Contents

1. Motivation

To provide a client-side solution to deliver a personalized experience when using a search engine.

Our engine will analyze user activity and try to build an approximate model of the user's interest and biases. We will use this to modify the ranks of web-pages served by a search engine to better match the predicted interests of the user.

Since our algorithm can run completely on the client-side, the user does not need to share his browsing behaviour and activity with any search engine, thus benefiting individuals concerned for their privacy.

Additionally, this approach can easily accommodate the case of the user using multiple search engines and multiple browsers, which can enable us to build a more complete model for the user's interest than any single search engine can build.

2. Approach

There are two parts to our approach.

1. Data collection and user profiling.
2. Reranking search results based on a model of users' preference.

2.1 Data Collection

This section of the algorithm is responsible for tracking the user activity and building a model of the user's preferences.

2.1.1 Live User Activity Tracking

We improved upon the previous approach of statically obtaining user history once a day.

Using a Python Flask server and a Chrome browser extension, we are employing a live data-collection system which monitors the user's activities in real-time and logs statistics which we consider to be indicative of the user's interest in that particular site.

Statistics currently implemented in the extension:

- Web Pages visited
- Active time on that webpage
- Distance scrolled on the page

```
kanishk509@kanishk-dell:~/code/sop/flask_server$ python3 time_track_server.py
 * Serving Flask app "time_track_server" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
Scroll recorded on: whatsapp.com, dist = 0
127.0.0.1 - - [14/Mar/2020 12:02:44] "POST /scroll_update HTTP/1.1" 200 -
Open website: codeforces.com
127.0.0.1 - - [14/Mar/2020 12:02:44] "POST /send_url HTTP/1.1" 200 -
Scroll recorded on: codeforces.com, dist = 975.2000122070312
127.0.0.1 - - [14/Mar/2020 12:02:49] "POST /scroll_update HTTP/1.1" 200 -
Open website: google.com
127.0.0.1 - - [14/Mar/2020 12:02:49] "POST /send_url HTTP/1.1" 200 -
Scroll recorded on: google.com, dist = 0
127.0.0.1 - - [14/Mar/2020 12:03:36] "POST /scroll_update HTTP/1.1" 200 -
```

Image: Active time and scroll updates being logged.

2.1.2 Keyword Extraction

Since we are operating under the basic assumption that a user is more likely to visit web pages that are similar to the web pages already visited by him in the past i.e pages in his browser history, we need some concept to measure the similarity between web pages. For this, we have used keywords to measure the degree of similarity between web pages.

Keyword extraction is a technique used to extract the most important words and expressions which are indicative of the topic that the given text is talking about. This technique enables us to simply look at a few words indicative of the topic of the texts to measure the similarity between two pages rather than going through their full content.

There are a number of open-source keyword extraction libraries available but we'll be using YAKE, which is an unsupervised approach for automatic keyword extraction using text features, due to reasons mentioned in Vishwa's thesis. It associates a score with every keyword that it extracts which is inversely proportional to its importance in the text i.e more the importance of a keyword, the lesser the score associated with it and vice versa.

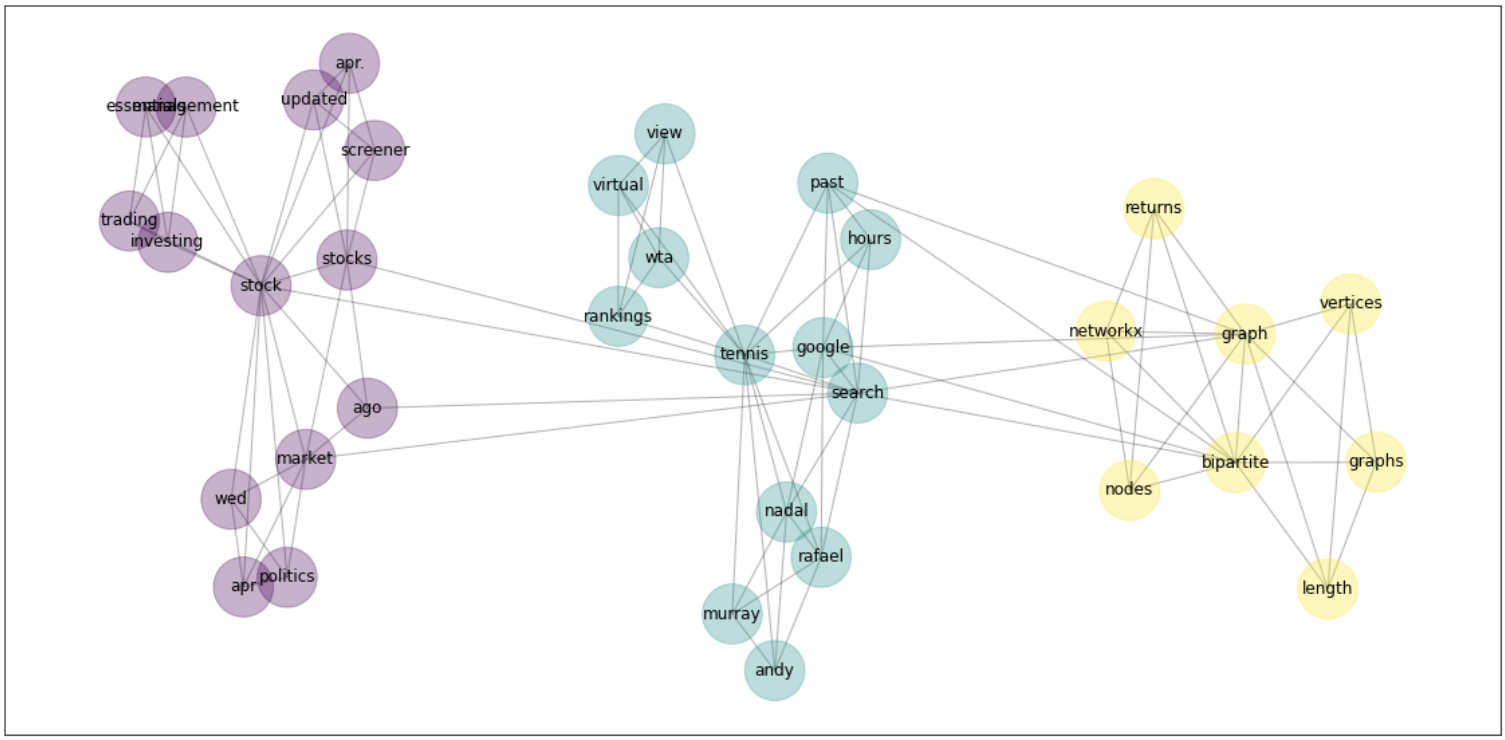
2.1.3 Clustering to Topics

We define(and maintain across session) a graph as follows:

- Each unique keyword in our database represents a node.
- If there exists any webpage which has two keywords A and B in its body, then the nodes corresponding to these keywords will have an edge between them.

Our motivation in doing so is that keywords relating to a particular topic will form *clusters*, and we can use community detection algorithms to classify the nodes into clusters, i.e., keywords into topics.

We use the *Louvain* community detection algorithm to identify the clusters in our graph.



Using the discussed approach, our algorithm classifies the keywords into 3 clusters(topics):

1. Keywords relating to **Graphs** (Yellow)
2. Keywords relating to **Stocks and Investments** (Purple)
3. Keywords relating to **Tennis** (Green)

2.2 Reranking

We first approximate the user's preferences by choosing the best cluster of words based on the scores of the keywords in each cluster and then, using the keywords present in our best cluster and their respective scores, rerank the retrieved webpages.

2.2.1 Cluster Selection

The keywords are scored based on the relevance feedback of the webpages they were extracted from. We make an edge from each webpage to its keywords in the cluster. These edges are multidimensional in the sense that they carry multiple weights corresponding to the various parameters of our relevance feedback. The

contribution to a keyword's score by an edge can be calculated by getting its dot product with a weight matrix that defines the weightage to be given to our parameters in scoring as shown below.

$$score\_contribution = [time\_spent, scroll\_activity] \cdot [weight\_for\_time\_spent, weight\_for\_scroll\_activity]$$

Thus we get the final score of a keyword by adding up the contribution of all the webpages it was present in. Also since these parameters keep on getting updated as the websites are accessed, again and again, it also indirectly takes into consideration the number of times a page has been visited as a parameter.

Finally, for each cluster, we add up all the scores of keywords present in them and choose the one which has the maximum score.

2.2.2 Scoring and Re-ranking

Since we have now extracted the cluster which, according to our assumptions, is the best keyword-based approximation of our user's interest, all we need to do is rerank the retrieved web pages on the basis of their alignment with the same.

For this we calculate the final score of each website in consideration by the given formula:

$$final\_score = \sum_{keyword \in best\_cluster} count\_in\_page[keyword] \times score[keyword]$$

Finally, we arrange the pages in the decreasing order of their final scores which gives us their final rank.

2.3 Relevance Feedback and Learning

After we produce a re-ranked list of results to the user, we monitor his behaviour using our browser extension and use it to gauge the performance of our re-rank system.

The way in which the user interacts with a particular result in our re-ranked list provides metrics of the relevance of that result, and our algorithm learns as follows:

Positive reinforcement, i.e. user satisfaction high-

- Consider the case when the user opens a webpage and finds it suitable to his needs. This implies that this page has high relevance with respect to our search query.
- The user spends time on that page, and high active time is recorded by our browser extension. Similarly, high scroll distance is also captured.
- The algorithm assigns high weight to that website and the keywords from that website.
- In the future, that page(and similar pages) will have a high probability of getting a good rank if the user inputs a similar search query.

Negative reinforcement, i.e. user satisfaction low-

- Let us suppose our algorithm gives a top rank to a webpage. The user visits that webpage but is not satisfied with the result.
- The user exits that page quickly. The active time and scroll distance on that page are registered as low.
- This means our algorithm has performed poorly in giving a top rank to that page.
- The algorithm learns and improves by giving a low weight to that page and the keywords extracted from that webpage.
- In the future, that page(and similar pages) will get a lower rank for similar queries.

3. Results

We ran several instances of the algorithm on our local machines and recorded the results and how the ranking produced by the algorithm deviated from the ranks that the search engine provided.

The following is a demo of the algorithm that was run:

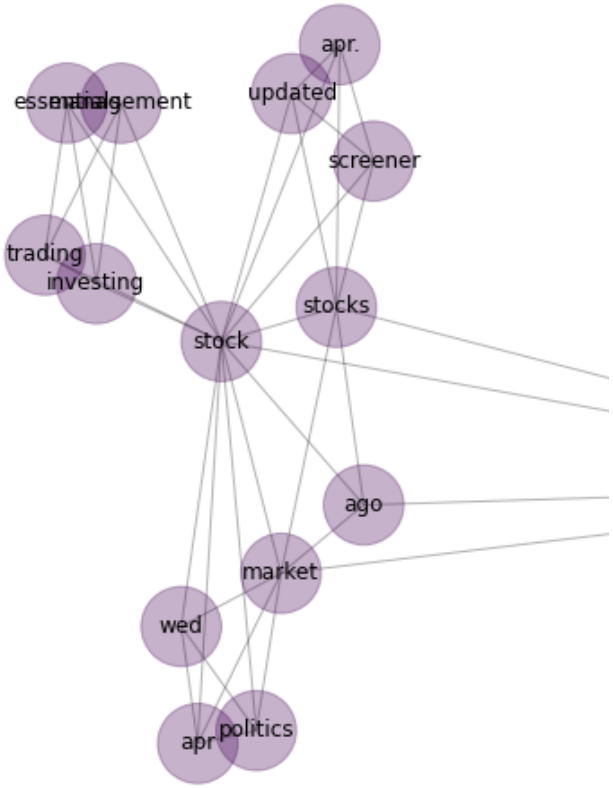
The algorithm tracked the browsing habits of the user while the user browsed various websites related to primarily 3 different areas: Graphs, Finance and Tennis.

The algorithm built up a database of keywords and their score. A graph was also maintained which was also used to cluster the keywords into topics. (Figure in section 2.1.3)

One of the queries we then tried on the algorithm was ***'nasdaq market'***.

The algorithm correctly clusters the data and produces the best-matching cluster which consists of keywords relevant to the domain of stocks and investment.

A small subset of that cluster:



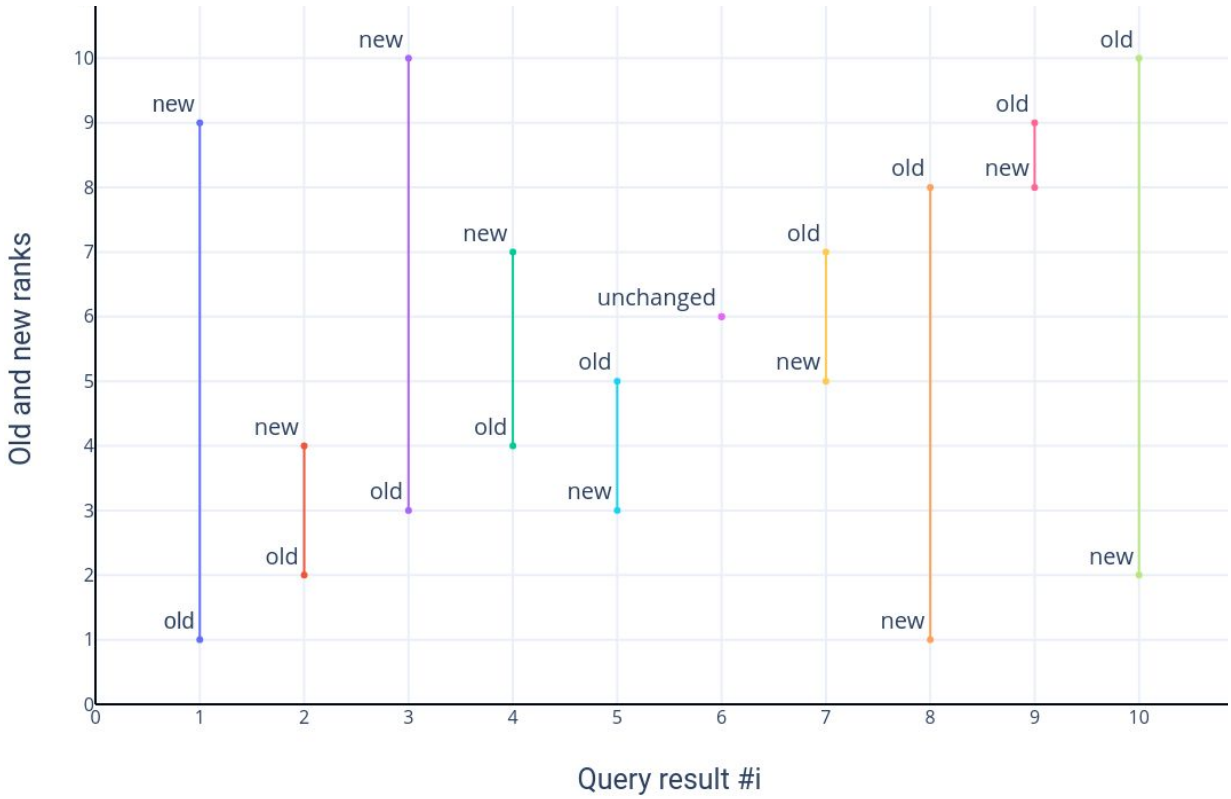
The 10 search results returned by the query were:

```
Query: nasdaq market

Initial rankings

Rank 1: https://www.nasdaq.com/
Rank 2: https://money.cnn.com/data/markets/
Rank 3: https://www.marketwatch.com/investing/index/comp
Rank 4: https://www.nasdaq.com/market-activity/indexes
Rank 5: https://www.nasdaq.com/authors/the-market-intelligence-desk-team
Rank 6: https://www.nasdaq.com/news-and-insights/markets
Rank 7: https://www.nasdaq.com/about
Rank 8: https://www.marketwatch.com/investing/nasdaq-stock-market
Rank 9: http://www.nasdaqtrader.com/Trader.aspx?id=TradingUSEquities
Rank 10: https://www.investopedia.com/terms/n/nasdaq-smallcap-market.asp
```

The deviation after the algorithm re-ranked the pages:



4.Further Work

1. More robust and informative relevance feedback. We can monitor how the user interacts with ‘good’ and ‘bad’ results in more detail and get stronger positive or negative feedback.

2. Some of the online datasets that we found were very vast and not aligned with our implementation, we couldn’t test our approach on them. That is something that can be looked further into.

3. Currently, some of the hyperparameters used in assigning weights of different components of relevance feedback are hardcoded. This can be optimized by tweaking, trial and error and machine learning techniques.

4.References

- Graphs and their application in User profiling for search (Thesis report, Vishwajeet Singh)

- Matthijs, Nicolaas, and Filip Radlinski. "Personalizing web search using long term browsing history." Proceedings of the fourth ACM international conference on Web search and data mining. 2011.

- Berlingerio, Michele, et al. "Multidimensional networks: foundations of structural analysis." World Wide Web 16.5-6 (2013): 567-593.

- De Meo, Pasquale, et al. "Generalized louvain method for community detection in large networks." 2011 11th International Conference on Intelligent Systems Design and Applications. IEEE, 2011.

- https://flask.palletsprojects.com/en/1.1.x/#api-reference