DSC180B Final Report

Langrun Zhu, Zeming Zhang

University of California, San Diego

Data Science Department, Halıcıoğlu Data Science Institute

DSC180B

Solutions of Foreground Window Recommendations

Abstract

Working alongside with the intel technology team, our group spent the first quarter and the winter break collecting many types of user activity data from a PC. We collected the movement data of a mouse or data which applications users opened. And we collected data of the foreground window duration, the class and the executive name of the application.With all this data, we can use them to make further predictions about which application the user will probably open at a certain time, and for how long on the desktop.

In this project, we first implemented the hidden markov model for predicting the next application based on the current application opened on the windows. To increase the accuracy, we predict the next few possible applications instead of just a single one. Furthermore, we implemented a Long short-term memory method based on the Recurrent neural network model to predict the total time an application is used in the foreground.

Keywords: User activity, foreground windows, HMM, LSTM.

1. Introduction

Nowadays, computers have become an inseparable part of our life, and most people spend a majority of their time working with computers. To successfully work on a computer, users require different applications to complete different jobs, such as doc file for paperwork, excel file for data processing, photoshop for image editing, etc. While people use their computers, most of the time they may face computer lag, slow response time. This is very frustrating, because it will reduce people's productivity, decrease the performance of the computers, and cause negative user experience. With the development of artificial intelligence, we are able to solve this problem and boost the efficiency of computers by letting the computer help me open the application and progress we need without telling the computer what we want. Before building models for training, first we have to collect user PC data for further data analysis.

In the first quarter, to collect the user PC data, we used different input libraries based on the Intel XLSDK toolkit to collect data from our PC. After the collection during the winter break and first quarter, we now have a handful of data to perform data analysis and solve the problems we address during our proposal. Currently we have the data for a user for about two months, mostly focusing on the application on foreground windows, including time, application name, class, and executive name of the application. With this data, we implemented Hidden Markov Model(HMM) to predict the next application a user is going to open and implemented a Long short-term memory model(LSTM) to predict the application time usage.

2. Data Collection

With the help from the Intel team, we are able to collect data from PCs using XLSDK tool kits. XLSDK is a set of libraries and tools developed by Intel to help data scientists to extract information from a PC. By using XLSDK, we implemented several input libraries which are modules that allow us to capture a specific dataset.

We implemented the user waits input library, the foreground window input library, mouse input library and the desktop mapper input library. Since our goal is to make predictions about which application the user will probably open at a certain time, and for how long on the desktop, we mainly focus on implementing foreground windows input library and desktop mapper input library because we want to access the data which contains the information of what applications the user has used and how long did the application open for.

For the foreground windows input library, we built a data collector to extract the information of the foreground windows of the user's PC. We log 11 variables into the database, which are the process ID of the foreground application, the thread ID of the foreground application, the top left, top right, bottom left, bottom right coordinate of a foreground window, whether the application is responsive, whether the application is in the Windows Store, the name of the application, the image name of the application, and the class name of the application. While we are dealing with the foreground window, most importantly, we use GetForegroundWindow() to retrieve a handle to a current foreground window. Based on the handle, we can record the process ID and the thread ID of a foreground window. Moreover, based on the handle, with the help of the function GetWindowTextW(), we can extract the name, class and image of a foreground window. With the help of the function GetWindowRect(), we can access and log the position of a foreground window. With all the information we collect for

the foreground window, we can make predictions about when and how the user likes the application to be opened, so that we can pre-process the application for the user.

For the desktop mapper input library, it is similar to what we have done for the foreground window input library, but instead of just monitoring the one window in the front, we captured the entire desktop, and all the windows on the desktop. Besides all the important features the foreground windows contains, now we can open and close the entire desktop, and we can collect the data of the windows order, such that we can know which application is more important than the others since the user put this window in the front, and we can know if the user need multiple windows to complete his job or not, so that when we need to make pre-process for the user, we can open all the windows user need instead of just one.

After our thoughtful consideration, we decided we should use the **foreground window input library** to collect data for making predictions on user application recommendation because we care most about the application names and the time when the user opened the application.

2.1 Foreground window IL difficulty - Repetition

As we implemented the foreground window input library to collect data like application names, images, classes as well as the time. One of the challenges we faced while implementing the foreground window input library is the repetition of data. The data collector collects information every 10 milliseconds. When the same foreground window remains opening for a time period, the same data will be collected many times.

7	2022-11-30 23:15:04.309	8	esrv.exe	0
8	2022-11-30 23:15:04.309	9	VsDebugConsole.exe	0
9	2022-11-30 23:15:04.421	8	esrv.exe	0
10	2022-11-30 23:15:04.421	9	VsDebugConsole.exe	0
11	2022-11-30 23:15:04.532	8	esrv.exe	0
12	2022-11-30 23:15:04.532	9	VsDebugConsole.exe	0
13	2022-11-30 23:15:04.642	8	esrv.exe	0
14	2022-11-30 23:15:04.642	9	VsDebugConsole.exe	0
15	2022-11-30 23:15:04.754	8	esrv.exe	0
16	2022-11-30 23:15:04.754	9	VsDebugConsole.exe	0
17	2022-11-30 23:15:04.866	8	esrv.exe	0
18	2022-11-30 23:15:04.866	9	VsDebugConsole.exe	0
19	2022-11-30 23:15:04.975	8	esrv.exe	0
20	2022-11-30 23:15:04.975	9	VsDebugConsole.exe	0

we don’t want to log the same application foreground window records if the window keeps opening because it is redundant. In order to solve this problem, we use the PID as an indicator so that we can log the information as long as there is a change on the foreground windows. By doing so, we can make the data cleaner for analyzing because we remove the meaningless repetitions of data.

2.2 Foreground window IL difficulty - Unicode problem

When we were implementing the foreground window input library and used it to collect PC data, we noticed that sometimes the data shows “Missing String.” like the image shown below.

21	2022-12-02 06:46:53.142	8	Missing String.	0
22	2022-12-02 06:46:53.142	9	explorer.exe	0
23	2022-12-02 06:46:53.641	8	Canada	0
24	2022-12-02 06:46:53.641	9	explorer.exe	0
25	2022-12-02 06:46:54.151	8	Canada	0
26	2022-12-02 06:46:54.151	9	explorer.exe	0
27	2022-12-02 06:46:54.648	8	Canada	0
28	2022-12-02 06:46:54.648	9	explorer.exe	0
29	2022-12-02 06:46:55.160	8	Canada	0
30	2022-12-02 06:46:55.160	9	explorer.exe	0
31	2022-12-02 06:46:55.656	8	Missing String.	0
32	2022-12-02 06:46:55.656	9	explorer.exe	0
33	2022-12-02 06:46:56.168	8	Missing String.	0
34	2022-12-02 06:46:56.168	9	explorer.exe	0
35	2022-12-02 06:46:56.668	8	Missing String.	0

Soon, we found out that the data collector could not recognize data which in another language or uncommon symbols or characters. In order to show the missing data, we implemented Unicode to deal with this problem. Unicode allows for the handling of a wide range of characters from various languages. Unicode is a universal character encoding standard that assigns a unique code point to each character in most of the world's writing systems. This means that each character can be represented by a specific code that can be recognized and processed by computers.

After solving the repetition and unicode problem, we are able to collect a relatively clean and ready-to-use dataset for further analysis.

3. Exploratory Data Analysis (EDA)

After we collected all the data using visual studio, we have about 52 different databases in DB form, and the first thing we need to do is to merge all the dataset into one entire database. We complete this step using the sqlite3 package, with the provided connect function, we can convert all the db type of files into sql queries, and then we use the pandas function read_sql_query to read all sql queries into pandas dataframe, then we call merge functions to combined all these dataframes. Now we have the entire dataset, we then start the data cleaning and EDA steps. The first thing we try to accomplish is to split all the records into different groups, like executable files, title of the files, class of the files, etc. First we tried using the INPUT_ID column to determine which group does the records belong to, but soon we find out there is a problem, since we collected 30 db files and have 30 different attempts instead of keep the data collector running all the time, this cause a problem that the INPUT_IDs for each databases are different. For some of the sql queries we can see 8, 9, 10, 11, but for others they might be something else like 5,6,7,13, and this is totally random. To solve this problem, we decided to first pick only the executable files since we can filter them out based on whether the record ends with “.exe” or “.EXE”. With the filtered dataset, we then decided to conduct further exploratory analysis.

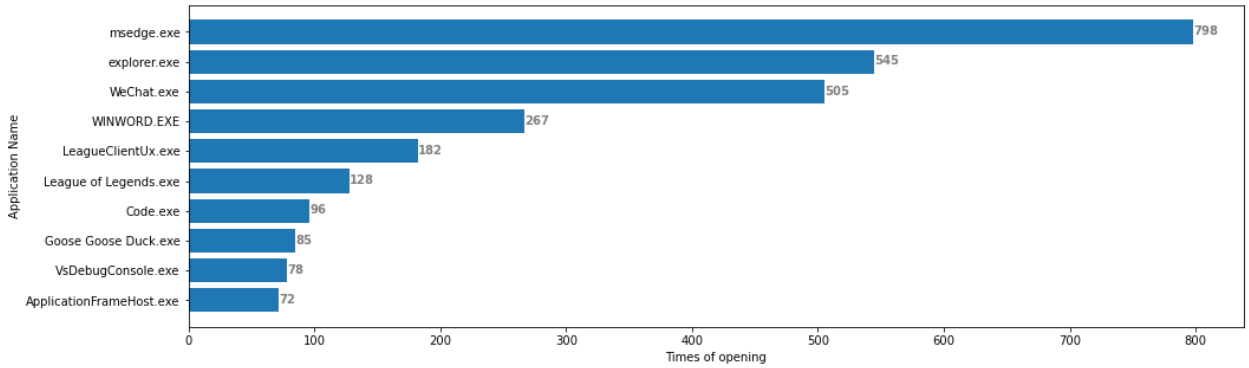
	MEASUREMENT_TIME	ID_INPUT	VALUE	PRIVATE_DATA
0	2022-12-08 06:23:06.681	2	esrv.exe	0
1	2022-12-08 06:23:06.681	3	VsDebugConsole.exe	0
2	2022-12-08 06:23:06.681	4	ConsoleWindowClass	0
3	2022-12-08 06:23:06.681	5	\DISPLAY2	0
4	2022-12-08 06:23:12.697	2	League of Legends	0
...	...	...	...	...
12820	2023-02-10 10:53:52.681	4	\DISPLAY1	0
12821	2023-02-10 10:53:55.693	13	Start	0
12822	2023-02-10 10:53:55.693	2	StartMenuExperienceHost.exe	0
12823	2023-02-10 10:53:55.693	3	Windows.UI.Core.CoreWindow	0
12824	2023-02-10 10:53:55.693	4	\DISPLAY1	0

12825 rows × 4 columns

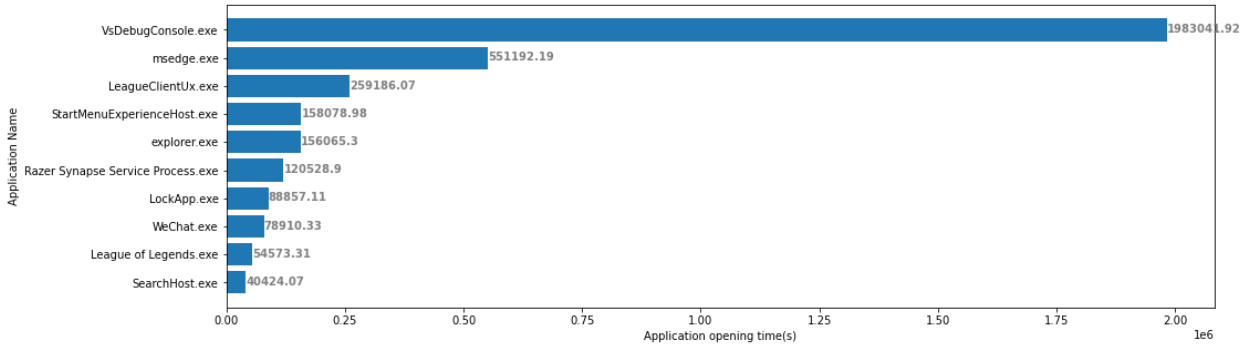
The data consists of 12825 entries. Our data was collected from December 8th, 2022 to February 10th, 2023. The data includes 1540 hours of PC usage which means that from December 8th, 2022 to February 10th, 2023, the user used his PC for 1540 hours and we collected all his foreground window activity.

The user had opened 51 unique PC applications including Microsoft Edge browser, Spotify, Microsoft Word and other commonly used applications. We first find out the most frequently used application which is Microsoft Edge browser (msedge.exe), and we plot the top 10 most frequently used applications in a barplot. And from the plot we can see that it appears

the user used the computer mainly for searching, studying and entertainment.

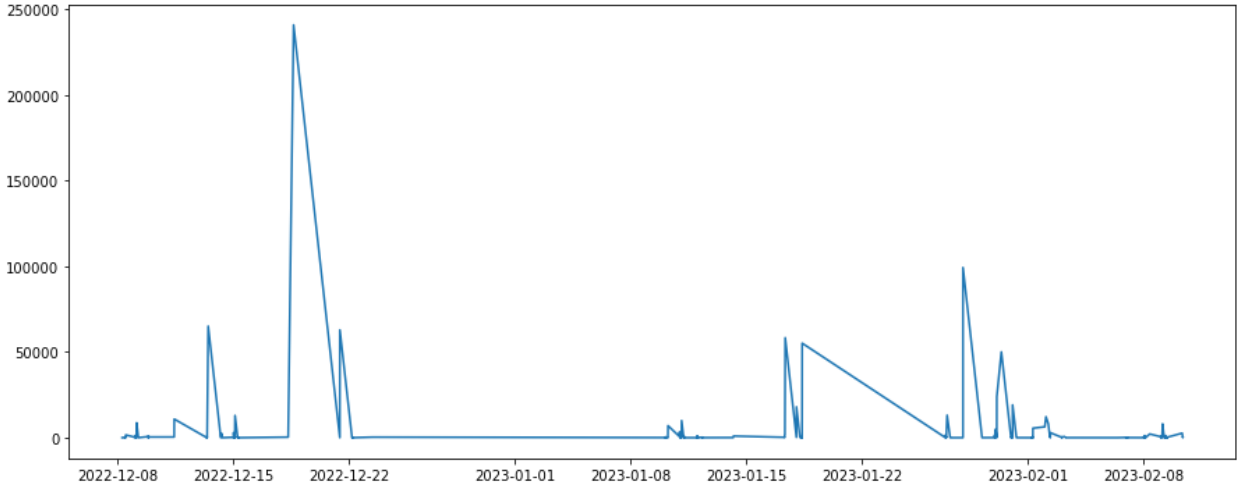


And based on this result, we also want to see if there will be a difference between the most frequently used applications plot and the applications with the longest duration plot. We can see that these two plots are slightly different, the user spends more time on entertainment applications but less frequently using them, while the searching engines were frequently accessed, the total time the user spends is not that long.



We also take a look at the time duration for the most frequently used application, the microsoft edge, and below is the result, we can see that besides the daily usage of several minutes, there are also some really outstanding long time duration of the application. Probably

the user was watching long videos or reading long journals.

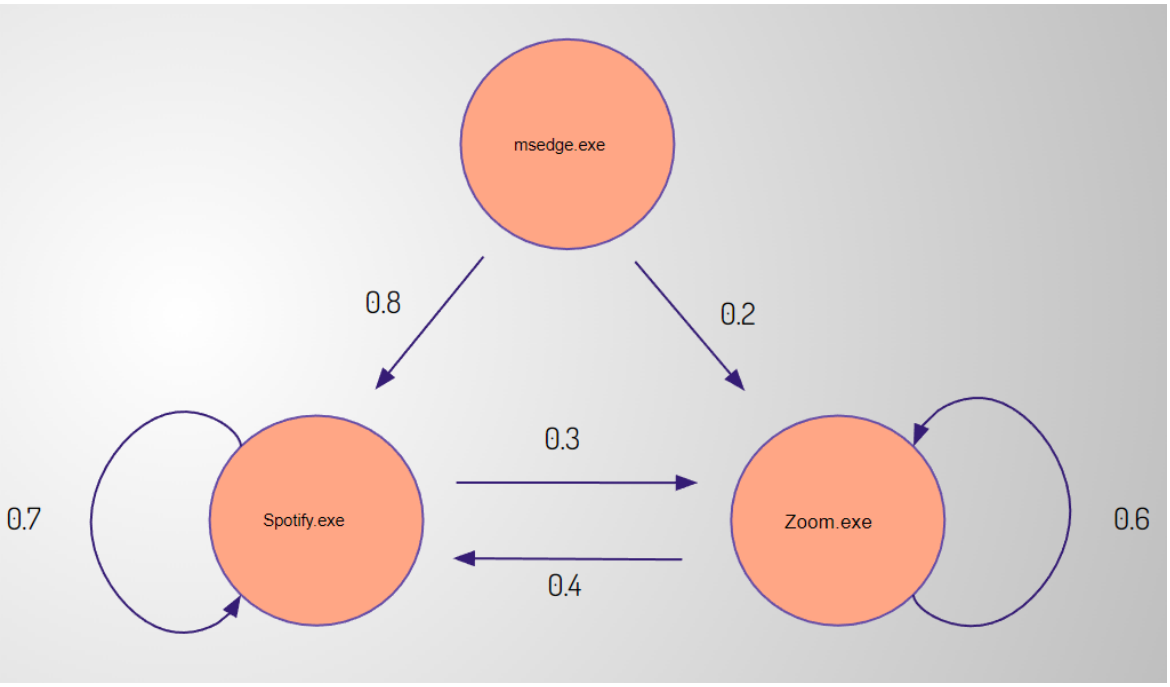


4. Methodology

4.1 Hidden Markov Model

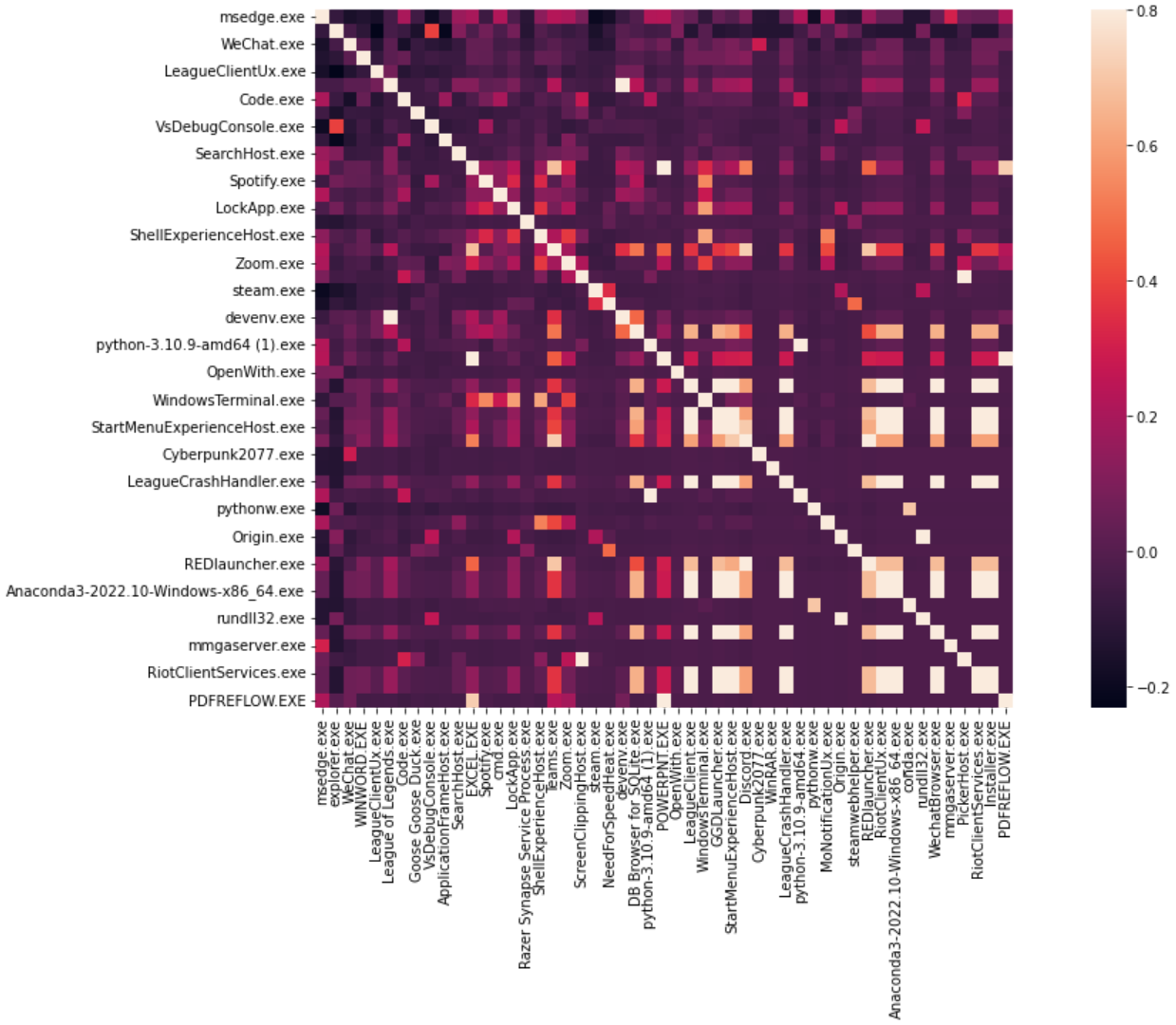
First we decided to use the Hidden Markov model to make predictions about the next applications the user will probably use given the previously used application. A Hidden Markov Model is a type of statistical model that is often used to model sequences of events or observations, where some aspects of the underlying process generating the data are unknown or hidden. The idea behind HMM is that there is a sequence of hidden states that generate the observed data, and these hidden states can only be inferred from the observed data. Think of it like a game of 20 questions. You know some things about an object, but not everything. So, you ask questions to try to determine the underlying hidden state that generated the data you have. In HMM, the hidden states are like the object in the game of 20 questions, and the observed data are like the answers to the questions. The key idea behind HMM is that the hidden states follow a Markov process, which means that the current state only depends on the previous state. This allows us to use mathematical techniques to make predictions about the future based on the past. In summary, a Hidden Markov Model is a statistical model that is used to model sequences of

data when there is some underlying structure that is not directly observable, but can be inferred based on the observations.



In our project, the hidden state is the connection between the applications.We can see this kind of hidden connection and the hidden probability from the figure above. We can use this figure as an example, starting with msedge.exe, it has 2 start probabilities that go to either spotify or zoom. And for spotify and zoom, they have their own probabilities. Those probabilities came from the hidden connections. As a human being we can easily figure out these kind of connections, such as when you are working with a windows word file, you are more likely to access the browser or visual code or other applications that related with studying, but when you are currently playing games or watching videos, you probably won't open a word document all of a sudden and start to do your work. And what we want to do here is to use the hidden markov model to teach the computer about this hidden connection, and use this model to make predictions based on the previous application the user is working with. To accomplish this, we first implemented the start probability of all the applications appearing in the dataset. We do that

by taking each applications' count and dividing it by the sum of all the counts. And then for all of the applications, computed the transition matrix, which is a matrix that indicates the probability each application will be opened based on the application provided. This step is conducted by repeating the same process that we do for the starting probability, but this time we need to multiply by the probability of each previous application. Finally, we integrate all the previous steps into a hidden markov model, and then we can test out the model with the entire dataset split into training and testing data. Here is the heatmap showing the corresponding correlation matrix of all the applications.



Based on the correlation matrix, we can then use it to implement our prediction model. To increase the accuracy, also to provide a more reasonable output, we include variable n to represent the number of possible output we want, which is the applications that have the highest n probabilities in our transition matrix. The total accuracy increased while n increased. Here is a table showing how the total accuracy changed while variable n changed.

Number of predictions	Total accuracy
1	22.05%
2	33.82%
3	60.29%
6	69.85%
8	73.53%

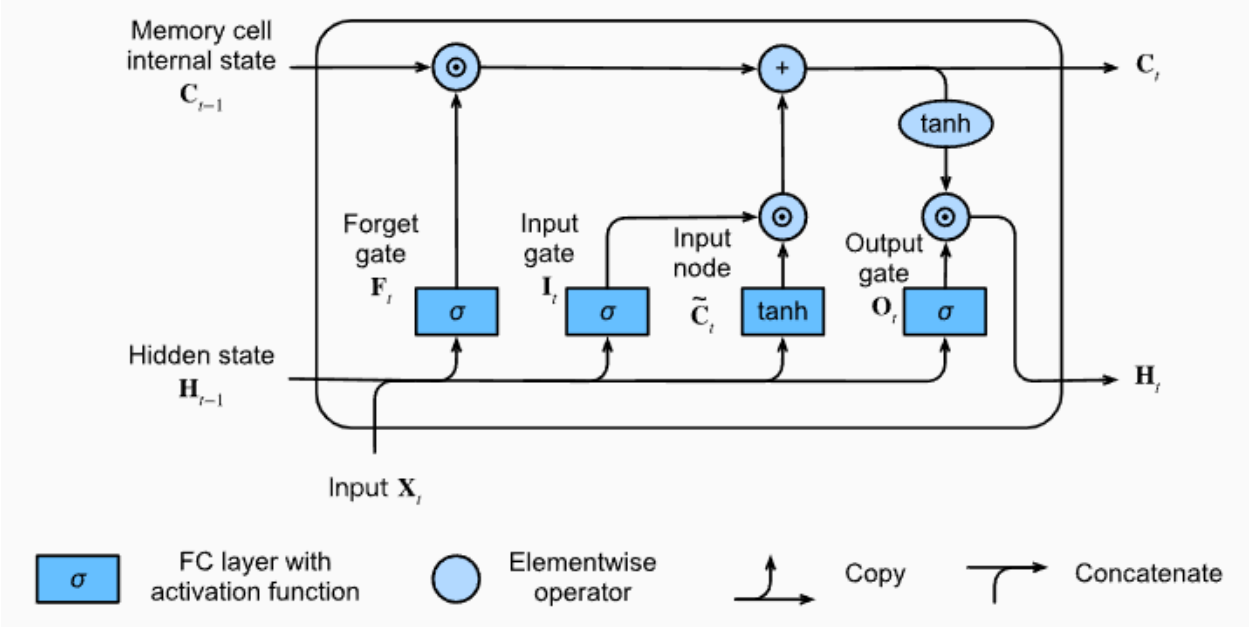
From the table we can see, with this model, we developed our maximum accuracy to 73.53% with 8 possible results. Compared with general prediction models, the advantage for HMM is we don't need to trace through the entire training dataset every time we make a prediction. With the help of the transition matrix, we can easily search for the hidden connections we need and return the result based on that.

4.2 LSTM Time Series Forecasting

Recurrent Neural Network (RNN) is a type of deep learning model that is commonly used for processing sequential data, such as natural language processing, and time series prediction. Unlike traditional neural networks, RNNs have the ability to retain information from previous time steps, which allows them to model the dependencies between elements in a sequence. The basic idea behind RNNs is to have a hidden state which is passed from previous

step to the next, allowing the model to store information of the past inputs and use that information to make predictions about future outputs. There are various types of RNNs, including Simple RNNs, Long Short-Term Memory (LSTM) networks, and Gated Recurrent Units (GRUs).

We chose LSTM networks to help us make predictions on the time duration of a foreground window, because LSTM can effectively capture short-term and long-term dependencies in sequential data. LSTM can use memory cells and gates to control the flow of information into and out of the memory cells so that when we input our data into the model, the gates will determine when to add new information getting in the input gate and prevent irrelevant information from being a part of the calculation by passing it into the forget gate. Therefore, LSTM is widely used in time-series forecasting such as stock price prediction, sales forecasting, traffic flow prediction, energy demand forecasting, and weather forecasting.



Since we want to make a prediction on the application time usage along with the timeline, LSTM is a good machine learning model to apply, which helps us achieve this task. We decided to use the most frequently used application - Microsoft Edge browser(msedge.exe) as the primary application to fit into the LSTM model because this application was used actively which can be more representative as we put it into the LSTM model.

Before we implement the LSTM model, we first need to do feature engineering and transform the data into a condition which is ready to use for the LSTM model. Firstly, we extracted the time and categorized it to hour, minute, day, month. Then, we encoded the application names and added it to the dataframe. Lastly, we added the time duration of each foreground window into the dataframe.

	time_diff_seconds	ts_hour	ts_minutes	day	month	app_encoded
1	6.016	6	23	8	12	8
5	9.033	6	23	8	12	4
9	6.002	6	23	8	12	0
13	18.058	6	23	8	12	4
17	3.003	6	23	8	12	10
...	...	...	...	...	...	...
12802	12.043	10	53	10	2	7
12806	3.003	10	53	10	2	0
12810	3.002	10	53	10	2	1
12814	3.009	10	53	10	2	10
12818	3.012	10	53	10	2	0

\

After we extracted and filtered the data we needed for further use, we had to transform the numerical data with MinMaxScaler which is to normalize the input variables.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

With the MinMaxScaler, the input variables will be scaled to range[0,1]. We wanted the input variables to be scaled because variables are measured at different scales and do not contribute equally to the model fitting, and thus end up with bias. Normalizing the input variables can help improve the accuracy of the model and make it better fitted.

After standardizing the time duration of each query, we one-hot encoded the categorical data which are hour, minute, day, and month. One-hot encoding is a process of converting categorical data into a binary representation that can be easily processed by machine learning algorithms. Each categorical variable is transformed into a binary vector of zeros and ones. The reason that we used one-hot encoding is because categorical variables cannot be processed directly by LSTM, which typically only work with numerical data. One-hot encoding transforms categorical data into a numerical format that can be used as input to our LSTM model.

After feature engineering, we created a function to reshape the variable inputs which is able to determine how many inputs can be passed into the model at a time. Then, we split the input into a train set(80%) and a test set(20%).

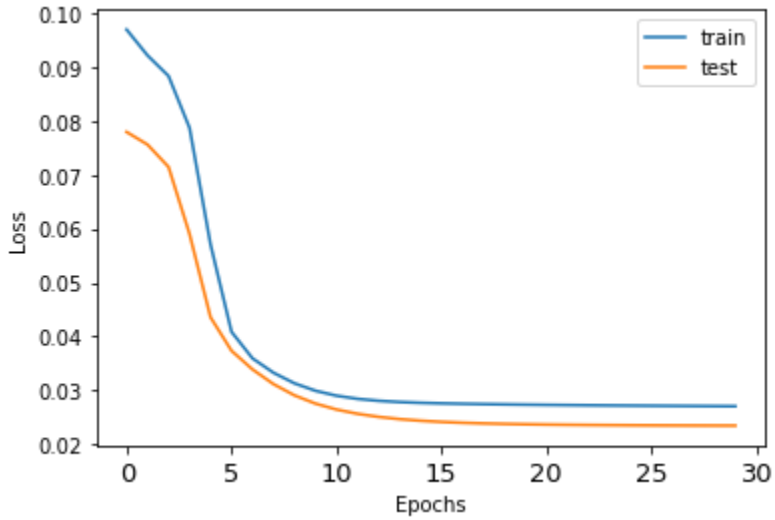
Finishing all the data preparation process, we started to build our LSTM model. We added 4 LSTM layers with 5, 10, 3, 10 units respectively following with a dropout layer. At the end, we added a dense layer with 1 unit because we wanted the model outputs to be one dimension after passing through all the LSTM layers.

```
model = Sequential()


model.add(LSTM(units = 5, return_sequences=True,input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
model.add(Dropout(0.2))

model.add(LSTM(units = 10, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units = 3, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units = 10))
model.add(Dropout(0.2))

model.add(Dense(units = 1))

model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_13 (LSTM)               (None, 1, 5)              220
_________________________________________________________________
dropout_4 (Dropout)          (None, 1, 5)              0
_________________________________________________________________
lstm_14 (LSTM)               (None, 1, 10)             640
_________________________________________________________________
dropout_5 (Dropout)          (None, 1, 10)             0
_________________________________________________________________
lstm_15 (LSTM)               (None, 1, 3)              168
_________________________________________________________________
dropout_6 (Dropout)          (None, 1, 3)              0
_________________________________________________________________
lstm_16 (LSTM)               (None, 10)                560
_________________________________________________________________
dropout_7 (Dropout)          (None, 10)                0
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 11
=================================================================
Total params: 1,599
Trainable params: 1,599
Non-trainable params: 0
_________________________________________________________________
```

c

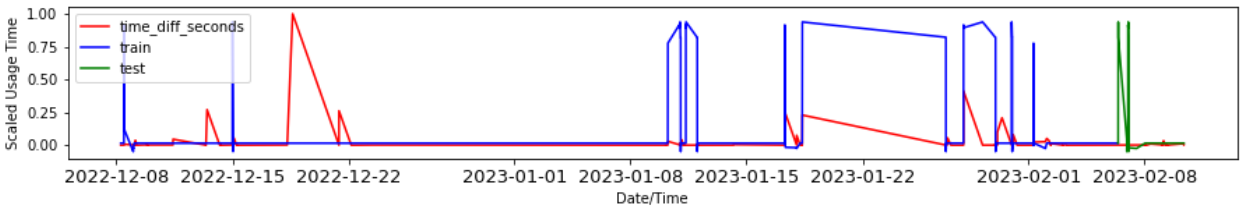
Then, we plotted the loss of the training set and the test set.



From the above graph of loss of the training set and test, we found that the loss of both sets dropped significantly at the beginning of 30 epochs and the loss of both sets converged after 30 epochs, which indicates that our LSTM model is effective.

As we go deeper, we scaled all the collected data and used the scaled data to make predictions. And to analyze the accuracy of the scaled true value and prediction, we implemented a threshold of 0.02 to check if the difference between predictions and true values are significant or not. 0.02 is in a unit that data was standardized by using MinMaxScaler.

With the threshold, we can successfully measure the difference between the predicted and true value, which then makes the accuracy up to 88.67%.



5. Future Expectation

5.1 Additional data

In our project, we only collect PC usage data for 3 months. We hope we can collect more data to fit into the HMM and LSTM in the future which may help improve the accuracy of the models. Firstly, having more data can help the model, especially LSTM, learn a more accurate representation of the underlying patterns in the data, which can improve its ability to generalize to new and unseen data, since LSTM is a type of time-series forecasting model. Secondly, collecting more data can help reduce overfitting by providing more examples for the model to learn from.

5.2 User Interface Development

Since we have successfully built a HMM model to predict the next application a user is going to open based on the current foreground window, we hope we can build a user interface for users to help them open the application they may want to open next faster. The user interface is like a taskbar located at the bottom of the window. There will be four applications shown on the taskbar which correspond to the four most possible applications which the user is going to open next. Users can easily access the recommended applications through the application icons on the taskbar.

5.3 foreground window auto adjustment

We have collected foreground window position data while we implemented the foreground window input library but those data have not been used in this project. In the future, we hope we can utilize that data to generate a machine learning model to predict the size and position of a foreground window which can help users make adjustments to their foreground window automatically.

5.4 Identify and close pop-up harassment windows

Often, when people are browsing on the internet, some pop-up harassment windows will suddenly show up from nowhere, which seriously affects the user experience on a PC. People really hate these kinds of harassment messages. By analyzing foreground window data we collected, in the future, we are able to identify and close pop-up harassment windows for users. Most of the time, the user will close the window as soon as this kind of window shows up, so the time duration of this opening window is very short. In this situation, we will assign a ‘spam’ score to it. If the window with the same application name, image, and class shows up multiple times and was canceled immediately multiple times, we will assign a higher and higher ‘spam’ score to the window. If it passes a threshold we set, this window would be classified as ‘spam’. When the window shows up next time, it will be closed as soon as possible by using our algorithm.

6. Conclusion

Studying user’s application usage habits, prediction the user application launch and usage time, improving PC user experience are our objectives. Beginning from the data collection part, we have built input libraries from scratch, and we used those input libraries to create a data collector to fetch data that we need for building the prediction models. We mainly used the data collected with the foreground window input library on our prediction models.

After cleaning and filtering the data, we constructed an HMM model to achieve foreground application recommendation by predicting the next applications which the user is

going to open next based on the current foreground window. We also built a LSTM model to predict the application time usage of the most frequently used application.

With all the work we developed in our entire project and all these high accuracy results of our prediction compared to the real value, we can now come back to the question we introduced in the beginning, can we now use the user's data we collected, to build up an application prediction and recommendation system for the user? I believe that the answer will be yes. Within our project, we can already predict the user's next possible applications with a relatively high accuracy, and also we can predict the duration of different applications. So if we combine these two models together, we can build up a system that opens and closes applications for the user, and the user will be more efficient during his working experience. And with more and more data having been collected, we believe the accuracy of the model will also increase a lot. From here, the next step we will focus on is to identify pop up windows for the user, such as all those harassment advertisements, or those pop-up windows installed with the application itself, and try to close them in advance before the user even notices them.

References

Long Short-Term Memory (LSTM)(2022), Dive into deep learning. 10.1. Long Short-Term Memory (LSTM) — Dive into Deep Learning 1.0.0-beta0 documentation (d2l.ai)