



**University of  
Nottingham**

UK | CHINA | MALAYSIA

# Visual Analytic for Sensemaking

Submitted April 2023, in partial fulfillment of  
the conditions for the award of the degree **BSc Computer Science with AI**.

**20214825**

**Supervised by Kai Xu**

School of Computer Science  
University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the  
text:

Signature: Q.C.

Date: 22 / 4 / 2023

I hereby declare that I have all necessary rights and consents to publicly distribute this  
dissertation via the University of Nottingham's e-dissertation archive.

Public access to this dissertation is restricted until: DD/MM/YYYY



## **Abstract**

The difficulties in completing sensemaking tasks on a daily basis primarily origin in two aspects, the ambiguous and subjective nature as well as the high demanding of cognitive ability, which respectively correlate to two stages of sensemaking. While various tools and frameworks are available on the internet that focus on information gathering stage, support for other sensemaking stages is often lacking. This project aims to address this issue by combining a visual interface with machine learning models to provide users with support in both stages. The project is presented as a Google Chrome extension, which allows users to gather information and perform exploratory data analysis without leaving the webpage or switching between different software. In the information foraging loop, the workflow is simplified, and information is presented in a more intuitive way to reduce cognitive overload. In the sensemaking loop, recommendations are generated based on users' patterns and past clickstream data.



## Acknowledgements

I would like to express my sincere gratitude to my supervisor, Kai Xu, for his invaluable guidance, support and patience throughout the project. His insightful comments have greatly contributed to the quality of this work.

I am also grateful to the participants of the data collection process, who generously gave their time and shared their thoughts. Without their contribution, this work would not have been possible.



# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgements</b>	<b>i</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	2
1.2 Aims and Objectives . . . . .	2
1.3 Description of the work . . . . .	3
1.3.1 Functional Requirements . . . . .	3
1.3.2 Non-functional Requirements . . . . .	4
<b>2 Background and Related Work</b>	<b>6</b>
2.1 Visualization . . . . .	6
2.2 Foraging Loop . . . . .	6
2.2.1 Visual Summarization . . . . .	6
2.2.2 Pattern Recognition & Sequence Prediction . . . . .	7
2.2.3 Sensemaking Loop . . . . .	8
<b>3 Design</b>	<b>10</b>
3.1 Chrome Extension . . . . .	10
3.2 RNN . . . . .	12
3.2.1 Web Scrappers . . . . .	12
<b>4 Implementation</b>	<b>13</b>
4.1 Web Scraper . . . . .	13

4.2	Dataset . . . . .	14
4.2.1	Preparation . . . . .	14
4.2.2	Email . . . . .	15
4.2.3	Target participants & Results . . . . .	15
4.3	RNN Model . . . . .	17
4.3.1	Initial Attempt . . . . .	17
4.3.2	Main Development Loop . . . . .	19
4.3.3	Modification of preprocessing of URL . . . . .	23
4.4	Interface . . . . .	27
4.4.1	HTML . . . . .	29
4.4.2	Script . . . . .	30
4.4.3	Style . . . . .	34
4.5	Chrome Extension . . . . .	34
4.5.1	Extension Display . . . . .	36
4.5.2	Problem encountered . . . . .	37
<b>5</b>	<b>Evaluation</b>	<b>38</b>
5.1	RNN . . . . .	38
5.2	Browser extension . . . . .	38
<b>6</b>	<b>Summary and Reflections</b>	<b>39</b>
6.1	Project management . . . . .	39
6.2	Contributions and reflections . . . . .	40
6.2.1	The bigger picture . . . . .	41
	<b>Bibliography</b>	<b>41</b>
	<b>Appendices</b>	<b>45</b>
	<b>A User Manuals</b>	<b>45</b>
	<b>B User Evaluation Questionnaire</b>	<b>46</b>



# List of Tables



# List of Figures

4.1	IP address block . . . . .	13
4.2	Instructions for data collection . . . . .	15
4.3	Data structure . . . . .	16
4.4	Required libraries for development . . . . .	18
4.5	Initial attempt . . . . .	18
4.6	Data extraction . . . . .	19
4.7	Dataframe and its information . . . . .	19
4.8	Label encoding and onehot encoding . . . . .	20
4.9	Encoding Result . . . . .	20
4.10	Getting Time Sequence . . . . .	21
4.11	RNN model structure . . . . .	22
4.12	Training process . . . . .	23
4.13	saved model structure . . . . .	23
4.14	RNN performance . . . . .	24
4.15	Additional libraries . . . . .	24
4.16	Import result . . . . .	25
4.17	regex and tokenization . . . . .	25
4.18	regex&tok result . . . . .	26
4.19	Lemmatization . . . . .	26
4.20	Lemmatization_result . . . . .	27
4.21	URL vocab . . . . .	27
4.22	URL vocab result . . . . .	28

4.23	New encoding . . . . .	28
4.24	lexical RNN performance . . . . .	28
4.25	vue example structure . . . . .	29
4.26	APP.vue template1 . . . . .	30
4.27	APP.vue template2 . . . . .	30
4.28	HTML result . . . . .	31
4.29	Compile Error . . . . .	31
4.30	Script 1 . . . . .	32
4.31	Script 2 . . . . .	33
4.32	Add school . . . . .	33
4.33	Add criterion . . . . .	33
4.34	Load data from site . . . . .	33
4.35	Read Excel to array . . . . .	33
4.36	CSS . . . . .	34
4.37	Example command window for chrome-ext cli . . . . .	34
4.38	Example extension structure . . . . .	35
4.39	Loading extension into chrome . . . . .	35
4.40	Using extension . . . . .	36

# Chapter 1

## Introduction

The field of Sensemaking and its related disciplines have gained considerable attention since its introduction in the 1970s. This concept has been found to be closely interrelated to various aspects of society in comparison to its rare presence in secular life. Weick et al. (2005) defined sensemaking as the continuous retrospective development of plausible images that rationalize individuals' actions, usually comprising two stages, the information foraging loop, and the sensemaking loop. This process of clarification enables individuals to accumulate accuracy in meaning while decreasing confusion (Munya et al., 2005).

Despite the significance of sensemaking, an imbalance of development of support in different stages can be observed, that is, individuals can get access to extensive handy tools for foraging loop while remain struggling in the sensemaking loop. Therefore, this project aims to provide further support for browser-based sensemaking tasks in both stages, such as selecting appropriate graduate schools to apply for or purchasing a camera suitable for taking pictures of children. This involves incorporating more automation into both loops, leveraging the power of machine learning to better understand users' intentions and providing interactive visualizations to reveal latent relationships between features and outcomes. The ultimate objective of this project is to enable users to achieve better performance in complex sensemaking tasks.

## 1.1 Motivation

Living in an data-driven era, individuals are expected to comprehend complex phenomena in greater depth by consuming copious amounts of data. As such, making sense of data has become paramount (Lu et al., 2017). However, due to the exploration-enrichment-exploitation trade-off within the foraging loop, the sheer volume of data can sometimes complicate the process of targeting and analyzing objective information via the internet (Pirolli and Card, n.d.). This is particularly evident when individuals engage in tasks that require intensive searching, such as selecting a camera that best suits their needs or conducting literature research for a machine learning model. People tend to become lost in a maze of browser tabs with inconspicuous titles and, ultimately, become discouraged and abandon the task. These challenges underscore the need for advanced analysis techniques and methods that can unveil the underlying essence of the data and simplify the workload for users involved in sensemaking tasks.

## 1.2 Aims and Objectives

The key objectives of the project are:

1. To build an interactive graphical visual analytic system aiding users to extract and classify data from resources, and enabling users to apply exploratory data analysis. Information should be presented in a format matching the tasks and easy to updated.
2. To implement a model for the foraging loop. It should be capable of understanding or predicting the user's intention given past clickstream data. Considering the accessibility and the representativeness of the training dataset, public datasets relating to e-commerce and academic areas will be taken into consideration. If none of these are accessible or usable, an example dataset will be built.
3. To introduce more automation and support into the sensemaking loop. Observing the similarity in the working pattern in the sensemaking loop and that of the recommender system, work could be transformed to implementing a recommender

system, partially mirroring the function. Due to the intelligence-involved nature of the recommender system, the introduction of automation can be achieved and facilitated by adding machine learning methods instead of using only traditional rating methods (Hossain et al., 2022).

4. To integrate the whole system as a browser extension.
5. To evaluate the project. This can be done by asking the users to perform the same sensemaking tasks with and without the extension and collecting their user experience and feedback. Ideally, the working process should seem more fluent with a decrease in the number of repeat movements, such as repeatedly clicking into the tabs to see contents when dozens of tabs exist, and less time consumed to find target products. Metrics on measuring such activities need to be defined.

## 1.3 Description of the work

After discussion, three use cases are specified as the starting point of the project.

- Buying a laptop that is suitable for your major/occupation.
- Planning a trip in summer vacation to Japan/France/Turkey/Brazil
- Select three graduate schools that you intent to apply for

### 1.3.1 Functional Requirements

- The result product should be a chrome extension and should be able to use as other normal extensions.
- The extension has a GUI/visualization window to enable users' exploratory data analysis.
- The extension should have functions or buttons enable users to enter, update and delete the data.

- The extension should be able to collect information from certain websites based on users needs.
- The extension should be able to process the data and generate a visualization method.
- The extension should be able to give customized recommendation based on the clickstream data received.
- The language required for visualization part is JavaScript because it will be implemented as a function in the chrome extension.

### 1.3.2 Non-functional Requirements

The following requirements describe the metrics that will be evaluated to ensure the quality of the project.

- **Accessibility:** The key idea is to build an extension to assist users in different daily sensemaking tasks. This requires the extension and all its functions to be accessible in normal chrome browser environment.
- **Consistency:** As the output of the project will be an extension, users tend to use the extension in a plug-and-play style, requiring the operation method and underlying principle of the extension should be consistent with normal usage.
- **Usability:** To assure the usability, this demands that the design and implementation of the project should be fully considered, especially in the visualization part. Human computer interaction principles may be applied to build a user-friendly interface.
- **Efficiency:** One of key concept is to assist users in sensemaking tasks, this means that the efficiency using such tool to complete the task should surpass that doing manually. The data storage method, the running speed of the prediction/recommender models should be taken into account to prevent occupying excessive resources.



- **Performance:** Another significant concern is the performance of the model. The model should be trained to be empowered to understand/predict users' behavior properly. This requires a meticulous preparation of the training dataset and a fine tuning of the parameters inside models.

# Chapter 2

## Background and Related Work

### 2.1 Visualization

Thompson and Zhang have proposed a potential solution for the visualization of data relationship by using animation and bubble trees in a spatial layout [13, 15]. This approach allows users to interact with the graphical user interface (GUI) and gain a deeper insight into the information. Furthermore, Nguyen [10] introduced SenseMap and SensePath in 2016, which record and display users' activity history in a hierarchical manner that is clear to view and analyze. These tools are implemented as chrome extensions and can be invoked through pop-up windows, making them easily accessible for users.

### 2.2 Foraging Loop

#### 2.2.1 Visual Summarization

To enable users to gain a comprehensive understanding of the information they have encountered, visual summarization techniques offer several approaches, including explicit summarization, implicit summarization, and clustering. Explicit summarization generates grouping results directly from the raw data, which facilitates clear correspondence. [8] introduced an interactive visualization tool that assists game players in analyzing their behavior, presenting events in temporal order and allowing experts to uncover behavior

patterns related to actions and timing when facing different solution complexities.

In contrast, inexplicit summarization may use data mining techniques to reveal common structures. [1] utilized the Minimum Description Length Principle, a combination of data mining techniques and querying, to capture important patterns from event sequence data while compressing the loss. However, these methods tend to sacrifice low-level details and focus more on simple aspects of users' behavior, rendering them incapable of handling more sophisticated situations. Furthermore, the output of these models is typically a visualization of raw clickstream data, which creates a barrier to interpretation and understanding.

To provide a more comprehensive insight into the data, clustering methods inspect sequence-wide correlations. Wang proposed an unsupervised model that partitions clickstream data into a similarity graph, compares the normalized polar distance between two sequences, and presents captured patterns in an intuitive way [14]. However, unsupervised learning also suggests instability and unpredictability to some extent. Our project aims to stabilize this process by integrating more human intelligence during pruning and training.

### 2.2.2 Pattern Recognition & Sequence Prediction

Pattern recognition and matching are crucial prerequisites for understanding user behavior. Various models have been developed and applied to data to uncover sequence patterns in the field of user analysis and recommendation. These methods range from traditional sequential pattern mining and Markov chain approaches to advanced deep neural networks, which can be mainly divided into three categories: RNNs, CNNs, and Attention [16]. According to [6] and [2], RNN models and their relatives have dominant usage in the field. The paper listed possible time series behavior and compared different RNN cell structures to provide insight into their suitable tasks, respectively. Similar to the development path of other neural networks, pattern recognition can also be facilitated by constructing deeper neural networks, such as Deep RNNs [3]. The processing or pre-processing of raw sequence data also plays a paramount role in the tasks and has various approaches. Normally, most data is processed using certain encoders, such as one-hot

encoders. However, novel approaches, including lexical encoding and transferring scalar data into graphic features in higher-dimensional space, have been proposed to discover a more concise representation in higher-dimensional space while retaining as much low-level information as possible [5].

To address the specific research question of the project, the next-website prediction based on clickstream data (URLs), the data can be narrowed down to URLs. In [7], the authors presented the disadvantages of the current popular representation of URLs, the lexical encoding scheme, and introduced a better method by utilizing word embedding and deep CNNs. [9] moved a step further by incorporating transformers, including Bidirectional Encoder Representations from Transformers (BERT) and its variant RoBERTa, with normal tokenization, to successfully show the potential in this direction.

### 2.2.3 Sensemaking Loop

In order to enhance the degree of automation in the sensemaking loop, the implementation of a recommender system seems to be a logical step given its similarities with other stages in the workflow. Recommender systems can be classified into three main categories: content-based, collaborative filtering, and hybrid. Content-based models evaluate the similarity between items based on the preferences expressed by users and recommend items with high similarity values. This type of recommender system was prevalent in the early stages of development; however, its limitations are evident. By focusing on consistency with users' past preferences, the models reduce the opportunities for users to explore different options and discover new items.

Collaborative filtering models shift the focus of attention to users' feedback on items, including ratings or browsing histories. The models aim to find comparable customers and calculate the similarity between their profiles, thereby forming a neighborhood of customers with similar interests and making recommendations based on this. However, this type of model is limited by the amount of information it collects; the models can only identify the best neighborhoods after receiving sufficient relevant information, which can require a significant amount of data.

Hybrid methods combine different recommender strategies to address the limitations of both content-based and collaborative filtering models [4]. Noted that as the essences of recommendation and prediction task are both raise new items based on past experience, the recommender system may be integrated as a function of the prediction model.

# Chapter 3

## Design

Based on the discussion with the supervisor and the knowledge gained from literature review, the project will be built into a chrome extension, with other modules integrated as buttons or functions, and can be invoked from the interface.

### 3.1 Chrome Extension

The idea of building this project as a chrome extension has several reasons. Firstly, though sensemaking tasks happen everywhere, it is much easier to provide support via the Internet than in reality. Considering the barrier of usage in software, not only is that software's code and architecture needs to be modified to adapt to different platforms, it is also because that users will need to switch between browsers and the software frequently during the sensemaking tasks, adding their workload and compress the efficiency, which is just the opposite of the initial principle of the project. Reversely, the chrome extension is closely built inside the browser and can be used in a plug-and-play way, ensuring the continuous work flow and even enabling multitasking. It is also favoured because it can conveniently get access to the plentiful APIs in Google, saving much work.

The interface of the extension is built using Vue, which is a progressive JavaScript framework, emphasizing simplicity and ease of use, compressing the learning time. Comparing to React and Angular, the syntax and APIs of vue are designed to be more intuitive and simplify users work. It can also be integrated into existing projects, which fits the

need for frequent modifying and updating. Since the extension need to scrape data from the website and process the data to generate an output, Vue's efficient virtual DOM implementation is another merit.

Inside the interface, several functions were integrated as buttons. These functions include 'Add School', 'Add Criteria', 'Load Data From Site', 'Update' and 'Recommend', which will be introduced in detailed in the **Implementation** chapter.

The extension, consisting of three parts, 'Table', 'Radar Chart' and 'RNN recommendation', is designed for tasks that involve information collection and comparisons. It enable users to easily collect information and make comparison of different items based on the customized criteria. During the process, custom recommendations are given to broaden their choices. The working flow of the extension can be described as follows:

*The user click on the icon and open the extension. The user enter school names and criteria names into the table and click the load button, telling the extension to load information from the Internet. This will trigger a web scrapper to scrape data from predefined websties, e.g. 'www.usnews.com' and another function will fetch the results and fill them into the tables. After getting the information, the user can visualize the information in a radar chart. Scrolling down the window, the user can get some recommendation sites after a certain amount of events performed.*

Theoretically, the first part, the 'Information Table' part is designed to provide support to the information forage loop, as it help users to load or extract considerable amount of information from a website without any demanding of memory load, coding skills and searching techniques. The second part is a radar chart, provided for a basic exploratory data analysis with customized criteria, enabling users to view the disparities between items intuitively. In terms of the **Recommend**' button in the 'RNN' part, this recommender function seems to be excessive. Though might not being useful at most of the times, these related interesting choices may act as innovations when users mind are retard after length work.

By leveraging all these simple yet useful functions, users are able to gain a great understanding and knowledge of the data from multiple views, which thus support users'

decision making process. Additionally, as all the functions are general and not limited to a certain scope of usage, the extension therefore become versatile and can handle copious tasks.

## 3.2 RNN

As mentioned in the related work, RNN model is one of the most powerful and widely-used tools for sequence recognition and predictions. Another reason for choosing RNN lies in its popularity and which thus leads to a greater number of related work and potential community support that can be found. Details of the model construction will be mentioned in the following 'Implementation' chapter. The next-website prediction/recommendation task can be viewed as a multi-class classification problem. As the website URL will be converted into a binary vector, the task is to predict either 1 or 0 for each class.

### 3.2.1 Web Scrappers

Among all the libraries to build web scrappers, 'scrapy' in python is chosen for its scraping speed and the build-in support for parsing. The developer is also more familiar with python than JavaScript. The path used to extract information is chosen as 'Xpath', which is more powerful compiring to 'CSS selector' and having more flexibility than using 'regular expression' or string parsing.



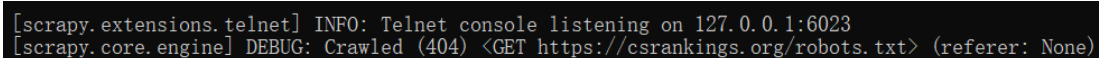
# Chapter 4

## Implementation

Unlike previous section, this part will introduce the implementation of each part of the project in a bottom-up way, aiming to maximized the fluency and understanding in reading.

### 4.1 Web Scrapper

The web scrapper designed to integrated into the project is the 'usnew scrapper' built by [11]. The GitHub repository for this is: GitHub. By design, this scrapper will be modified and exposed to the webserver using Flask. Then the Vue project can utilize the 'axios' library to make a 'POST' request to the endpoint created by the scrapper file and pass the 'URL' as parameter, and ultimately get the path of the output file. Comparing to the 'usnew scrapper', we also defined spiders written using Python 'scrapy' library, including scrapers for Amazons and usnews, but were less robust. They ran fine in the command line window but received IP address blocking problems in [Figure 4.1], and that is the reason why we turned to resources online.



```
[scrapy.extensions.telnet] INFO: Telnet console listening on 127.0.0.1:6023
[scrapy.core.engine] DEBUG: Crawled (404) <GET https://csrankings.org/robots.txt> (referer: None)
```

Figure 4.1: IP address block

## 4.2 Dataset

### 4.2.1 Preparation

Most machine learning models are data-driven, so the quality of the dataset significantly affects the model's performance. Large companies like Amazon have published open-source datasets, such as the clickstream dataset from Black Friday, which were initially expected to be readily available on the internet. However, these datasets are often inaccessible or charged, making them impractical. Despite numerous attempts to find free datasets, many of them lack clear structure and adequate information about the data. For example, the 'Data ClickStream Banco Galicia 2019' clickstream dataset on Kaggle is free but lacks proper documentation, making it challenging to understand the content of users' actions and the meaning of each feature. This lack of basic understanding of the dataset hinders the cleaning and processing of the data. As a result, we decided to collect our own clickstream dataset using the browser extension, 'Sensepath.' This extension helps users track their research and generate '.json' files of browsing history and events. We designed three tasks for users to complete, which were later reduced to one considering the time cost for each user to complete the task. Participants were asked to select three graduate schools they intend to apply to using the English version of Google Chrome. To complete the task, users needed to install and enable the 'SensePath' extension on their browsers and conduct normal search processes. Once users made their decisions, they listed them in a text file and briefly explained their reasons for choosing them. Users then clicked the 'save' button in the 'SensePath' interface and downloaded a json file to their local computer. They were required to send both the text file and json file to us via email and were rewarded for their participation.

To ensure adherence to computer science community standards, we anonymized all files before any analysis. The json file was used to create the dataset, while the text file served as auxiliary data for analyzing the correlation between users' behaviors and their results.

### 4.2.2 Email

To contact and recruit more participants, and to protect users' privacy, we decide to use emails as communication tools. The email content is as follows: *Hi, As part of our year3 project. We are developing a visual analytic chrome extension in the hope of assisting users in different sensemaking tasks. We have created some simple tasks for users to complete and which should take about 30 minutes. Please email me if you are interested. Rewards will be distributed after finishing the task. Thanks in advance, xxx*

#### User Instruction

In the email, we will provide the following instruction for users. [Figure 4.2]

Hi, <sup>42</sup>

As part of my 3<sup>rd</sup> year project, I'm planning to develop visual analytical tools for sensemaking tasks. I am looking for people to participate and give usable clickstream data on browsers. It should take about 30 minutes to complete all the tasks. <sup>43</sup>

There will be one task for you to complete, and should be performed on the Chrome browser due to technical requirements. The task is listed as follows:<sup>44</sup>

- Select 3 graduate schools that you intend to apply for.<sup>45</sup>

<sup>46</sup>

These are just topics to motivate you on starting a "sensemaking task", specific demands or limitations are up to you decide. For example, you have a ¥8000 budget limit on purchasing a laptop, or you only accept business class plane tickets or above, then of course you can apply a filter to only choose from products that meet the requirements. <sup>47</sup>

Please note, you are not required to reach a perfect final result or decision on the tasks given. It is the process that matters.<sup>48</sup>

Follow the steps to complete tasks:<sup>49</sup>

- 1) Open [this link](#) in Chrome browser, click "add to Chrome", then confirm adding "SensePath" to your browser. It should appear on the Extensions icon on the upper-right corner of your browser.<sup>50</sup>
- 2) Start using SensePath by clicking on it, a new window should pop up. Once you click on "Record", You are free to do all the searches on any website you like, and these actions you do will automatically be recorded by SensePath.<sup>51</sup>
- 3) Any text you select will be highlighted and noted by SensePath. If you did it by accident or you want to delete it, just edit it in the SensePath window.<sup>52</sup>
- 4) When you finish a task, click "Stop". A json file will be automatically generated. <sup>53</sup>
- 5) Please write down your final selection in a text file and briefly state your reason for choosing them in several sentences. Save it as 'Text.txt'.<sup>54</sup>
- 6) Please make sure you save and locate the json file and send both files to us for further analysis. Your private information will not be collected during the process, and the information will be de-identified.<sup>55</sup>

Thanks in advance,<sup>56</sup>

xxx<sup>57</sup>

Figure 4.2: Instructions for data collection

### 4.2.3 Target participants & Results

To ensure the representativeness of the dataset of common users, it is important to recruit participants from different ages, genders, and backgrounds. However, given our limited access to potential participants, we primarily recruited university students and staff, which may have restricted the age and background diversity of our sample. Nevertheless, we

endeavored to balance other features of the group, such as gender distribution and major distribution across the dataset.

In total, we received 30 responses from participants with diverse majors ranging from Computer Science to Applied Mathematics, most of whom were final year students. As graduate school searching and filtering is often part of their daily routine, they were expected to be well-suited for the task. Although we initially planned to provide rewards to participants upon completion of the tasks, this plan was disrupted due to unforeseen reasons in our research funding application. We apologized and consulted with all the participants, and received their understanding.

After reviewing and pruning the collected data, we were left with a final dataset consisting of 26 pieces of data. The structure of the data is shown in Figure 4.3

```
{
  "startRecordingTime": "2023-03-02T19:01:26.355Z",
  "data": [
    {
      "time": "2023-03-02T19:01:49.014Z",
      "url": "https://csrankings.org/",
      "text": "CSRankings: Computer Science Rankings",
      "type": "link",
      "id": 1677783789014,
      "endTime": "2023-03-02T19:01:50.014Z",
      "zoomLevel": 2,
      "customTranscript": "1 seconds spent in browsing 'CSRankings: Computer Science Rankings'",
      "transcript": "[01:00:22 - 01:00:23] (link follow) \n1 seconds spent in browsing 'CSRankings: Computer Science Rankings'"
    },
    {
      "time": "2023-03-02T19:01:59.759Z",
      "url": "https://csrankings.org/#/fromyear/2012/toyear/2022/index?all&us",
      "text": "Carnegie Mellon University",
      "faviconUrl": "https://csrankings.org/favicon.ico",
      "id": 1677783719759,
      "path": "/HTML[1]/BODY[1]/DIV[5]/FORM[1]/DIV[1]/DIV[2]/DIV[2]/DIV[1]/DIV[1]/TABLE[1]/TBOODY[1]/TR[1]/TD[1]",
      "className": "cm-1677783719756",
      "type": "highlight",
      "zoomLevel": 2,
      "level": 1,
      "customTranscript": "At second 11, highlighted 'Carnegie Mellon University'",
      "transcript": "[01:00:33] (highlight) \nAt second 11, highlighted 'Carnegie Mellon University'"
    }
  ]
}
```

Figure 4.3: Data structure

The data contains two parts, the start recording time, indicating the time that users open the *'SensePath'* extension, which will not be used in the models. The browsing history and events are stored in the *data* part, containing following elements. The actual number of elements shown varies depend on situations :

- ***'time'***: starting time of this event
- ***'url'***: current URL of the event
- ***'text'***: title of the website or event
- ***'type'***: 8 predefined types that will be assigned based the events taken, including:

'link', 'revisit', 'highlight', 'search', 'filter', 'unknown', 'type' and 'location'.

- **'id'**: a unique number to identify each event
- **'path'**: path indicating the element being highlighted in the page, only shown after highlight action
- **'classId'**: a unique number to identify highlight event, only shown after highlight action
- **'endTime'**: the ending time of this event, however it seems this function from 'SensePath' is not working correctly, and always return the time one second after the 'time' element.
- **'zoomLevel'**: the zoom in/out level of the page
- **'from'**: indicating the link id where current page is jumped from
- **'Level'**: highlight level of page, occurs only when user highlight something on the page. Only shown after highlight action
- **'customTranscript'**: a summary of the action
- **'transcript'**: a summary of the event, including the 'time' and 'endTime'.

## 4.3 RNN Model

To utilize the mainstream libraries and existing frameworks, Python and VScode were designed as the develop language and the platform for operating.

### 4.3.1 Initial Attempt

Firstly, all libraries are imported to build the environment for developing.[Figure 4.4]

Then, data need to be extracted from the json files. [Figure 4.5]

In the code, a clickstream list is built. Inside, each click represents an event in the dataset and is wrapped in a dictionary. To convert the **'time'** element into a format suitable

```
import numpy as np
import os
import pandas as pd
import json
import random
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from tensorflow import keras
from keras.models import Sequential, load_model
from keras.layers import LSTM, Dense, Dropout, Bidirectional, SimpleRNN
```

Figure 4.4: Required libraries for development

```
# Extract data
clickstream = []
for item in data:
    # Extract information
    time = datetime.fromisoformat(item['time'].replace('Z', '+00:00'))
    url_parts = item['url'].split('/')
    domain = url_parts[2]
    highlighted = False
    if('path' in item):
        highlighted = True

    # Create a dictionary with the preprocessed data
    click = {
        'time': time,
        'duration': 0,
        'domain': domain,
        'type': item['type'],
        'text': item['text'],
        'highlighted': highlighted
    }

    # Append the dictionary to the clickstream list
    clickstream.append(click)

for i, click in enumerate(clickstream):
    if(i < len(clickstream)-1):
        click['duration'] = clickstream[i+1]['time'] - clickstream[i]['time']
        click['duration'] = click['duration'].total_seconds()
        # we assume users only highlight on webpages, not on the browser page
        if(click['type'] == 'highlight'):
            click['type'] = 'link'
```

h

Figure 4.5: Initial attempt

for presenting and calculation, it is formatted using the *'fromisoformat()'* method in the *datetime* library. The *duration* of each event is added and calculated using the difference of starting time between the next event and current event, which is based on the assumption that users conduct activities continuously. The reason for having *'duration'* element lies in the potential connection of the significance of the event and the time spend on it. Additionally, some types in the data are changed into two types: 'link' and 'search', while concurrently adding a 'highlight' element to the click. This action proved to be excessive and can be attributed to the insufficient familiarity to the information correlating to each item in the data.

The developing process of the first attempt was cut here, however, because of finding a better way for data extraction and putting all them together.

### 4.3.2 Main Development Loop

#### Data Extraction

The required libraries are the same as mentioned above. The following steps varies. **Dataframe** object from the **Pandas** library is used to extract data from the json files. 'Duration' element is calculated using the same equation. An addition duration with random value ranging from 5 seconds to 20 seconds is added to the last element of a file to ensure that all the data samples having the same number of features. A new column, 'index' is inserted as to keep the index for each file. Excessive columns are dropped. The code is shown in Figure 4.6

After the extraction, we can get a dataset with 1176 samples and 4 columns. [Figure 4.7]

```
dfs = pd.DataFrame()
for i in range(1,27):
    filename = f"rawdata/{i}.json"
    if os.path.exists(filename):
        with open(filename, "r") as f:
            data = json.load(f)
            df = pd.json_normalize(data.get('data'))

            # Add duration by using next time - this time
            df['time'] = pd.to_datetime(df['time'])
            df['duration'] = df['time'].diff().apply(lambda x: x.total_seconds())
            df['duration'] = df['duration'].shift(periods=-1)
            df.at[df.index[-1], 'duration'] = random.randint(5,20)

            # Drop unnecessary columns
            df = df[['url','type', 'duration']]
            df = df.reset_index()
            dfs = pd.concat([dfs,df],axis=0, ignore_index=True)
```

Figure 4.6: Data extraction

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1176 entries, 0 to 1175
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  -
0   index       1176 non-null   int64
1   url         1176 non-null   object
2   type        1176 non-null   object
3   duration    1176 non-null   float64
dtypes: float64(1), int64(1), object(2)
memory usage: 36.9+ KB
None
```

	index	url	type	duration
0	0	https://www.google.com/search?q=umich+biostat...	search	7.302
1	1	https://sph.umich.edu/biostat/	link	6.700
2	2	https://sph.umich.edu/biostat/programs/masters...	link	5.986
3	3	https://sph.umich.edu/biostat/apply-ms-biostat...	link	13.079
4	4	https://rackham.umich.edu/admissions/applying/	link	8.930

Figure 4.7: Dataframe and its information

## Preprocessing

Seeing that the elements' type in 'type' and 'url' columns are both strings, encoding methods need to be applied to transfer them into numerical features. Utilizing the **LabelEncoder** and **OneHotEncoder** from the scikit-learn library, elements in column 'type' is encoded into ordinal numbers while urls are converted into a binary vector where all values are zero except for the value being encoded, which is represented as '1'. We then dropped the 'url' column since its features were already extracted into the binary vectors. This process is shown in Figure 4.8. The first five rows of the result of encoding is shown in [Figure 4.9].

```
# Apply label encoding to type and one-hot encoding to urls
le = LabelEncoder()
oe = OneHotEncoder()
dfs['type'] = le.fit_transform(dfs['type'])

onehot = oe.fit_transform(np.array(dfs['url']).reshape(-1,1))
onehot_df = pd.DataFrame.sparse.from_spmatrix(onehot)
# onehot_df
dfs_encoded = pd.concat([dfs, onehot_df], axis=1)
dfs_encoded.drop('url', axis=1, inplace=True)
dfs_encoded.head()
```

Figure 4.8: Label encoding and onehot encoding

index	type	duration	0	1	2	3	4	5	6	...	646	647	648	649	650	651	652	653	654	655
0	0	5	7.302	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	1	2	6.700	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	2	2	5.986	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	3	2	13.079	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	4	2	8.930	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 4.9: Encoding Result

## Getting time sequences & Splitting

Subsequently, the data need to be processed in to time sequence, which is the input format accepted by RNN models. By design, each sequence is consists of 5 time steps, and each time step is represented as an event. That is to say, for each sequence, the input will be 5 events, and the output will be the next event after. In the code, 'X' refers to input, and 'y' refers to output.[Figure 4.10]]



```
# Convert data into time sequences, 1 events forms a time step
samples = []
X_sequences = []
y_sequences = []
start_index = 0
for row_index in range(len(dfs_encoded)-1):
    if (dfs_encoded.iloc[row_index+1]['index'] == 0 or row_index+1 == (len(dfs_encoded)-1)):
        samples = dfs_encoded.iloc[start_index:row_index+2, 1:].values.tolist()
        for i in range(5, len(samples)):
            time_step = samples[i-5:i]
            X_sequences.append(time_step)
            y_sequences.append(samples[i][2:])
            i+=1
        start_index = row_index+2
```

Figure 4.10: Getting Time Sequence

By applying this, we get an input and an output sequence, both of 1046 samples. The real output will not be shown considering the large space it will consume. Suppose that 'X' indicates a time step(event), 'a', 'b' and 'c' each represent a feature of the time step(event), 'y' refers to the output event. The example format of a row combining the input and output sequence separated with '|' will be:

$$(X_1, X_2, X_3, X_5, X_5)|y$$

Each time step inside both sequences with three features will be:

$$X = (X_a, X_b, X_c)$$

With the input and output sequences ready, we can do a train test split with a train:test = 8:2 ratio by using the 'train\_test\_split' function from 'sklearn.model.selection' package.

### Defining RNN model

At such, we are ready to operate on the RNN models. Leveraging the functions and methods provided by Tensorflow, the model is defined to have a sequential structure. The first layer of the model is implemented using the 'LSTM' cell, which is more robust in handling long term dependencies without experiencing a vanishing gradient problem that is associated with 'Simple RNN'. A 'Dropout' layer with a 'dropout.rate = 0.2' is inserted after to balance the learning process of the model aiming to compress overfitting. Specifically, during training, this layer will randomly drops out a fraction of the input neurons of the layer to improve the generalization performance. The sturcture of RNN

model can be found in Figure 4.11.

```
# RNN
model = Sequential()
model.add(SimpleRNN(64, return_sequences=True, input_shape=(5,658)))
# model.add(LSTM(32, return_sequences=True, input_shape=(5,658)))
# model.add(Dropout(0.2))
model.add(SimpleRNN(256, input_shape=(5,658)))
model.add(Dropout(0.2))
model.add(Dense(256, activation='relu'))
model.add(Dense(656, activation='sigmoid'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, batch_size=5, epochs=200, validation_data=(X_test, y_test))
print("Training.....")
model.save('model.json')
```

Figure 4.11: RNN model structure

As mentioned in the **Design** chapter, the problem is viewed as a multi-class classification problem, thus the output layer is implemented using a 'Dense' layer, also called a 'fully connected layer'. This layer has 656 classes, which is two classes less than the input sequence since the expected output is only the URL. Different from normal multi-class classification problem, the output is expected to be a binary vector with each of class falls in a probability between 0 and 1. Thus, 'sigmoid' function is selected as the activation function instead of 'softmax' function based on their output format. The model is compiled with the loss function of 'Categorical\_crossentropy' and 'accuracy' as the evaluation metric. To take advantage of the flexible learning process, 'adam' is favored due to its adaptive learning rate.

The model then starts learning from the training set with a bath\_size of 5 and 100 epochs. X\_test and y\_test are used as validation set to monitor whether the model starts overfitting. Visualization of the training process is provided in Figure 4.12. After training and fine-tuning, a template of the current model will be saved to 'model.json' file, for future development and can be portable to other parts of the project. The structrue of the saved model can be seen in Figure 4.13.

However, the discrepancy between the training accuracy and validation accuracy suggest that the model is overfitting. [Figure 4.14] Limited by the size of training data and the complexity of the model, the cross validation function was deployed hoping to mitigate the problem.

```

Epoch 3/200
168/168 [=====] - 11s 43ms/step - loss: 6.4975 - accuracy: 0.0084 - val_loss: 6.4994 - val_accuracy: 0.0143
Epoch 4/200
168/168 [=====] - 2s 10ms/step - loss: 6.2993 - accuracy: 0.0251 - val_loss: 6.7114 - val_accuracy: 0.0143
Epoch 5/200
168/168 [=====] - 2s 9ms/step - loss: 6.1224 - accuracy: 0.0275 - val_loss: 6.9504 - val_accuracy: 0.0143
Epoch 6/200
168/168 [=====] - 2s 10ms/step - loss: 5.9642 - accuracy: 0.0335 - val_loss: 7.2896 - val_accuracy: 0.0238
Epoch 7/200
168/168 [=====] - 2s 14ms/step - loss: 5.7618 - accuracy: 0.0371 - val_loss: 7.1579 - val_accuracy: 0.0238
Epoch 8/200
168/168 [=====] - 2s 11ms/step - loss: 5.6051 - accuracy: 0.0383 - val_loss: 7.0957 - val_accuracy: 0.0190
Epoch 9/200
168/168 [=====] - 2s 10ms/step - loss: 5.4368 - accuracy: 0.0407 - val_loss: 7.2028 - val_accuracy: 0.0238
Epoch 10/200
168/168 [=====] - 2s 10ms/step - loss: 5.2826 - accuracy: 0.0514 - val_loss: 6.9538 - val_accuracy: 0.0286
Epoch 11/200
168/168 [=====] - 2s 10ms/step - loss: 5.1481 - accuracy: 0.0526 - val_loss: 7.2879 - val_accuracy: 0.0333
Epoch 12/200
168/168 [=====] - 2s 10ms/step - loss: 4.9915 - accuracy: 0.0658 - val_loss: 7.4288 - val_accuracy: 0.0333

```

Figure 4.12: Training process

```

✓ model.json
> assets
> variables
≡ keras_metadata.pb
≡ saved_model.pb

```

Figure 4.13: saved model structure

### 4.3.3 Modification of preprocessing of URL

After seeing the slight improvement of using cross validation, another method is needed to solve the problem. Based on the assumption that the amount of training data is insufficient to support the number of classes, we decided to apply a new encoding scheme to the URLs instead of onehot encoding, aiming to reduce the number of features. Inspired by [9] and [7], we decided to introduce Natural Language Processing into the process and build a vocabulary of the URLs to apply lexical encoding. Firstly, new libraries and packages need to be imported. [Figure 4.15]

One of the newly introduced package, *Natural Language Toolkit*, which is also known as 'nltk', is a leading platform designed for python program to work with human languages. It provides copious libraries and programs for NLP tasks, including tokenization, tagging, sparsing, lemmatizing and etc. After running the import block, we can get the result of downloading as shown in the figure below. [Figure 4.16]

The extraction part remained the same. To convert the raw URLs data into useful learnable features, we incorporated the python *regular expression*, also called 're', with the nltk functions to preprocess the data. Firstly, using 're' expression to remove the

```

loss: 0.2153 - accuracy: 0.9593 - val_loss: 11.8988 - val_accuracy: 0.1286
loss: 0.1899 - accuracy: 0.9653 - val_loss: 11.8998 - val_accuracy: 0.1190
loss: 0.1924 - accuracy: 0.9713 - val_loss: 11.9904 - val_accuracy: 0.1190
loss: 0.1882 - accuracy: 0.9617 - val_loss: 11.9791 - val_accuracy: 0.1238
loss: 0.1992 - accuracy: 0.9617 - val_loss: 12.0220 - val_accuracy: 0.1095
loss: 0.1944 - accuracy: 0.9557 - val_loss: 12.0188 - val_accuracy: 0.1143
loss: 0.1950 - accuracy: 0.9605 - val_loss: 12.0594 - val_accuracy: 0.1190
loss: 0.1879 - accuracy: 0.9617 - val_loss: 11.9244 - val_accuracy: 0.1143

```

Figure 4.14: RNN performance

```

import re
import string
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('omw-1.4')

import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences

```

Figure 4.15: Additional libraries

domain names and protocols without actual meanings. Then we convert all the characters into lower case and empirically applied 're' with some customized parameters to remove meaningless characters based on a trial-and-error experiment. Later, the punctuation and stop words were removed using lambda functions with nltk packages. Tokenization would be applied to the data after all the steps completed, utilizing the 'split()' function instead of 'tokenize()' of nltk since the space has already been removed. Details can be seen in Figure 4.17.

The result of regex and tokenization is a list of strings that contains informative words that we believe can provide a better insight of the data. [Figure 4.18]

The number of unique words have been dramatically suppressed at this stage, but that is still space to move one step further. By viewing the results in detail, we found that there are still words can be minimized or reduced. For instance, 'apply', 'applying' and 'application' are different forms of the word 'apply' and provides similar meanings in the context of URLs, and thus they can be all replaced by 'apply'. This can be done in two ways, *stemming* and *lemmatizing*. In the code, lemmatization is applied since it leverages a morphological analysis to reduce the words to their root form while stemming

```
[nlk_data] Downloading package punkt to
[nltk_data] D:\Year3\Dissertation\data\nltk...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data] D:\Year3\Dissertation\data\nltk...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data] D:\Year3\Dissertation\data\nltk...
[nltk_data] Package averaged_perceptron_tagger is already up-to-
[nltk_data] date!
[nltk_data] Downloading package wordnet to
[nltk_data] D:\Year3\Dissertation\data\nltk...
[nltk_data] Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data] D:\Year3\Dissertation\data\nltk...
[nltk_data] Package omw-1.4 is already up-to-date!
```

Figure 4.16: Import result

```
punc=string.punctuation
stop_words = set(stopwords.words('english'))

dfs['re_url'] = dfs['url'].replace(regex=(r'http://: ', 'https://: ', 'www.: ',
'.edu':, '.org':, '.net':,
'.uk':, 'chrome':, 'sourceid':, 'utf':,
'and':, '.cn':))

for i in range(len(dfs['re_url'])):
    # Remove query parameters and special characters
    dfs.loc[i,'re_url'] = re.sub(r'\.com\b', ' ', dfs.loc[i,'re_url'])
    dfs.loc[i,'re_url'] = re.sub(r'\.ac\b', ' ', dfs.loc[i,'re_url'])
    dfs.loc[i,'re_url'] = re.sub(r'[\.\?!\/\&+=\-\%#]', ' ', dfs.loc[i,'re_url'])
    dfs.loc[i,'re_url'] = re.sub(r'\bin\b', ' ', dfs.loc[i,'re_url'])
    dfs.loc[i,'re_url'] = re.sub(r'\bthe\b', ' ', dfs.loc[i,'re_url'])
    dfs.loc[i,'re_url'] = re.sub(r'\bfor\b', ' ', dfs.loc[i,'re_url'])
    dfs.loc[i,'re_url'] = re.sub(r'\bhtml', ' ', dfs.loc[i,'re_url'])
    dfs.loc[i,'re_url'] = re.sub(r'\bphp', ' ', dfs.loc[i,'re_url'])
    dfs.loc[i,'re_url'] = re.sub(r'\be5\b', ' ', dfs.loc[i,'re_url'])

    dfs.loc[i,'re_url'] = re.sub(r'[A-Z]', lambda m: m.group(0).lower(), dfs.loc[i,'re_url'])
    dfs.loc[i,'re_url'] = re.sub(r'(<[a-z])([a-z]?[a-z])', ' ', dfs.loc[i,'re_url'])
    dfs.loc[i,'re_url'] = re.sub(r'\b\d+\b', ' ', dfs.loc[i,'re_url'])
    dfs.loc[i,'re_url'] = re.sub(r'\b\d+\b', ' ', dfs.loc[i,'re_url'])
    dfs.loc[i,'re_url'] = re.sub(r'\bw(1,2)\b', ' ', dfs.loc[i,'re_url'])
    dfs.loc[i,'re_url'] = re.sub(r'\but\b', ' ', dfs.loc[i,'re_url'])
    dfs.loc[i,'re_url'] = re.sub(r'[\w\s]', ' ', dfs.loc[i,'re_url'])
    dfs.loc[i,'re_url'] = re.sub(r'\bw(1,2)\d+\b', ' ', dfs.loc[i,'re_url'])

dfs['re_url'].apply(lambda x: [word for word in x if word not in punc])
dfs['re_url'].apply(lambda x: [word for word in x if word not in stop_words])
dfs['tok_url'] = dfs['re_url'].apply(lambda x: x.split())
```

Figure 4.17: regex and tokenization

does it naively by removing the suffixes. We need to first insert tags to the words determining their part of the speech using the 'pos.tag' library provided by 'nltk'. Next, the '*get\_wordnet\_pos*' function is written to work cooperately with the 'wordnet' library in 'nltk' to convert the tags of the words to their base form, e.g. changing '**vbn**', referring to verb in past participle tense, to '**v**', basic verb. Then, by utilizing the '*WordNetLemmatizer()*' function, all the words are converted to their base forms. The code and its result will be shown in Figure 4.19 and Figure 4.20.

```

0      [google, search, umich, biostatistics, master,...
1          [sph, umich, biostat]
2          [sph, umich, biostat, programs, masters]
3          [sph, umich, biostat, apply, biostatistics]
4          [rackham, umich, admissions, applying]
...
1171          [gradadmissions, stanford]
1172      [gradadmissions, stanford, explore, programs]
1173      [ealc, stanford, academics, applying, graduate...
1174      [gradadmissions, stanford, explore, programs]
1175      [ealc, stanford, academics, applying, graduate...
Name: tok_url, Length: 1176, dtype: object

```

Figure 4.18: regex&amp;tok result

```

# Lemmatization
from nltk import pos_tag
dfs['pos_tags'] = dfs['tok_url'].apply(pos_tag)

def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN

dfs['wordnet_pos'] = dfs['pos_tags'].apply(lambda x: [word, get_wordnet_pos(pos_tag) for (word, pos_tag) in x])

wnl = WordNetLemmatizer()
dfs['lemma'] = dfs['wordnet_pos'].apply(lambda x: [wnl.lemmatize(word, tag) for word, tag in x])
dfs.head()

```

Figure 4.19: Lemmatization

At this stage, we can build the vocabulary of all URLs based on the words they contains. Employing the *counter()* function in the 'collections' library, all the words are stored into a list and sorted based on their frequencies across the dataset. Finally, a vocabulary of 800 words is built [Figure 4.21] [Figure 4.22].

Now, we can apply encoding to the features inside the dataframe. 'Type' is still encoded with 'LabelEncoder()' while the URLs are encoded into sequences of numerical values using the vocabulary. To ensure all the samples having the same length, zero padding is applied using the '*pad\_sequences*' function from 'Tensorflow.keras.preprocessing'. Therefore, all URLs have a length of 22 features, which is far less than 656 classes using the onehot encoding scheme. [Figure 4.23].

## Problem encountered

The processes of creating time sequences, train test splitting are similar to that mentioned above. However, when trying to feed the data to RNN model, several problems occurred. First and foremost, aiming to retain as much information about the URL, the input data has the shape of (SampleNumber, timestep, featurenumbers). However, as to retain as much information of the site, URLs are input as one class, which is a list of numbers

index	url	type	duration	re_url	tok_url	pos_tags	wordnet_pos	lemmas
0	0	https://www.google.com/search?q=umich-biostat...	search	7.302	google, search, umich, biostatistics, master...	[google, NN], [search, NN], [umich, JJ], [biostatistics, NN]	[google, n], [search, n], [umich, a], [biostatistics, n]	[google, search, biostatistics, master...]
1	1	https://sph.umich.edu/biostat/	link	6.700	sph, umich, biostat	[sph, NN], [umich, NN], [biostat, NN]	[sph, n], [umich, a], [biostat, n]	[sph, umich, biostat]
2	2	https://sph.umich.edu/biostat/programs/masters...	link	5.986	sph, umich, biostat, programs, masters	[sph, NN], [umich, NN], [biostat, NN], [programs, NN], [masters, NN]	[sph, n], [umich, a], [biostat, n], [programs, n], [masters, n]	[sph, umich, biostat, program, master]
3	3	https://sph.umich.edu/biostat/apply-ms-biostat...	link	13.079	sph, umich, biostat, apply, biostatistics	[sph, NN], [umich, NN], [biostat, NN], [apply, NN], [biostatistics, NN]	[sph, n], [umich, a], [biostat, n], [apply, n], [biostatistics, n]	[sph, umich, biostat, apply, biostatistics]
4	4	https://rackham.umich.edu/admissions/applying/	link	8.930	rackham, umich, admissions, applying	[rackham, NN], [umich, NN], [admissions, NN], [applying, NN]	[rackham, n], [umich, a], [admissions, n], [applying, n]	[rackham, umich, admission, apply]

Figure 4.20: Lemmatization\_result

```

# Build the url vocabulary
from collections import Counter
vocabulary = {}
for row in df['lemmas']:
    vocabulary.extend(row)

vocabulary = [word.translate(str.maketrans("", "", string.punctuation)) for word in vocabulary]
vocabulary = [re.sub(r'\b(w{1,2})\b', '', word) for word in vocabulary]
vocabulary = [re.sub(r'\b(w{4,d+})\b', '', word) for word in vocabulary]
vocabulary = [re.sub(r'\b(d{4,w+})\b', '', word) for word in vocabulary]
vocabulary = list(filter(None, vocabulary))

freq_dist = Counter(vocabulary)
sorted_vocab = sorted(freq_dist.items(), key=lambda x: x[1], reverse=True)
vocab_size = 800
url_vocab = [token for token, freq in sorted_vocab[:vocab_size]]
url_vocab

```

Figure 4.21: URL vocab

representing the words inside. This format is not accepted for the standard RNN model in 'keras' library. After several attempts of transfer them into tensors failed, the URL had to be input as 22 classes, representing the 22 word digits. Since the words are in numerical forms, that is, similar to having 800 labels filling into 22 holes, it is hard to define the activation function for the last layer. Normally, classification output should be a probability distribution across classes, but in this context, no existing function can be best fit to format of output. The 'softmax' function is chosen at last, but all this compromising seem to counteract the advantage of the lexical encoding scheme and leads to a result far from expectation. The model performed slightly better, having an accuracy raised from '0.11' to '0.23' but was underfitting, to both the training data and the validation data. [Figure 4.24]

## 4.4 Interface

The Interface is build using Vue.js. Thanks to powerful functionality of Vue CLI, which is a command line tool for creating vue projects. By using it, we can easily create a vue project with essential configurations and package structures from scratch. This can be done using the '**vue create project-name**' in the windows powershell. After selecting the default built-in of vue 3 with bable and eslint and press enter, a brand new vue project

```
['science',
 'graduate',
 'university',
 'program',
 'computer',
 'search',
 'study',
 'postgraduate',
 'google',
 'course',
 'school',
 'degree',
 'academic',
 'master',
 'admission',
 'ranking',
 'aqs',
 'msc',
 'illinois',
 'top',
 'best',
```

Figure 4.22: URL vocab result

```
# Encoding
le = LabelEncoder()
dfs['type'] = le.fit_transform(dfs['type'])

# Encoding urls using the vocabulary and convert all the type into float32
word_to_label = {word: i for i, word in enumerate(url_vocab)}
dfs['encoded_urls'] = dfs['lemma'].apply(lambda x: [word_to_label[word] for word in x if word in url_vocab])

dfs_encoded = dfs[['index', 'type', 'duration', 'encoded_urls']]
dfs_encoded['type'] = dfs_encoded['type'].apply(lambda x: np.array(x, dtype=np.float32))
dfs_encoded = dfs_encoded[['index', 'type', 'duration']]

padded_urls = pad_sequences([x for x in dfs['encoded_urls']], maxlen=22, padding='post')
# padded_urls = list(padded_urls.astype('float32'))
df2 = pd.DataFrame(padded_urls, columns=[f'({i})' for i in range(1, 23)])
dfs_encoded = pd.concat([dfs_encoded, df2], axis=1)
padded_urls
```

Figure 4.23: New encoding

is created. Here is the initial structure of the empty project. [Figure 4.25]. Users can always keep an eye on their result in a browser view using the command 'vue run serve' after located in the vue project root directory.

The App.vue under the 'src' folder is the main body containing most of the information for the interface. Aiming to compress the complexity, we choose to implement the interface in a vue **Single File Component** format, which also known as 'sfc' file, consist of three sections, template, script and style, corresponding to HTML, JavaScript and CSS codes.

```
loss: 34208.9922 - accuracy: 0.2229 - val_loss: 31160.2402 - val_accuracy: 0.2326
loss: 34630.0508 - accuracy: 0.2229 - val_loss: 31357.4141 - val_accuracy: 0.2326
loss: 34711.3750 - accuracy: 0.2229 - val_loss: 31550.8750 - val_accuracy: 0.2326
loss: 34863.2539 - accuracy: 0.2229 - val_loss: 31757.4746 - val_accuracy: 0.2326
loss: 35376.7891 - accuracy: 0.2229 - val_loss: 31978.9316 - val_accuracy: 0.2326
loss: 35319.4453 - accuracy: 0.2229 - val_loss: 32163.0410 - val_accuracy: 0.2326
loss: 35647.7383 - accuracy: 0.2229 - val_loss: 32382.2227 - val_accuracy: 0.2326
loss: 35980.5078 - accuracy: 0.2229 - val_loss: 32623.9277 - val_accuracy: 0.2326
loss: 35951.1836 - accuracy: 0.2229 - val_loss: 32819.2695 - val_accuracy: 0.2326
```

Figure 4.24: lexical RNN performance



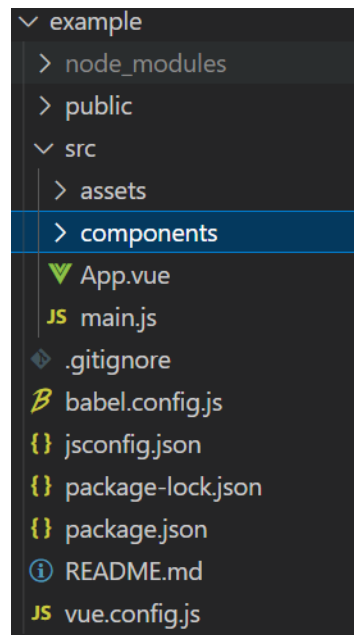


Figure 4.25: vue example structure

This leads to the convenience to writing and updating different parts in one file, without switching among three files.

#### 4.4.1 HTML

Starting with visual part, after having a basic idea of how our interface should look like, we first split the page into several parts, including 'Table', 'Radar Chart' and 'RNN model'. Then, functions and buttons were added to each section. '@', short for 'v-on', which is used to bind event listeners to a specific element, is used to enable buttons to listen to users input and update the values accordingly. The size of each element is also adjusted after several experiment. The code is provided in Figure 4.26 and Figure 4.27.

The result can be shown running the 'vue run serve' command and copy the link generated into the browser [Figure 4.28]. One of the merit to use vue CLI is that it enable you to manage project using command line. The content window will be automatically updated with the newest version of saved code as long as the users keep the command line window running, making it efficient to keep track on the project progress.

Noted that the result seems naive and monotonous with plain HTML, so scripts and styles need to be added.

```

<template>
  <div id="App" class="popup">
    <!-- Header -->
    <div class="header">
      <h1>School Comparison. The greatest Extension!!!</h1>
    </div>

    <!-- Criteria and School Buttons -->
    <div class="buttons">
      <button @click="addCriteria">Add Criteria</button>
      <button @click="addSchool">Add School</button>
    </div>

    <!-- Table of Schools -->
    <table>
      <thead>
        <tr>
          <th>School Name</th>
          <th v-for="(criteria, index) in criteria" :key="index">
            {{ criteria.name }}
          </th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="(school, index) in schools" :key="index">
          <td>{{ school.name }}</td>
          <td v-for="(criteria, index) in criteria" :key="index">
            {{ school[criteria.name] }}
          </td>
        </tr>
      </tbody>
    </table>
  </div>

```

Figure 4.26: APP.vue template1

```

<!-- Trying to Read excel file -->
<div class="buttons">
  <button @click="readData">ReadExcel</button>
</div>

<!-- Radar Chart -->
<div>
  <apexchart type="radar" height="350" :options="chartOptions" :series="series">Best Radar Chart</apexchart>
</div>
<div>
  <button @click="updateChart">Update!</button>
</div>

<!-- RNN Model Prediction -->
<div class="rnn-predict">
  <h2>RNN model</h2>
  <div id="rnn-predict">
    <button @click="predict">Predict!</button>
  </div>
</div>
</div>
</template>

```

Figure 4.27: APP.vue template2

## 4.4.2 Script

Here comes the Javascript part, which defines how the html elements should react to interactions. We need to first import all the libraries and plugins. When first running the code like `'import * as d3 from 'd3';'`, the browser window will generate compile errors show in [Figure 4.29], which is because these libraries are not found in the project directory. The way to solve this is to run `'npm install --save xxxxx'`, and the the vue CLI will install the latest version of these plugins and automatically add the configuration to the 'node modules' folder, which stores the dependencies.

After instantiating the component 'apexChart' from the 'VueApexCharts' library, we can start coding the 'data()' part, which encapsulates all the data objects that will be parsed

**School Comparison. The greatest Extension!!!**

School Name

Best Radar Chart

**RNN model**

Figure 4.28: HTML result

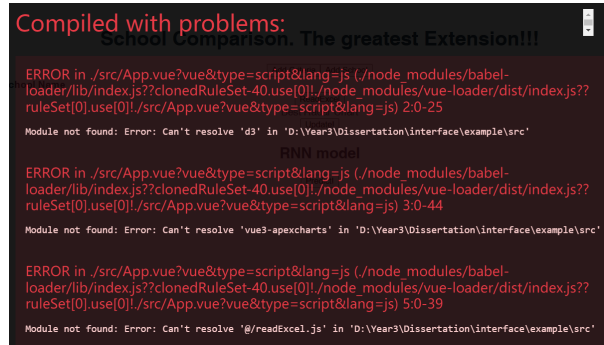


Figure 4.29: Compile Error

around functions. We initialize two empty lists: *schools[]* and *criteria[]* to store the information of schools and criteria input by users. Then the 'chartoptions', which is the initial parameter that will be passed to generate a radar chart, is set using example data. The purpose of the pre-setting is to check whether the radar chart will be generated, and it will be updated later with real information. The details of the 'component' and 'data' parts can be found in [Figure 4.30] and [Figure 4.31].

Having the data objects and radar chart presets, we still need to define the methods created in the HTML file. All the methods are designed as follows:

- **'addSchool'**: should be able to prompt users with a input window and check whether the input is empty when users press enter. Input will be added into the 'schools[ ]' list. Once there is an element in schools list, the table will be created, and new elements will be added to the first column of the table.[Figure 4.32]
- **'addCriterion'**: should be able to prompt users with a input window and check whether the input is empty when users press enter. Input will be added into the 'criteria[ ]' list. The elements in the list will be added to the table header.

```

<script>
import * as d3 from 'd3';
import VueApexCharts from "vue3-apexcharts";
// eslint-disable-next-line no-unused-vars
// import readExcel from '@readExcel.js';
export default {
  name: 'SchoolComparison',
  components:{
    apexchart: VueApexCharts
  },
  data() {
    return{
      cri_id: 0,
      sch_id: 0,
      criteria: [
        { index: cri_id++, name: 'Rankings' },
        { index: cri_id++, name: 'Tuition fee' },
        { index: cri_id++, name: 'Facilities' },
        { index: cri_id++, name: 'Faculties' },
      ],
      schools: [
        { index: sch_id++, name: 'Columbia' },
        { index: sch_id++, name: 'Harvard' },
        { index: sch_id++, name: 'UCLA' }
      ],
    }
  }
}

```

Figure 4.30: Script 1

- **'loadData'**: this method will first call the 'scrape' method and fetch the output excel file to the 'readExcel' method. It then put the information in the array into the table according to the given schools and criteria, and update the table.
- **'readExcel'**: this method will be read the information from '.xlsx' file into an array. It requires that the excel to have a header.
- **'updateChart'**: this method will update the radar chart based on the information in the table.
- **'recommendSite'**: this method will generate several websites basedon the users past events in the browsers. It will call the pretrained RNN model and get the prediction of next website URL as output.
- **'scrape'**: this method will invoke the 'usnews scrapper' and return the excel file containing the scrapped information.

```

chartOptions: {
  chart: {
    height: 350,
    type: 'radar',
    dropShadow: {enabled: true, blur: 1, left: 1, top: 1}
  },
  xaxis: {categories: ['2011','2012','2013','2014','2015','2016']},
  yaxis: {min: 0,max: 10,tickAmount: 5},
  title:{text: "Best Radar Chart"},
  stroke: {width: 1.5},
  fill: {opacity: 0.1},
  markers: [{size: 2.5}],
},
series: [{
  name: 'Series 1',
  data: [8, 5, 3, 4, 10, 2]},
  {name: 'Series 2',
  data: [2, 3, 4, 8, 2, 8]},
  {name: 'Series 3',
  data: [4, 7, 7, 1, 4, 10]}],
},
],

```

Figure 4.31: Script 2

```

addSchool() {
  const school = prompt("Enter School Name")
  if(school){
    const newschool = {
      name:school,
      id: this.sch_id++;
      this.schools.push(newschool)
    }
  },
},

```

Figure 4.32: Add school

```

addCriterion() {
  const criterion = prompt("Enter Criterion Name")
  if(criterion){
    const newcriterion = {
      name: criterion,
      id: this.cri_id++;
      this.criteria.push(newcriterion)
    }
  },
},

```

Figure 4.33: Add criterion

```

// Read data for selected schools
async loadData() {
  const rows = await readExcel('school_info.xls');
  console.log(rows);
},

```

Figure 4.34: Load data from site

```

import * as XLSX from 'xlsx/xlsx.mjs';

function readExcel(file) {
  return new Promise((resolve, reject) => {
    const reader = new FileReader();
    reader.onload = (e) => {
      const data = new Uint8Array(e.target.result);
      const workbook = XLSX.read(data, { type: 'array' });
      const sheetName = workbook.SheetNames[0];
      const worksheet = workbook.Sheets[sheetName];
      const rows = XLSX.utils.sheet_to_json(worksheet, { header: 1 });
      resolve(rows);
    };
    reader.onerror = (e) => {
      reject(e);
    };
    reader.readAsArrayBuffer(file);
  });
}

```

Figure 4.35: Read Excel to array

### 4.4.3 Style

At last, to enhance the aesthetic of the interface, css styles can be added. [Figure 4.36]

```
<style>
.popup{
  width: 500px;
  height: 800px;
}
table {
  border-collapse: collapse;
  width: 100%;
}
th, td {
  text-align: left;
  padding: 8px;
  border: 1px solid #b4b4b4;
}
th {
  background-color: #d8d8d8;
}
</style>
```

Figure 4.36: CSS

## 4.5 Chrome Extension

To create a chrome extension based on a vue project can be done efficiently by utilizing the a plugin developed by [12]. Here is the link to the GitHub repository: [GitHub](#). To installing this plugin into the project using command line tool: 'vue add chrome-extension-cli'. During the process, by entering text or selecting the choices, a vue project-based chrome extension can be built having customized configurations. An example questions and answer is provide in Figure 4.37.

```
❖ Installing vue-cli-plugin-chrome-extension-cli...

up to date, audited 997 packages in 2s
105 packages are looking for funding
  run npm fund for details

found 0 vulnerabilities
❑ Successfully installed plugin: vue-cli-plugin-chrome-extension-cli

? Name of the Chrome Extension? example
? Description for the Chrome Extension? no
? Version for the Chrome Extension? 5.0
? manifest_version for the Chrome Extension? 3
? delete Initial file? (src/main.js src/components) No
? Please select the required components : background, popup, content

❖ Invoking generator for vue-cli-plugin-chrome-extension-cli...
❑ Running completion hooks...

❑ Successfully invoked generator for plugin: vue-cli-plugin-chrome-extension-cli
create entry folder error
```

Figure 4.37: Example command window for chrome-ext cli

At such stage, though the project is equipped with all the frameworks and configurations,

it cannot be loaded into the chrome extension store because the whole project exceed the size limit. To develop a runnable chrome extension, we then need to use the 'npm run build-watch' command in the command line window, which build the project for deployment and generate a **dist** folder. The **dist** folder contains a distributable and minimized version of the web application that can be read by web servers. The current structure of the project should be something like this in [Figure 4.38].

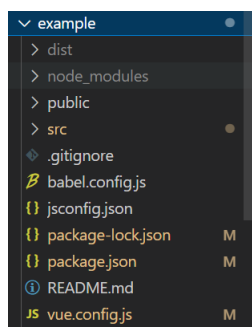


Figure 4.38: Example extension structure

We are now ready to deploy the extension into the chrome extension store. By clicking on the puzzle icon on the up right corner in the chrome browser and click 'manage extensions'. After ensuring the 'developer mode' button is switched on in blue color, we can then click on 'Load unpacked' button on the top left. A pop-up window will be shown. In the window, we select the 'dist' folder and successfully load the extension. [Figure 4.39]

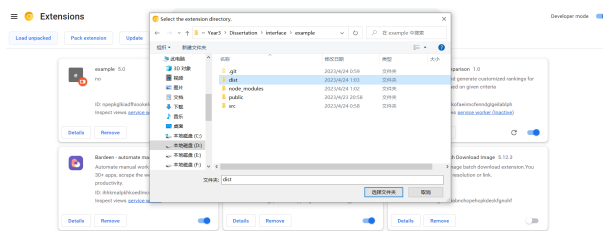


Figure 4.39: Loading extension into chrome

We should then see the extension with the name, version and its description appears at the first of the all extensions. To see whether the extension displays things as designed, we click on the 'puzzle' icon on the right and select our extension. To modify and update it, 'npm run built-watch' command is needed to generate a new 'dist' folder. Subsequently, the extension will be updated by clicking on the refresh button on its bottom right.

### 4.5.1 Extension Display

Here are some figures showing some example functions of the extension. The process simulates the user entering schools and criteria, and updating the radar chart with new information, which is shown in Figure 4.40

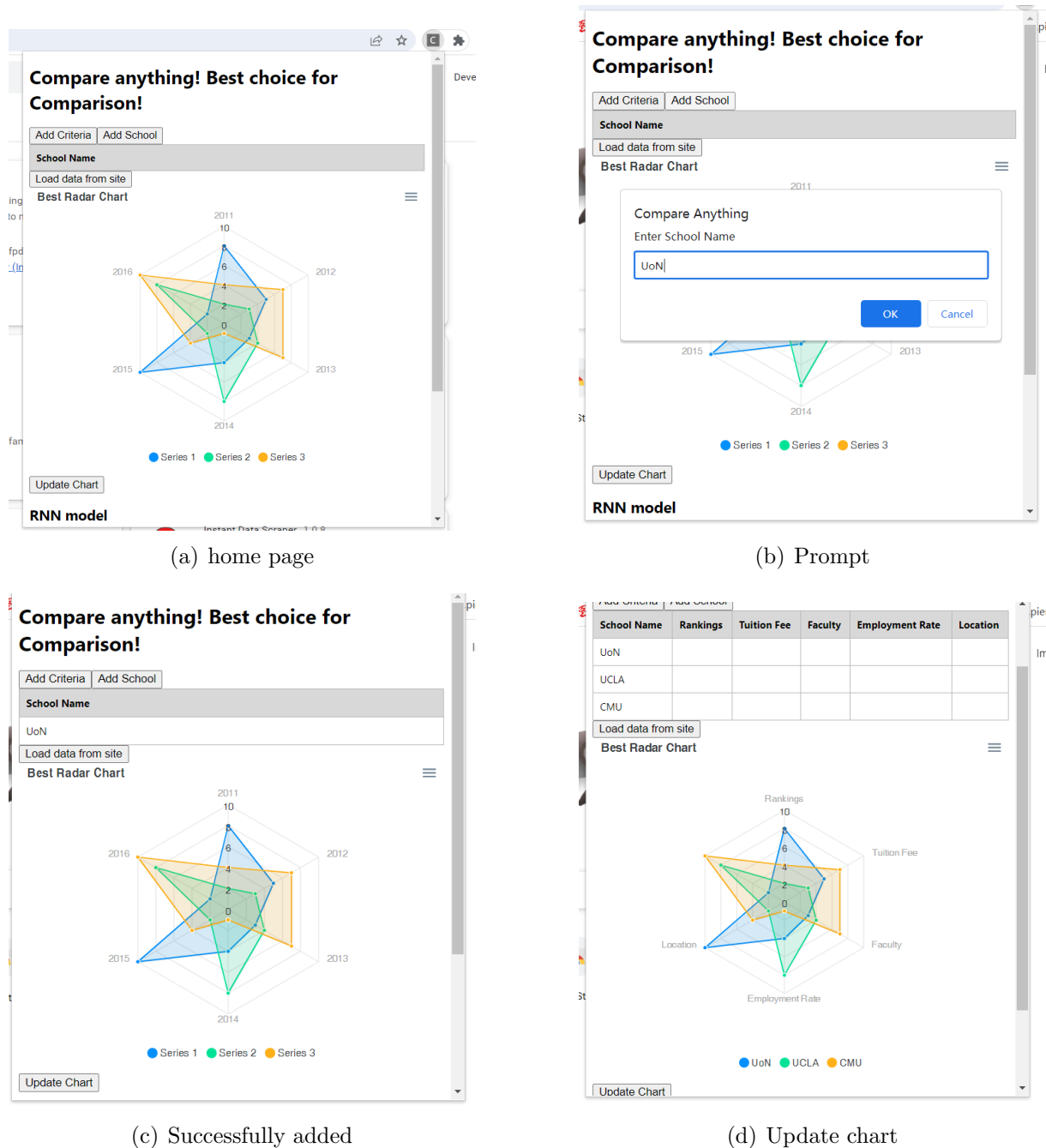


Figure 4.40: Using extension



### 4.5.2 Problem encountered

One of the biggest problem encountered is related to how to build a chrome extension based on a vue project. Initially, the work was started from scratch, however, due to the complex data interaction structures and massive number of configurations and frameworks, little progress has been made and that lead to the turning of the focus towards existing work and resources though out the internet. Several repositories have been found closely related and were regarded as the solutions. However, intensive effort has been made to figure out the correct path to a working project. The reason lies in the both the similarity and disparity between the tasks of building a website using vue and constructing an extension. They are alike in the type of work flow, both aiming to build an runnable pack of files to be accepted by servers, while concurrently, they are also diverse, with unique architecture and different specialized domains. Therefore, when attempts are made to take advantage of the experience of the other in building a certain project, it may be not applicable and hard to be recognized through debugging process. A considerable high level of understanding of both technologies is required when trying to adapt vue components into chrome extension structures, including configuring manifest files and handling browsers' API. For instance, the manifest file has two version, version 2 and 3, both of which are widely used. Errors relating to 'background\_service\_worker' are frequently triggered when loading extension to the browsers, since there is a migration in syntax and function from manifest v2 to v3.

# Chapter 5

## Evaluation

### 5.1 RNN

The RNN model is tested using the test set derived from the data collected. Cross validation was applied to the model using 'accuracy' as the scoring metric. One of paramount reason is that the amount of training data is far from adequate for the model to reach its optimal. Additionally, the way to preprocess and understand the collected data can be optimized to extract more learnable features.

### 5.2 Browser extension

A user evaluation questionnaire was supposed to made to evaluate the experience of using the result of the project, which is listed in the Appendix. By design, users are going to be asked to complete a regular sensemaking task in limited time, such as choosing a holiday destinations. They will then record their feelings and reviews on the extension though filling out the questions, ranging from user experience to aesthetic level, in the questionnaire. However, due to the suspension of the expected finish time, no evaluation has been made.

# Chapter 6

## Summary and Reflections

### 6.1 Project management

Throughout the year, agile method was used to manage the project. Work has been split into sprint of two or three weeks, with meetings with supervisors hosted every two weeks. A personal to-do list is used to record the progress. In last semester, the progress was delayed due to both the overoptimistic expectations and unforeseen reasons in the dataset collection process, which consequently leads to the suspension on the development on the machine learning model. For the visualization part, initial goals are reached by successfully building the prototypes and made great progress in learning JavaScript. Realizing the project is behind the schedule, a reallocation of work were conducted to the second semester. The proposal and the goal of the project were also adjusted and narrowed down to building a chrome extension that focus on supporting a specific aspect of sensemaking tasks first. Following the plan, web scrappers, nlp models, local storage and Indexed DB projects were built. Some of these assets were not presented or included in the final dissertation project, yet, through the developing process, better understanding and valuable insights of the project were gained. One of problem observed is that the lack of continuity in the work distributed in adjacent weeks, which entailed the time wasting on shifting between modules. It should be also noted that the work was not distributed wisely and the purposes for doing some of them were not clear at the time, increasing unnecessary workload. Overall, the project was slack at the front and speeded up towards the end.

The initial plan and goal proved to be overly ambitious, which can also be attributed to the insufficiency in literature review and the understanding of the topic. A lesson learned is to start building things early instead of taking much time looking back and forward, that is because the adjustment based on assumptions cannot match the value of the intrinsic need originated from real experience.

## 6.2 Contributions and reflections

Providing the details of your achievements and contributions including innovation, creativity and novelty (if there is any) as well as a personal reflection on the plan and your experience of the project (a critical appraisal of how the project went). The project was first splitted into three parts, the visualization, information foraging stage and sensemaking stage, and was later modified and narrowed down into two main components, the visualization and the behind models, which were more specific and executable. To understand users' intention and discover their behavior patterns, a RNN model is built using Python. Based on the accumulated knowledge and the inspiration from several papers, the encoding scheme of the URLs inside the data upgraded from normal onehot encoding to lexical encoding incorporated with NLP libraries. This upgrade represented the deepening comprehension of the topic, and the attempt to create new method to extract more practical information from data, though the final performance was not optimized. Several web scrappers were built utilizing the 'scrapy' library in Python. They ran fine and can scrape information from certain websites until after while were all confronted IP address blockage for unforeseen reasons and thus lead to the researching for available substitution online. A chrome extension was built. It has an interactive interface written using Vue.js and is portable and easy to update. The interface provides users functions to collect and visual information without a high demand in searching and memorizing skills and thus reduce the cognitive load. The functions and visualization of the extension enable users to do intensive comparison of data from customized aspects. Initially, more automation was designed to add to the project, that is, to utilize the convenience of web scrappers and the prediction ability of the RNN models. However, the scrappers are not closely con-

nected and hard to invoke while the RNN model suffers from a unsatisfied performance, which will be the future directions for the project. Overall, after working on the project, knowledge in specific areas is greatly widen and deepened, including web designing, web interactions, web scrapping and machine leaning(sequence pattern recognition).

### 6.2.1 The bigger picture

In terms of the bigger picture, as the technology of web scraping advance, related laws and ethical problems should be taken into consideration. On the one side, being able to scrape any desired data from the Internet signs the development of such technology and can dramatically increase convenience in many aspects. On the other side, to reach the goal of getting any information, it is highly possible to invade the intellectual property rights and privacy of both organization and individuals. Seeing this as a trade-off between two side, a balance need to be settle and related enforcement should not be blank. Another problem that needs to be concerned relates to the machine learning area. The areas of patterns matching and prediction, advancing rapidly, receive increasingly more commercial interest. As people nowadays can be recognized through the biometric data such as face recognition, it is potential that more pervasive and common patterns may be used as identification, e.g. walking patterns. This also requires the matched construction in laws and professional areas to protect people's rights.

# Bibliography

- [1] CHEN, Y., XU, P., AND REN, L. Sequence synopsis: Optimize visual summary of temporal event data. *IEEE transactions on visualization and computer graphics* 24, 1 (2017), 45–55.
- [2] CHU, Y., YANG, H.-K., AND PENG, W.-C. Predicting online user purchase behavior based on browsing history. In *2019 IEEE 35th international conference on data engineering workshops (ICDEW)* (2019), IEEE, pp. 185–192.
- [3] DUDUKCU, H. V., TASKIRAN, M., TASKIRAN, Z. G. C., AND YILDIRIM, T. Temporal convolutional networks with rnn approach for chaotic time series prediction. *Applied Soft Computing* 133 (2023), 109945.
- [4] HOSSAIN, I., PALASH, M. A. H., SEJUTY, A. T., TANJIM, N. A., NASIM, M., SAIF, S., AND SURAJ, A. B. A survey of recommender system techniques and the ecommerce domain. *arXiv preprint arXiv:2208.07399* (2022).
- [5] JASTRZEBSKA, A. Time series classification through visual pattern recognition. *Journal of King Saud University-Computer and Information Sciences* 34, 2 (2022), 134–142.
- [6] KHALDI, R., EL AFIA, A., CHIHEB, R., AND TABIK, S. What is the best rnn-cell structure to forecast each time series behavior? *Expert Systems with Applications* 215 (2023), 119140.

- [7] LE, H., PHAM, Q., SAHOO, D., AND HOI, S. C. Urlnet: Learning a url representation with deep learning for malicious url detection. *arXiv preprint arXiv:1802.03162* (2018).
- [8] LI, W., FUNK, M., LI, Q., AND BROMBACHER, A. Visualizing event sequence game data to understand player’s skill growth through behavior complexity. *Journal of Visualization* 22 (2019), 833–850.
- [9] MANERIKA, P., STOKES, J. W., LAZO, E. G., CARUTASU, D., TAJADDODI-ANFAR, F., AND GURURAJAN, A. Urltran: Improving phishing url detection using transformers. In *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)* (2021), IEEE, pp. 197–204.
- [10] NGUYEN, P. H., XU, K., WHEAT, A., WONG, B. W., ATTFIELD, S., AND FIELDS, B. Sensepath: Understanding the sensemaking process through analytic provenance. *IEEE transactions on visualization and computer graphics* 22, 1 (2015), 41–50.
- [11] OVROABIR. Ovroabir/usnews-scraper: This application collects grad schools data from <https://www.usnews.com> and gives output in an excel file.
- [12] SANYU1225. Sanyu1225/vue-cli-plugin-chrome-extension-cli: Use vue plugin easy create chrome extension template.
- [13] THOMPSON, J. R., LIU, Z., AND STASKO, J. Data animator: Authoring expressive animated data graphics. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (2021), pp. 1–18.
- [14] WANG, G., ZHANG, X., TANG, S., ZHENG, H., AND ZHAO, B. Y. Unsupervised clickstream clustering for user behavior analysis. In *Proceedings of the 2016 CHI conference on human factors in computing systems* (2016), pp. 225–236.
- [15] ZHANG, X., CHANDRASEGARAN, S., AND MA, K.-L. Conceptscope: Organizing and visualizing knowledge in documents based on domain ontology. In *Proceedings of the 2021 chi conference on human factors in computing systems* (2021), pp. 1–13.

- [16] ZHANG, Y., YIN, G., DONG, H., AND ZHANG, L. Attention-based frequency-aware multi-scale network for sequential recommendation. *Applied Soft Computing* 127 (2022), 109349.



# Appendix A

## User Manuals

## Appendix B

### User Evaluation Questionnaire