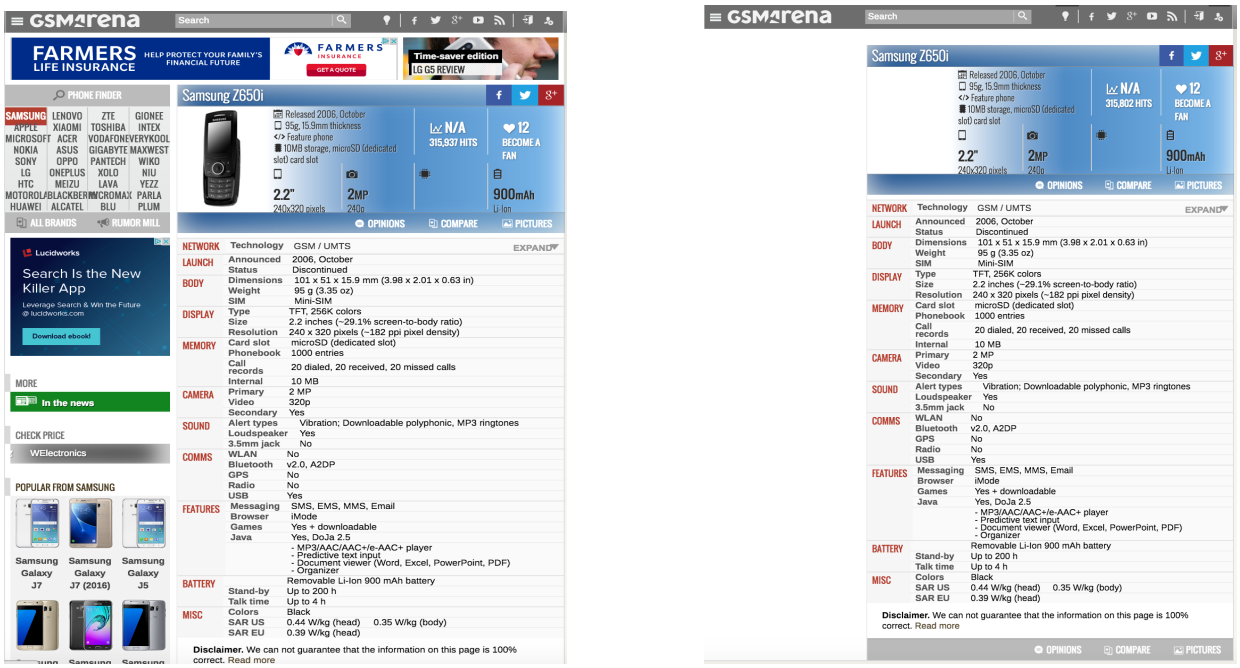

Figure 1. The Noise removed HTML for gsmarena.com

Indexing and stemming:

When a document is indexed, its individual fields are subject to the analyzing and tokenizing filters that can transform and normalize the data in the fields. For example — removing blank spaces, removing html code, stemming, removing a particular character and replacing it with another. At indexing time as well as at query time you may need to do some of the above or similar operations.

To build our search engine, we have extracted the metadata of the crawled HTML pages and used them for building our index. Some of the pages did not contain the details of the URL. For those pages we have

added the url tag in the HTML metadata through a java module inside the crawler. Thus, crawler ensures that all the downloaded webpages have the same metadata tags. This makes building the index easier.

For better search results and relevant information retrieval the stemming and stopping for our engine is done at the query and page retrieval time. Stemming, also called conflation, is a component of text processing that captures the relationships between different variations of a word.

For our search engine, the stemming is partially based on the Porter transformation where the plurals are considered for stemming and is less aggressive than the actual Porter Transformation. This was done by using the 'EnglishMinimalStemFilterFactory' of the Solr library. We used Solr's text_en attribute to index the content of the documents. This attribute uses Solr's default English stop word list and uses the default Porter stemmer. However, we modified it so that no stopping or stemming is done at index time and leave that during query time to increase the flexibility of the search engine. We found that the Porter stemmer was too aggressive and changed it to Solr's English Minimal stemmer. The English Minimal stemmer only stems plurals which suited our search engine. For query expansion, there's only a few words that we manually specified to expand. "GB, gigabyte, gigabytes", "MB, megabyte, megabytes", and "MP, megapixel, megapixels" are the three expansion categories. We also specified Solr to look for the facet fields in the document metadata and index them into fields 'brands', 'cameras', 'display', 'memory', and 'price'.

```
<dynamicField name="*_txt_en" type="text_en"  indexed="true"  stored="true"/>
<fieldType name="text_en" class="solr.TextField" positionIncrementGap="100">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <!-- in this example, we will only use synonyms at query time
    <filter class="solr.SynonymFilterFactory" synonyms="index_synonyms.txt" ignoreCase="true" expand="false"/>
    -->
    <!-- Case insensitive stop word removal.
    -->
    <!--     <filter class="solr.StopFilterFactory"
            ignoreCase="true"
            words="lang/stopwords_en.txt"
        />
    -->
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.EnglishPossessiveFilterFactory"/>
    <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>
    <!-- Optionally you may want to use this less aggressive stemmer instead of PorterStemFilterFactory:
    <filter class="solr.EnglishMinimalStemFilterFactory"/>
    -->
    <!--
    <filter class="solr.EnglishMinimalStemFilterFactory"/>
    -->
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
    <filter class="solr.StopFilterFactory"
            ignoreCase="true"
            words="lang/stopwords_en.txt"
    />
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.EnglishPossessiveFilterFactory"/>
    <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>
    <!-- Optionally you may want to use this less aggressive stemmer instead of PorterStemFilterFactory:
    <filter class="solr.EnglishMinimalStemFilterFactory"/>
    -->
    <filter class="solr.EnglishMinimalStemFilterFactory"/>
  </analyzer>
</fieldType>
```

Figure 2: The SolrConfig.xml for text processin

Facet/Classification:

We decided to label the documents with facet information during crawl time. We parsed each specification page to get the information for our facets with the help of Jsoup. We got the camera quality in megapixels, the display size in inches, the price in dollars, the ram in gigabytes, and the brand name. These facets information were then added as metadata in the html documents, once again with the help of Jsoup. Saving these information as metadata in the documents' header makes it easy later to generate facet fields when we use Apache Solr to index the documents. Also when using Solr to index, we only indexed the terms that are inside the main content of the specifications page. We didn't want to index any terms that were in advertisements and we used the help of our Content Processor from Project 1 to remove all unwanted noise. Ultimately, our crawler got specification information of all types of devices from brands such as Apple, Samsung, LG, Sony, Motorola, HTC, Microsoft, and Blackberry. There were 5,512 total documents crawled and later indexed.

Page ranking:
PageRank is a way of measuring the importance of website pages. PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. [3] We did not make any changes to Solr's default retrieval model. By default, Solr uses a combination of the Boolean model and the Vector Space mode. Documents that pass the Boolean model are then scored by the Vector Space model. The vectors are weighted by using Tf-idf values. Tf is the term frequency, it is higher when the term is more frequent in a document. Idf stands for inverse document frequency, it is higher when the term is rarer in the whole collection. The Vector Space model then computes the score of a document by calculating the Cosine Similarity between the document and the query.

User Logging:
The search has been designed on the system itself. In this we make use of the session Storage and ensure that the user data per task is recorded in it.

Login.html: In this page the user name is accepted and his data is stored in session Storage of the browser and his session start time is recorded when he selects a certain button (basic/ advanced search engine). He is then directed to the search engines. Below is the screenshot of the data stored after the session has begun. For each user session details like user name, login time, logout time, Query, Facets clicked, Total clicks, satisfied links, relevant links etc.
JSON recorded:


```
{
  "UserName": "justin",
  "UserSessionStartTime": "2016-05-30T22:41:24.981Z",
  "SearchEngineSystem": "advanced",
  "Query1Facet": [
    {
      "facet": "brands_Samsung"
    },
    {
      "facet": "memory_8"
    },
    {
      "facet": "display_3.2"
    }
  ],
  "TotalQuery1Facet": 3,
  "TotalNoOfFacetsInSession": 3,
  "Query1": "samsung 3.2 inch screen 8gb ",
  "Query1timestamp1": "2016-05-30T22:41:46.567Z",
  "TotalQuery1ResultsClicked": 0,
  "Query1MarkedAsRelevant": [

  ],
  "Query1MarkedAsSatisfied": [

  ],
  "TotalQuery1MarkedAsRelevant": 0,
  "TotalQuery1MarkedAsSatisfied": 0,
  "TotalNoOfQueries": 1,
  "TotalNoOfClicksInSession": 0,
  "TotalNoOfSatisfiedInSession": 0,
  "TotalNoOfRelevantInSession": 0
}
```


Figure 3: JSON of the user login per session

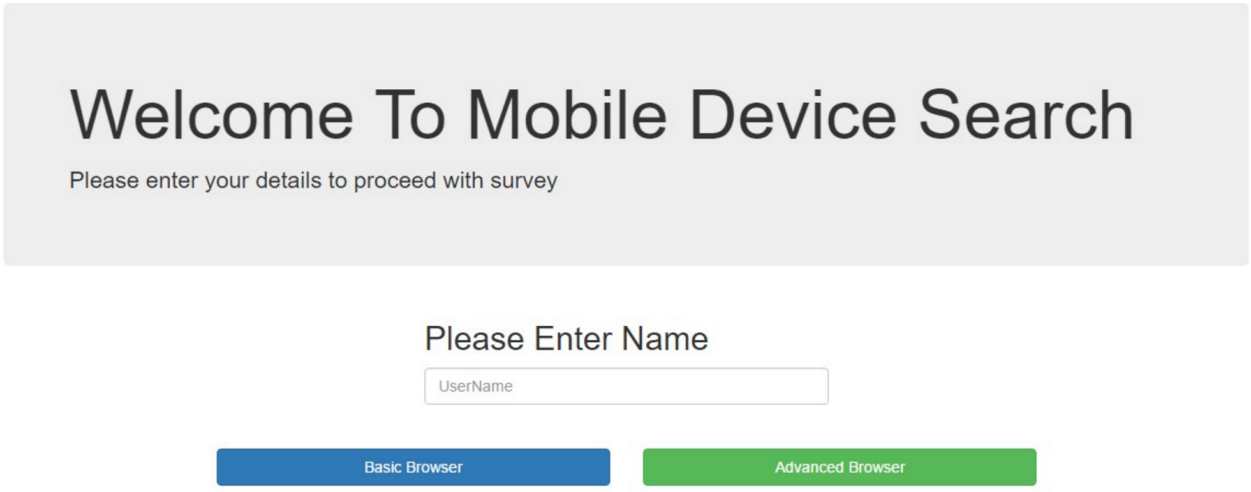

Figure 4: The User logging Page

Search Engine versions and components:
We used an open-source Javascript framework called Ajax-Solr as the front end of our search engine. Ajax-Solr provides us with a dynamic UI that uses Ajax calls to Apache Solr to retrieve relevant documents to the query. Ajax-Solr also has widgets that gave us functionality like facet checkboxes that once again do Ajax calls to Solr for faceted search. Ajax-Solr made it simple for us to create a UI and we did not need to be Javascript experts to accomplish it. It also communicated with Solr with ease and its widgets were easily customizable to fit our needs, like displaying our facets with MP at the end for cameras. The Ajax-Solr widgets we used to be the Text Widget as the search bar, Result Widget to display the search results, Multi Select Widget as the checkboxes for facets, and Current Search Widget to display the current query and current facets selected. Each search result contains the document title, the live URL, and the document description as the **snippet**. The title, URL, and description was retrieved from the document's metadata during indexing. The schema and Solr configuration files were updated to point to the core containing the search engine codes.

a) **Basic Search –**
The basic search engine has a query search box when user can enter his query. The search engine performs the parsing(tokenization/Stemming/stopping) on the query entered on the search bar. Retrieves the documents on the screen.
For eg: Entered query – 'Apple 6 Gold 8GB Memory'

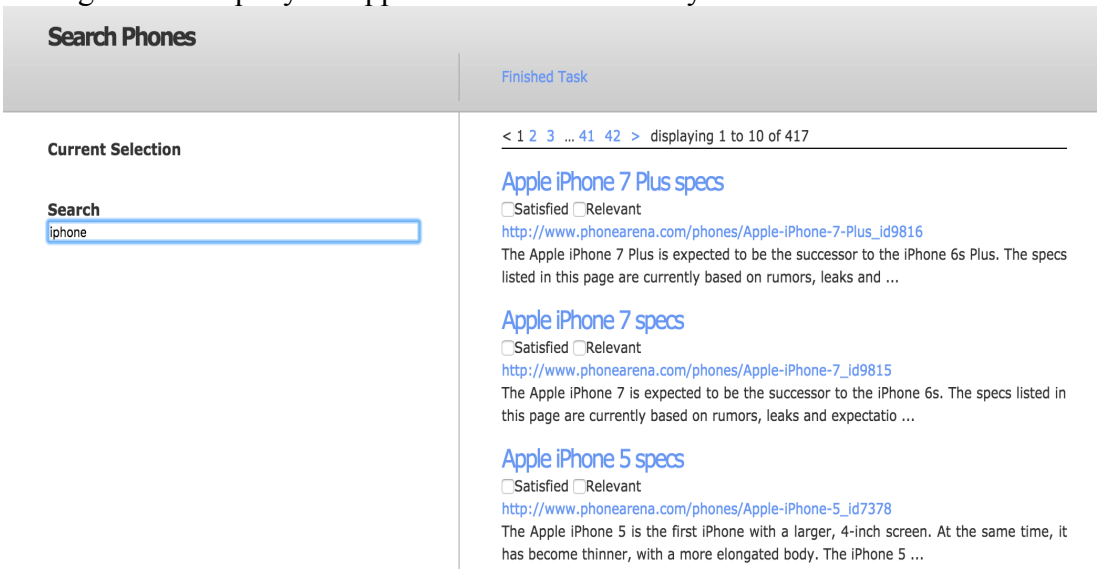

Figure 5: Screenshot of a Basic Search engine

b) **Advanced faceted search-**
The advanced search has the query bar as well as the facets based on Brands, Display size, Memory, Camera, Price. Thus user can enter a word text in the search bar and narrow down the search using the facets provided.
For eg: Entered Query – Apple 6 Gold Facets – Camera: 2MP

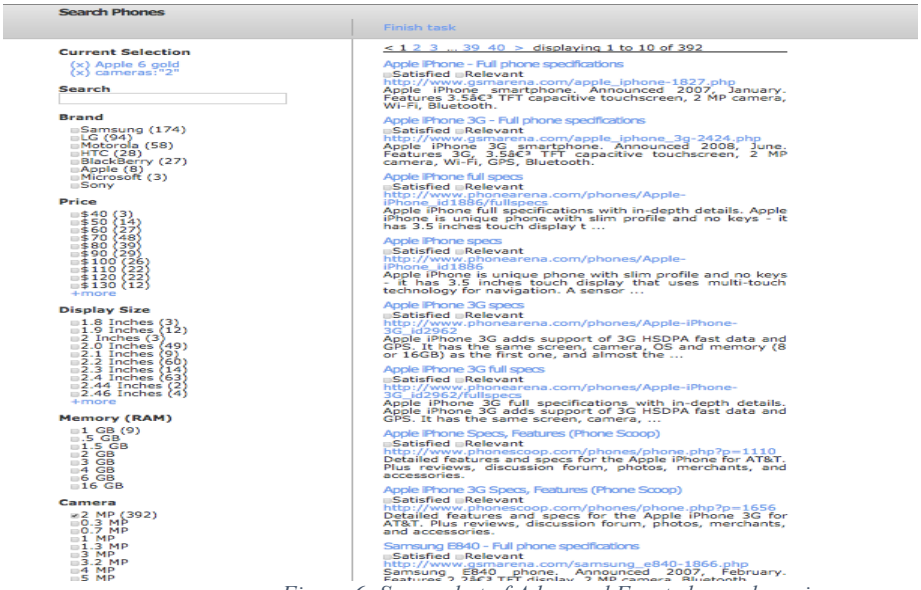

Figure 6: Screenshot of Advanced Faceted search engine

User Study Methodology and Results:

A user survey was performed with 5 users from varied age group (18-35). We went to people in the library and asked them to try our search engines to find a specific mobile device. We gave them some instructions and a particular task. User had to run perform the task on both basic and the advanced search engines and mark the urls that they found relevant or satisfactory to the task they performed. The user could mark the urls using the checkboxes below them. The checked url and their positions are recorded in the logging detail JSON.

Figure 7: The relevance of the page retrieved

User instructions –

- Each task should be done in separate user login session
- User should mark pages as relevant and satisfied
- For advanced faceted search user must use the query bar and facets together.

Tasks details –

Tasks were designed to search for specific mobile devices using various queries. Some tasks do not find any results. There were ten tasks listed and each user was given two tasks. User performed one task each for Basic and Advanced Search Engines.
Some tasks are listed below:

- Find Microsoft products with 8GM memory.
- Find Samsung phones with 3.2 Inches 8GB ram.
- Find 12MP, 3.5 inches' phones.

User Data-

User data was collected after completion of each task from the JSON created by the user session. The JSON files were stored in repository and provided in the appendix.

Methodology –

***Efficiency (Paired T-tests)*:** Paired sample t-test is a statistical technique that is used to compare two population means in the case of two samples that are correlated.
The logging data was collected based on the number of queries, number of clicks and the time taken to complete each task on basic and the advanced faceted search engines. These were averaged for number of queries and clicks and the paired t-Tests values were calculated for time taken per task. Null hypothesis is that the mean of the distribution of differences is zero.

Formulae : $t = \frac{\overline{B-A}}{\sigma_{B-A}}.\sqrt{N}$ $\sigma = \sqrt{\sum_{i=1}^{N}(x_i - \overline{x})^2/N}$, where $\overline{x}$ is the mean.

Name	Basic Clicks	Advanced Clicks
Hillary	Task 2: 3	Task 7: 3
Irene	Task 3: 5	Task 8: 4
Justin	Task 1: 2	Task 6: 3
Ronald	Task 4: 10	Task 10: 2
Saptarshi	Task 9: 5	Task 5: 5
Average	**5**	**3**

Table 1:Average score for number of clicks

Name	Basic Queries	Advanced Queries
Hillary	Task 2: 1	Task 7: 1
Irene	Task 3: 1	Task 8: 1
Justin	Task 1: 3	Task 6: 3
Ronald	Task 4: 3	Task 10: 1
Saptarshi	Task 9: 1	Task 5: 1
Average	**1.8**	**1.4**

Table 2: Average score on number of queries

Name	Basic Time Taken	Advanced Time Taken
Hillary	Task 2: 22	Task 7: 7
Irene	Task 3: 25	Task 8: 7
Justin	Task 1: 123	Task 6: 47
Ronald	Task 4: 136	Task 10: 8
Saptarshi	Task 9: 19	Task 5: 6
	P-score: 0.0932	Advanced slightly better than basic

Table 3: Paired t-Tests based on time taken per task

Depending on the Paired t-Tests on the tasks performed by each user with respect to time taken per tasks the advanced faceted search was better than the basic with the score less than the 0.1 mark. For the average score the Basic has higher average than the advanced search.

***Effectiveness* -** Users were asked to check a checkbox next to the document in the search result list if that document is relevant to the task given and satisfies the task. The users marked the pages as relevant if they found the pages satisfied the query partially. If the documents retrieved by the query

has the data required to fulfill the task, then user has marked them as relevant. If the documents have achieved desired result as perceived by the user as per the task, then those documents are marked satisfactory. This data is recorded in the user logging JSON along with the position of the documents. If the documents are marked as both relevant and satisfactory then they are considered as part of the effectiveness analysis. For each task, all five of the users found relevant documents that satisfied the task. Tasks 6 (Find Samsung phones with 3.2 Inches 8GB ram) and 9 (Find 12MP, 3.5 inches' phones) were unique in that there were no correct answers for that task within the corpus. For the rest of the tasks user was able to find the satisfactory relevant documents in the first page of the result found (within top 10 position).

Sample JSON recorded as follows:

{"UserName":"Irene","UserSessionStartTime":"2016-05-30T23:26:10.807Z","SearchEngineSystem":"basic","Query1":"samsung $300", "Querytimestamp1":"2016-05-30T23:26:26.540Z","TotalQuery1ResultsClicked":4, "Query1MarkedAsRelevant":[{"position":"0","url":"http://www.gsmarena.com/samsung_gear_s-6620.php"},{"position":"3","url":"http://www.phonearena.com/phones/Samsung-Galaxy-Tab-A-9.7_id9365"},{"position":"2","url":"http://www.phonearena.com/phones/Samsung-Galaxy-Tab-A-9.7_id9365/fullspecs"},{"position":"1","url":"http://www.phonearena.com/phones/Samsung-Droid-Charge_id5126"}],"Query1MarkedAsSatisfied":[{"position":"3","url":"http://www.phonearena.com/phones/Samsung-Galaxy-Tab-A-9.7_id9365"},{"position":"2","url":"http://www.phonearena.com/phones/Samsung-Galaxy-Tab-A-9.7_id9365/fullspecs"},{"position":"1","url":"http://www.phonearena.com/phones/Samsung-Droid-Charge_id5126"}],"TotalQuery1MarkedAsRelevant":4,"TotalQuery1MarkedAsSatisfied":3,"TotalNoOfQueries":1,"TotalNoOfClicksInSession":4,"TotalNoOfSatisfiedInSession":3,"TotalNoOfRelevantInSession":4}

Name	Basic			Advanced		
	Task	Total Satisfied and Relevant	Position	Task	Total Satisfied and Relevant	Position
Hillary	Task 2	3	0,3,1	Task 7	2	6,7
Irene	Task 3	5	0,1,2,3,4	Task 8	4	0,1,2,3
Justin	Task 1	1	6	Task 6	0	NA
Ronald	Task 4	4	0,1,2,3	Task 10	5	0,1,6,8,9
Saptarshi	Task 9	0	NA	Task 5	1	2

Table 4: Analysis of satisfactory and relevant URL marked by user for effectiveness study

User Satisfaction (SUS and NASA TLX)- The user were given standard SUS questionnaire for each task they performed. The System Usability Scale (SUS) is a simple, ten-item scale giving a global view of subjective assessments of usability based on Likert scale.

Question 1,3,5,7,9 –> score -1

Question 2,4,6,8,10-> 5-score

The sum of all ten questions was multiplied by 2.5. The score should be higher than 68 for acceptability.

A standard NASA TLX questionnaire was given to users to evaluate the mental, temporal demand and effort and performance of the search engines. The online calculation tool for TLX scoring was used to evaluate the user ratings. These scores were calculated through weighted factors. The SUS and TLX shows that Advanced faceted search was slightly better than the basic search as per user satisfaction.

	SYSTEM USABILITY SCALE				NASA TLX			
Name	**Basic**		**Advanced**		**Basic**		**Advanced**	
Hillary	Task 2	62.5	Task 7	82.5	Task 2	18.33	Task 7	17.5
Irene	Task 3	80	Task 8	72.5	Task 3	17.5	Task 8	16.66
Justin	Task 1	60	Task 6	75	Task 1	23.33	Task 6	16.66
Ronald	Task 4	60	Task 10	75	Task 4	43.33	Task 10	25.33
Saptarshi	Task 9	80	Task 5	60	Task 9	23.33	Task 5	21.66
	AVG	**68.5**		**73**		**25.164**		**19.562**

Table 5: SUS and NASA TLX evaluation for user satisfaction

Challenges

There were numerous challenges faced during the course of the project. The page ranking was affected due to the noise content of the downloaded documents. Therefore, the noise removal algorithm had to be modified to handle various sections of the page including user comments, related news, other related products, ads, navigation etc.

Some pages didn't contain url in the metadata. This posed an issue during snippet creation. Therefore, during crawling and storing the documents the url of the page is added to the metadata through the java code module.

Facets, attribute determination and classification from the crawled documents also posed an issue as the specifications were varied and scattered over the pages. The metadata of each crawled page were different in terms of contents. Therefore, we considered the keywords in the metadata as the facets.

The logging from Login.html page to the Apache Solr was difficult as it was hard to connect Jetty with Java code or PHP. Storing the user logging information, queries, number of clicks per task was difficult as we were unable to send data from Solr to any database or text file. The issue was solved by creating a JSON for each user logging session and manually downloading the JSON after each task.

Future work

In future the search engine could be made more robust by improving on the stemming and tokenization. Also, the paging algorithm can be improved to show more relevant documents on the top. Multimedia corpus could be built and integrated to search for YouTube videos or Image files. Also, visualization for graphs or histograms with the search pages fetched or number of hits per page could be created. Also, logging data along with browsed pages can be stored in database. Currently the search engine is built as stand alone system but it could be hosted on the distributed platforms like Amazon EC2 to let multiple users use it.

Conclusion

The Search engine for mobile devices can retrieve the web pages for the queried devices with given specifications. The basic and the advanced search engine have the same effectiveness and efficiency with advanced have a very small edge over basic. The SUS and TLX study shows that there is still room for improving the usability and robustness of the engine.

Reference

1. Apache Solr tutorials: https://wiki.apache.org/solr/
2. Ajax Solr tutorials : https://github.com/evolvingweb/ajax-solr
3. Solr configuration and set up : http://www-scf.usc.edu/~csci572/2016Spring/hw3/IndexingwithTIKAV3.pdf
4. TLX task score : http://jensgrubert.bplaced.net/nasa-tlx-short/TLX-English-short.html
5. SUS score : http://hell.meiert.org/core/pdf/sus.pdf
6. Paired t-Test : http://www.graphpad.com/quickcalcs/ttest1.cfm
7. Lucene and Solr download pages : http://lucene.apache.org/solr/
8. Apache Solr Confluence: https://cwiki.apache.org/confluence/display/solr/Pagination+of+Results
9. Textbook : http://ciir.cs.umass.edu/downloads/SEIRiP.pdf
10. Lecture slides from Camino

Appendix

1. User Evaluation:
    a. User Session JSON
    b. Efficiency and evaluation
    c. SUS and TLX evaluation https://drive.google.com/a/scu.edu/folderview?id=0Bzb4OyxiP8LfYURGSUtTZFh1bzA&usp=sharing
2. Task List : https://docs.google.com/a/scu.edu/document/d/1ghIWagL0YzqXottp5oVdqUJd8dzZSeqlMsMBMSnH-sM/edit?usp=sharing
3. SUS and TLX questionnaire from Users https://docs.google.com/a/scu.edu/spreadsheets/d/1gA5hHX6uA8f7NGVCNn9FaYFJngjT1AYOOAR8FIdphsI/edit?usp=sharing