Search Engine with Emotions

Anish V. Mahesh
Courant Institute
New York University
avm358@nyu.edu

Sanchit Mehta
Courant Institute
New York University
sanchit.mehta@nyu.edu

Naman Kumar
Courant Institute
New York University
nk2239@nyu.edu

Abstract

Plato described emotion and reason as two horses pulling us in opposite directions. Rightly so, the curiosity of a person is not just limited to an information need but also to the emotional satisfaction. We have built a search engine that makes it easier for a user to query articles based on their tone, thereby making the search more human centric.

In this report we talk about the design and implementation of the system. We briefly discuss the heuristics of our modified algorithms to optimise the speed of the search and how the emotion/tone based search can be applied in the real world. We also present an evaluation of our search engine.

1 Motivation

A search engine satisfies a user's information need by trying to learn the intention of the user. The intention however, cannot be objectified to just a query of facts. An intention is often accompanied by the emotional state of the user. A satisfaction of this factual curiosity coupled with the emotional need is what will lead to a complete user experience. It is this experience that BITSearch aims to create by facilitating emotion selection along with the issued query.

Generally, people use search engine for one of three things: research, shopping, or entertainment[1].

1.1 How emotional search aids research

A user can have two intentions when ensuing a query:

He/she is looking for affirmation. By allowing the user to select an emotion matching his/her preset state of mind, we provide a shorter path to this need by presenting the user with more emotionally relevant results.

He/she is looking to gain knowledge or develop opinion about a certain topic. Our search engine aids this need by allowing the user to explore the topic in both, a joyous and a sad context.

1.2 How emotional search aids entertainment

When a user is using a search engine for entertainment, he/she would often like to see a topic in various lights. Funny articles would be more satisfactory for a user with this intent. In addition, he/she may want to be amused by reading both the happy and sad side of things.

1.3 Controlled randomness in human search behavior

When a user is not looking for something specific, he/she uses a search engine for gathering information. In such a case, a random surfing can often lead to dissatisfaction if the results do not agree

with the user's mood. Our search engine mitigates this by providing information for only the mood set by the user.

2 Monetisation through Targeted Advertisement

Research dictates that mood influences shopping decisions. A happy user finds an item more desirable[2]. This opens up avenues for presenting e-commerce advertisements to users making frequent happy and funny searches. This will increase the probability of clicks thereby driving CPC revenue. A sad user on the other hand, is a potential consumer for motivational and therapeutic advertisements.

Similarly, the search engine can be used to advertise content based on a user's search history. A user issuing queries for Donald Trump with 'Joy' as emotion can be suggested links to Right Wing sources and vice-versa.

A person who frequently searches for funny content could be advertised with links to websites selling humorous items. We can also target such consumers with advertisements focussed on branding of comical content.

3 Ranking

3.1 Threshold based Ranking on sorted posting lists

We have implemented a ranking algorithm which utilizes the EQ based sorted posting lists to retrieve emotionally relevant results faster. A thresholding mechanism is applied for this retrieval which gives us a performance boost in most cases.

Our mechanism considers implicit dynamic weights for relevance and emotion. Initially the weight for emotional score is predominantly high owing to the sorted nature of our posting list. If the required number of documents are not found, we relax the bound by increasing the threshold multiplicatively until the results are found. Thus, more documents are scored in each iteration suggesting an increase in the implicit weight of the relevance score at the cost of the emotional score.

The pre-processed posting list allows us to look through a bounded section of the list each time and compute the score accordingly.This is achieved by setting a threshold on the number of documents to be retrieved in each iteration. For multiple query terms, we take the intersection of the document sets retrieved. The documents thus retrieved are ranked on relevance by linearly weighting the query likelihood of the body and title text.

3.2 Comprehensive ranking using the intrinsic quality and relevance score

This ranking mechanism iterates over the entire posting list and scoring the documents as we go. The score is computed by a linear model which has weights representing the topical relevance score and the emotional score. The topical relevance is in turn another linear weighting of the query likelihood scores of the body and title text. The title score has been assigned a higher weight as our experiments showed that the title better signifies the relevance of a document than the body.

3.3 Threshold based ranking vs comprehensive ranking

3.3.1 Performance Analysis

For c as the threshold, n documents in the posting list and 2 as the multiplier.

Best Case: All the documents that need to be retrieved are present within the first threshold.

Time Complexity:

Table 1: Best Case Time Complexity

	Threshold based	**Comprehensive**
Documents Fetching	O(c), only up to c documents are looked at	O(n), all the documents are looked at
Score Computation	O(c), only c documents are scored	O(n), all the documents are scored

Average Case: All the documents that need to be retrieved are present in the first half of the posting list

Time Complexity:

Table 2: Average Case Time Complexity

	Threshold based	**Comprehensive**
Documents Fetching	O((n-2c)/2)*	O(n)
Score Computation	O(n/2), only unscored documents are rescored, scored documents have already been added to the result list	O(n), all the documents need to be scored

Mathmatical Computation

Documents fetched in cycle 1: c
Documents fetched in cycle 2: 2c
Documents fetched in cycle 3: 4c

Say, 2mc documents were fetched until we reached n/2. Hence,

$$2mc >= n/2$$
$$m >= log2(n/2c)$$
$$m = log2(n/2c)$$

when n is c times a power of 2. This could be done by manipulating the multiplier or threshold or both based on the median sizes of documents list for terms in the corpus.

Hence, total computations:

$$c+2c+3c+...+2mc = c(2m+1-1) <= c(2log2(n/2c)+1-1) \approx c(2log2(n/2c)-1) = c(n/2c-1) = (n-2c)/2$$

In case 2mc is a very large value, we still can have bounds on threshold growth such that it never increases beyond max(n/2) for all n for all documents list in use.

It is evident from the equation that increasing the value of c will result in faster computation for average case but will affect the best-case efficiency. In addition, we did not want to lose the dynamic lambda effect obtained through emotional efficiency barriers. We have chosen through

experimentation, an ideal c for our corpus using frequent query terms as our experimental dataset.

Worst Case: All the documents that need to be retrieved are present near the end of the posting list.

Time Complexity:

Table 3: Worst Case Time Complexity

	Threshold based	**Comprehensive**
Documents Fetching	O(2n-c), based on computations	O(n)
Score Computation	O(n), all documents need to be scored	O(n), all the documents need to be scored

3.4 Advantages of Threshold based Ranking

- Faster computation and ranking
- Dynamic weight variation
- Can be enhanced by tuning parameters to make a more efficient, though complex ranker
- Given the need for higher emotional score weights for our corpus, this mechanism provides the required dynamic nature while improving the retrieval speed.

3.5 Disadvantages of Threshold based Ranking

- Observed to be slower for descriptive queries, the need for which was prodigiously eliminated by an emotion based search engine.
- Observed to be slower for unrelated phrase terms as the phrase needs to be checked in each document multiple times at every iteration. This problem could be eliminated with the use of an enhanced data structure to perform the check in less time.
- Observed to be slower for large number of documents searched or as we get nearer to the worst case. This problem can be resolved by stopping to look beyond a certain limit in the posting list as we can surely say we have searched through all the emotionally good documents since the posting list is sorted on the intrinsic quality. In our search results, emotions make for important parameters and we can avoid serving a document with a poor emotional score.

4 Ranking for Funny Documents

4.1 Modified Query likelihood

This ranking mechanism iterates over the entire posting list and scoring the documents as we go. The score is computed by a linear model which has weights representing the topical relevance score and the emotional score. The topical relevance is in turn another linear weighting of the query likelihood scores of the body and title text. The title score has been assigned a higher weight as our experiments showed that the title better signifies the relevance of a document than the body.

5 Indexing

The purpose of Indexing in a Search Engine is to collect, parse and store data for efficient and fast information retrieval. In accordance with this, our Search Engine uses the Indexer to ensure speed and performance in retrieving documents relevant to both, the query and the emotion selected along with the query.

To facilitate keyword based search, we build a Posting List based. Given the corpus contains tens of thousands of documents, a term can have a very long posting list. A linear search right up to the end of the list would take up a lot of time. In addition to this, each document has an intrinsic score based on the emotions conveyed in its content, which was calculated during the mining phase. We also need to ensure that the documents retrieved by the search have a good intrinsic score for the input emotion. To handle the above two constraints, we construct separate posting lists for separate emotions in the system. Each posting list then, is sorted based on the document's intrinsic score for that emotion. This way, we are guaranteed that the best documents will be at the front of the list and so we do not always need to search up to the end of the list. Given a query term and an emotion, we load the specific posting list for that term and perform a threshold based search (explained in the section on Search). Thus we reduce the time taken to retrieve results.

Constructing a data structure to hold posting lists for each term in a large corpus implies a great strain on memory. Constructing three such indexes further compounds this strain. Efficient memory management is thus a necessity for such an indexing. We achieve this by implementing a local MapReduce.

5.1 Mapping

As we iterate through each document, we add each term encountered to a hashmap in order to append future documents to the same term's posting list. Once we reach a pre-defined document count, we flush the posting lists created up till now to a file and clear them from memory. This process continues till there are no more documents to be processed, at which point there will exist multiple index files with a single term's posting list split across them.

5.2 Shuffle

We now merge all these separate indexes into one. We do this by applying a merge sort algorithm on the term ids. Whenever we encounter the same term id in different files, we append the posting lists together. This way we ensure the whole posting list of one term is now in one place.

5.3 Reduction

Now that we have each posting list as a whole in one place, we sort the list based on each document's intrinsic emotion score. It is here that we make multiple copies of the data for each emotion in our search engine as the sorting would be different for each of these emotions. Further, we split this merged index into multiple smaller files on the number of terms. Lastly, we compress these posting lists using the vByte compression logic to reduce the disk usage.

6 Mining

The mining phase is used for computing additional attributes of the documents in the corpus. One instance is the computation of intrinsic qualities of each document in the corpus, like PageRank or Number of views. The intrinsic score of a document in our search engine is the result of an Emotion/Tone Analysis of its text. A score is calculated for every emotion in the system and stored as tab separated values against the document name in a file (emotions.tsv). This file is then read during the Indexing phase to assign each document its intrinsic scores.

Tone Analysis is an enhanced version of Sentiment Analysis, which refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials. While Sentiment Analysis generally aims to determine the overall contextual polarity of a document, Tone Analysis further buckets a document into the various emotions conveyed in it.

A standard Tone Analyzer works by learning and then applying patterns in text that enable it to classify the said text into a bucket[3]. In the context of our search engine, these patterns are phrases with opinions or those that utilize words of a particular emotion. The buckets here are the two emotions, Joy and Sad. The Tone Analyzer reads through the document and comes up with a Joy

score which signifies the amount of joyous emotion used in the article, and a Sad score which informs us of the usage of sad or depressing content.

For the implementation of a Tone Analysis, we carried out extensive research to find the right tool. We experimented with various Open Source Tone Analysis tools on the web and also tried implementing our own Sentiment Analysis using the Natural Language Toolkit of Python. We used datasets from the web to train a Nave Bayes classifier in NLTK. The classifier then gave a positive and negative score for our test documents. We extrapolated these scores to represent Joy and Sad. There were various issues with this model. Firstly, the classification was based more on examining individual words and classifying them as either positive or negative. This does not work so well in classifying a document based on context and opinion. Secondly, the dataset we trained the model on were reviews labelled as positive or negative. The nature of words used in articles as opposed to reviews are very different and thus not compatible. We tried to explore the different kinds of datasets available on the web but could not find anything suitable to our need. The Open Source APIs we experimented with were: Tone Analyzer, IBM Watson's Alchemy API[4], Sentiment Analysis by Vivek Narayanan[5], Emotion Detection API by Carlos Argueta[6].

After our experiments with all these tools, we shortlisted the IBM Watson Alchemy API and the Emotion Detection API. We contacted its developer, Carlos Argueta for enquiring about the API. He provided us with additional documentation for using his API and gave us information on the technicalities of the API. Though the API was performing well for sentences, it did not perform very well for whole documents. The led us to go with the Alchemy API developed by IBM Watson for our project.

We have integrated the Alchemy API with our search engine. Once we have crawled all the documents, the text of each of the document is then passed to the Alchemy API and it gives us a result which comprises of scores for various emotions. We compute each document's Joy and Sad score by taking a composite score from the results of Alchemy. The scores for all the documents are then persisted in the emotions.tsv file.

7 User Interface

The UI presents an interface for a user to execute a search query by choosing an emotion and look through the results in an user friendly way. To achieve this have used the following GPU Licenced librararies : Twitter Bootstrap[7],StartBootstrap[8], JQuery[9] and FontAwesome[10].

The front end has two main features

- **AutoComplete :** Autocomplete enables a user to quickly find and select query suggestions from a pre-populated list of most searched terms by other users. Every query a user presents is logged into one of the log files, named from a to z - depending upon the first character of the query. This logging also ensures that no same query is logged twice and that particular search term's search count is incremented . To fetch the results to the front end : every change of a character in the search bar invokes an Ajax call to the backend to retrieve suggestions starting from this substring. To ensure these log files always return the most searched terms first, these log files are sorted by no of searches as a cron job after every 5 minutes.

- **Pagination :** Through Pagination we divide the search results into discrete pages enabling us to deliver results faster without having the need to present all the results at once. We have achieved this by integrating an extra parameter in the GET request called pagination. In the backend, we handle this by searching for one extra result than the no of results requested. This one extra result implies the possibility of a next page page of results.Finally, we pass only the no of results requested to the front end, but with a flag indicating the existence of next page of results. This flag is then used to decide whether to display the next button or not.

8 Crawling

We have forked a lightweight open sourced crawler : Crawler4J[11] and modified it to systematically browse white listed URL's and create a corpus for the documents using a max depth of 1500. While crawling the documents retrieved are parsed and main content is extracted using shallow text detection to remove the boiler plate[12]. Enabling the storage of strictly important content and thereby optimising the space.We have also written custom url extraction methods to extract URL's out of pages having redirects and infinite scroll.

9 Evaluation

We selected 4 queries of varying natures i.e a name, a place, an animal, a company and created a carefully curated corpus of 20 documents. The 20 documents were so chosen that 10 of them are relevant documents, both in terms of topical relevance and emotional scores, and 10 are non-relevant. This choice of ours was an aim at introducing maximum randomness. We evaluate the corpus of 20 documents for DCG at 5, 10 ,15 and 20 while considering two baselines for our evaluation:

- The documents scored by only relevance.
- The documents scored by only emotion.

These baselines help us determine how our search engine performs in gathering results that equally represent both topical relevance and emotions. Figure 1 shows the plots for the DCG values of our search engine against the baselines for each query.

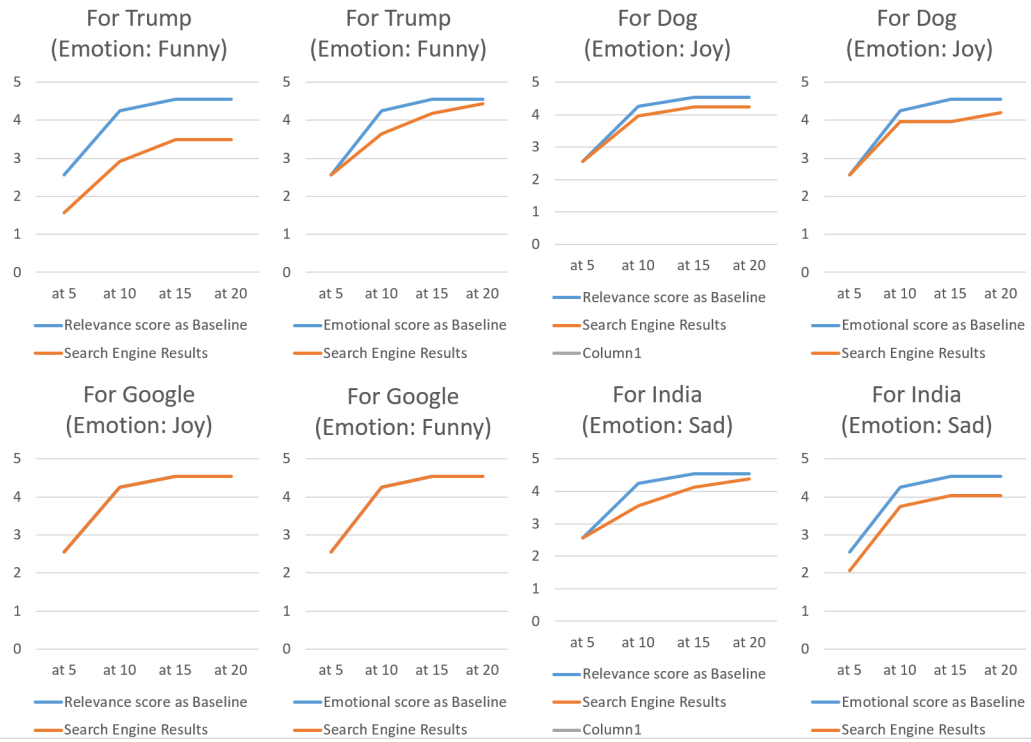

Figure 1: DCG

10 Observations

Through experimentation we have observed that our search engine performs best for queries on entities (more specifically, nouns). The ranking algorithm does not work efficiently for descriptive or long queries. Further, the emotion analysis works better for opinionated articles, especially editorials and conversations.

References

[1] W. S. D. for Dummies. (2016, Nov.) Why people use search engines : Research, shopping and entertainment. [Online]. Available: http://www.dummies.com/web-design-development/search-engine-optimization/why-people-use-search-engines-research-shopping-and-entertainment/

[2] R. Nauert. Mood influences shopping decisions. [Online]. Available: https://psychcentral.com/news/2010/02/18/mood-influences-shopping-decisions/11552.html

[3] M. S. Namrata Godbole and S. Skiena, "Large-scale sentiment analysis for news and blogs," *International Conference on Web and Social Media*, 2007.

[4] Ibm watson alchemy api. [Online]. Available: https://www.ibm.com/watson/developercloud/alchemy-language.html

[5] Sentiment analysis. [Online]. Available: http://sentiment.vivekn.com/

[6] C. Argueta. Emotion detection. [Online]. Available: http://demo.soulhackerslabs.com/emotion/analyze/

[7] Twitter bootstrap. [Online]. Available: http://getbootstrap.com/

[8] Start bootstrap. [Online]. Available: https://startbootstrap.com/

[9] Jquery. [Online]. Available: https://jquery.com/

[10] Font awesome. [Online]. Available: https://fontawesome.io

[11] Y. Ganjisaffar. Crawler4j. [Online]. Available: https://github.com/yasserg/crawler4j

[12] W. N. Christian Kohlschütter, Peter Fankhauser, "Boilerplate detection using shallow text features," *Web Search and Data Mining*, 2010.