Improving Yioop! User Search Data Usage

PROJECT ADVISOR: DR. CHRIS POLLETT

COMMITTEE MEMBERS: DR. MARK STAMP & DR. SOON TEE TEOH
Agenda

- Introduction
- Project Goal
- Background
- User Data Visualization
- Re-rank Yioop Result
- Related keywords
- Test and Result
- Demo
Introduction

- Users past search history can be used to provide customized search results.

- Commercial search engines like Google, Bing provide this feature called as Personalized search.

- However, storing of this user data in the server has some privacy concerns.
Project Goal

- The goal of the project is to use the user search data and provide valuable features to Yioop User without any privacy issue.
- Provide a visualization tool to see the search history.
- Customize Yioop Search result based on past user searches.
- Provide related searches.
We use Firefox extension for building the features because

- Build user confidence as user can view the code
- Provides Storage API to access browsers history
- Easy to implement using popular scripting language like Javascript
Firefox Extension

- Extensions allow users to add functionality to the browser and enhance the user interface.

- They are distributed in the form of zip bundle with a xpi (pronounced “zippy”) extension.

- Basic component of the extension
  - install.rdf
  - chrome.manifest
  - main.xul
Basic Components

- **install.rdf** – It contains details like unique id, version, min and max version of the target application details, etc. This file is read for installation.

- **chrome.manifest** – It contains folder hierarchy, skin details and the xul file to be overlaid on the browser.

- **main.xul** – It contains the UI details that needs to be overlaid on the browser. It also adds functionality by including JavaScript files.
One of the user benefits is to provide a visualization graph of the users past search history.

In this graph, the nodes represent the unique urls visited and the edge represents the navigation path to reach the url.

For this, we are using Force directed algorithm to draw the directed graph.
Force Directed Algorithm

- In this algorithm, there are two forces assigned at the edges and the nodes

- **Hooke’s Law** - If the spring is compressed or extended and released, it returns to its original, or natural, length, provided the displacement is not too great.

  \[ F_x = -k(x - x_0) = -kx \]

  Where \( k \) is the force of constant of the spring

- **Coulomb’s Law** - The magnitude of the Electrostatics force of interaction between two point charges is directly proportional to the scalar multiplication of the magnitudes of charges and inversely proportional to the square of the distances between them.

  \[ |F| = k_e \frac{|q_1 q_2|}{r^2} \]

  Where \( K_e \) is the repulsion constant and \( q_1, q_2 \) are the two point charges.
// Place the nodes at random position and initialize their velocity to (0,0)
Loop
   // Initialize the total kinetic energy
   kinetic_energy = 0;
   for each node
      // Net force of this particular node
      net_force = (0, 0);
      // Calculate the effect of Coulomb's Law
      for each other node
         net_force = net_force + Coulomb_repulsion (this_node, other_node);
      next node
      // Calculate the effect of Hooke's Law
      for each spring connected to this node
         net_force = net_force + Hooke_attraction (this_node, spring);
      next spring
      // Update the velocity of the node using a damping constant (0 < d < 1)
      // Here, we are using the damping constant to be 0.5
      this_node.velocity = (this_node.velocity + timestep * net_force) * 0.5
      // Update the node's position
      this_node.position = this_node.position + timestep * this_node.velocity
      // Update the kinetic energy of the system
      kinetic_energy = kinetic_energy + this_node.mass * (this_node.velocity)^2
   next node
   until kinetic_energy < 0.01 // A small constant
Sample Graph
The second goal of the project is to provide customized search result based user’s past searches.

Capture user searches and store it in the local machine.

Re-rank the Yioop result page at runtime based on these data.
Capture user search data from other search engines like Google, Yahoo, Bing, Yioop.

<table>
<thead>
<tr>
<th>Field</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword</td>
<td>The search query user entered in the search engine</td>
</tr>
<tr>
<td>url</td>
<td>The destination url the user reached by clicking the search result</td>
</tr>
<tr>
<td>title</td>
<td>Title of the destination page</td>
</tr>
<tr>
<td>visitcount</td>
<td>Keeps track of the number of visits</td>
</tr>
<tr>
<td>searchfrom</td>
<td>Keeps track of the search engine</td>
</tr>
<tr>
<td>timestamp</td>
<td>Keeps an update of the latest time stamp</td>
</tr>
</tbody>
</table>
Storage of user data

- The user searches are stored in the local machine.

- It is stored in the form of sqlite database.
  - Why?
    - It is lightweight
    - Easy to access using javascript
    - Readily available apis for data manipulation
How to manipulate the Yioop Result Page?

- Three ways to manipulate the Document Objet Model (DOM)
  - **Load Events** – Add a listener when the Yioop result page loads and start manipulating the DOM.
  - **HTTP Observer** – In this, the page is captured at the HTTP notification event and update it.
  - **WebProgressListeners** – More sophisticated way of intercepting and modifying at various stages of load event
Re-Rank Yioop Result

- Uses the “visitcount” to determine whether the result is included in the Yioop page.

- Use the “Load Event” method to manipulate the data.

```javascript
window.addEventListener("load", function load(event) {
    window.removeEventListener("load", load, false);
    myExtension.init();
}, false);

var myExtension = {
    init: function () {
        // The event can be DOMContentLoaded, pageshow, pagehide, load or unload.
        if (gBrowser)
            gBrowser.addEventListener("DOMContentLoaded", this.onPageLoad, false);

        // Initialization logic can be put here
    },
    onPageLoad: function (aEvent) {
        // Code that manipulates the web page
    }
};
```
Existing Yioop Search Result
Re-ranked Yioop Result Page
The final goal is to provide related keywords in the Yioop result based on the past searches.

Use the past user search keywords and calculate the most relevant.

We use Okapi BM25 to calculate the related keywords.
Okapi BM25

\[
\text{score}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})}
\]

where, \( \text{IDF}(q_i) \) is the inverse document frequency,

\( f(q_i, D) \) is the \( q_i \)'s term frequency in the given document,

\( k_1 \) and \( b \) are free parameters with \( k_1 = [1.2, 2.0] \) and \( b = 0.75 \)

\(|D|\) is the length of the document \( D \) and

\( \text{avgdl} \) is the average document length

\[
\text{IDF}(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}
\]

where, \( N \) is the total number of documents and

\( n(q_i) \) is the number of documents containing \( q_i \).
Related Keyword result

![Search results for camera with related searches](image-url)
Demo
Tests and Result

- Used feedback from five volunteers to test the re-rank feature and the related keyword feature.

- Used the measure of Precision and Recall to calculate the effectiveness of the feature.
  
  - **Precision** is the fraction of the result set that are relevant.
    
    $\text{Precision} = \frac{|\text{Rel} \cap \text{Res}|}{|\text{Res}|}$
  
  - **Recall** is the fraction of relevant documents that appear in the result set.
    
    $\text{Recall} = \frac{|\text{Rel} \cap \text{Res}|}{|\text{Rel}|}$
Recall comparison of Yioop and Re-Ranked Yioop Result
Re-rank result

Precision comparison of Yioop and Re-Ranked Yioop Result
Related keywords

![Graph showing related keywords]

- Effectiveness
- Satisfaction
- Overall

Number of Related Keywords
Conclusion

- The project improves the Yioop user experience by using the users' past searches.
- The re-rank feature has improved the mean recall value from 0.23 to 0.56 when top 5 results are added.
- The re-rank feature has improved the mean precision value from 0.33 to 0.51 when top 5 results are added.
- The related keywords is most effective with the maximum of 6 relevant results.
Questions