INTELli*next*: A Fully Integrated LSTM and HMM-Based Solution for Next-App Prediction with Intel SUR SDK Data Collection

Cyril Gorlla
Jared Thach
Hiroki Hoshida
cyril.m.gorlla@jacobs.ucsd.edu
j1thach@ucsd.edu
hhoshida@ucsd.edu
Halıcıoğlu Data Science Institute
University of California San Diego
La Jolla, California, USA

Abstract
Users often face a myriad of miniscule slow-downs when navigating a computer. These slow-downs largely take the form of application loading times and this data science project aims to first collect user data by building custom data collectors. The collected data can then be used to study user patterns and to extract insights via machine learning practices and models such as Hidden Markov Models (HMM) and Long Short-Term Memory (LSTM) Models. By leveraging these models, the inconvenience of accumulated application start-up times can be greatly reduced.

Key words: data collection, user pattern, machine learning, Hidden Markov Model, Long Short-Term Memory Model

1. Introduction

A

2. Methods

Data Collection and Preliminary Analysis

Before diving into our core data analysis and data science in order to develop solutions for next-app prediction, it is vital to retrospectively utilize the previous quarter's data collection modules. It is important to note that these modules were custom developed in C to collect data directly from the users' computers, rather than other data science pipelines which generally source their data online and for free. The four key data collection modules (input libraries) are listed below:

- `mouse_input`
- `user_waiting`
- `foreground_window`
- `desktop_mapper`

Each of these input libraries are detailed further in our previous quarter's white paper.

With our Input Libraries fully functional, they were continuously run on two computers (identified by the IDs LAPTOP-MP2GILK8 and DESKTOP-E83M0TM) over the course of three months, from November 2021 to February 2022. The users were advised to continue operating the machine according to their regular schedules to ensure that the following analysis would best generate insights on naturally occurring user patterns. In total, the Input Libraries collected over 160,000 rows of raw data in 76 database (.db) files from the two machines.

Thanks to our extensive data validation process we built in the first half of the project, the data cleanup needed was minimal. These processes include functions to check for appropriate data types, improbable values (such as negative mouse X and Y coordinates), and null values. As a result, the manual work required mainly consisted of column renaming and other simple DataFrame manipulations. After extracting just the foreground process information, we were left with two datasets with 5,241 and 10,113 rows respectively.

We found that each computer had separate periods of high and low usage, with general boundaries around Fall Quarter 2021, Winter Break 2021/2022, and Winter Quarter 2022. Because of the higher consistency of data of some periods depending on the machine, we used data before December 10, 2021 for the laptop and we used data after January 7, 2022 for the desktop.

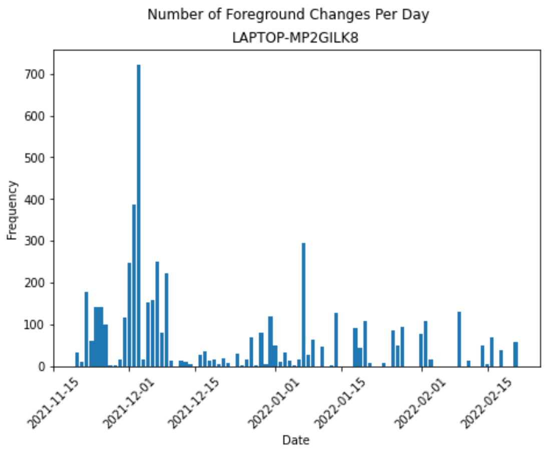

Figure 1: *laptop foreground changes per day*

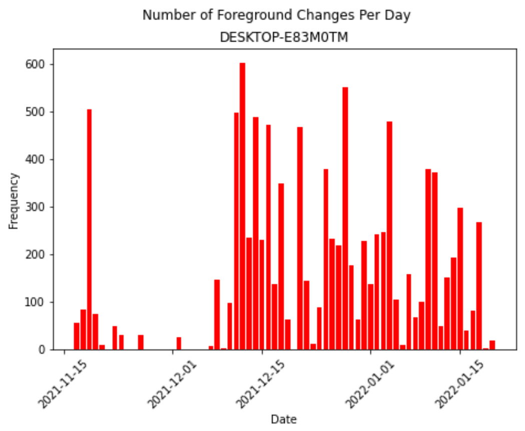

Figure 2: *desktop foreground changes per day*

Some trends in the data can be seen in the exploratory plots below. For instance, Chrome was the most common application in the dataset followed by Windows Explorer for both datasets, but the trends differ after those.

We found that each computer had separate periods of high and low usage, with general boundaries around Fall Quarter 2021, Winter Break 2021/2022, and Winter Quarter 2022. Because of the higher consistency of data of some periods depending on the machine, we used data before December 10, 2021 for the laptop and we used data after January 7, 2022 for the desktop.

Some trends in the data can be seen in the exploratory plots below. For instance, Chrome was the most common application in the dataset followed by Windows Explorer for both datasets, but the trends differ after those.

(PLOT HERE slide 13)

3. Data Science/Data Analysis

We addressed two predictive tasks using the data collected. For our first task, we built models to predict the next application used based on previous user activity. For our second task, we built a model to predict the duration the next application would be used based on previous user activity.

Task 1 Approach 1. Next-App Prediction: Hidden Markov Model

The Hidden Markov Model (HMM) is a particular machine learning approach that ingests sequences of time-related events in order to predict future events. For our purposes, we manually implemented a First Order HMM (adopting a similar model class as the package, scikit-learn) which looks at a single previous application in order to predict the single next application. A HMM’s “order” refers to the amount of previous applications, or “look-back”, the HMM will use in order to generate a single next prediction, and therefore, our First Order HMM will use a single previous event to predict a single next event. This is equivalent to simply computing conditional probabilities of each unique 2 sequence event; for a given previous event A, the probability of the next event B is determined by:

$$P(B|A) = \frac{P(B \cap A)}{P(A)}$$

These probabilistic values are calculated upon model training and stored into a posterior matrix instance variable. These probabilities are then recalled upon prediction.

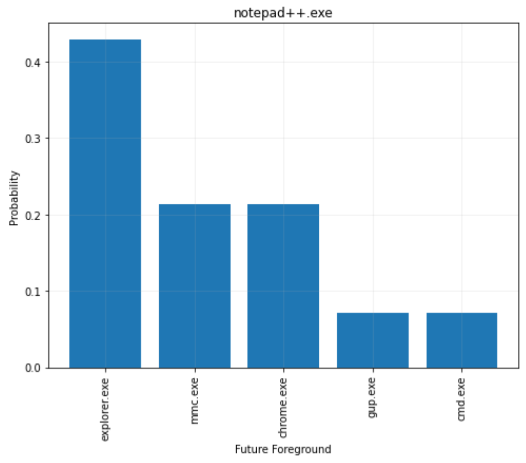

Figure 3: *conditional probabilities of "notepad++.exe"*

Given an input foreground of "notepad++.exe," for example, the probability of "explorer.exe" being the next foreground is approximately 43% as shown in *Figure 3*.

One special implementation of our First Order HMM is the custom `predict` function which has an optional parameter of `n_foregrounds` which specifies the number of foregrounds to return for a single input, based on the foregrounds with the highest probabilities. When viewing *Figure 3* yet again, a `predict` with `n_foregrounds = 3` will return the list `['explorer.exe', 'mmc.exe', 'chrome.exe']`.

Accuracy for a single observation is therefore calculated by determining whether or not the true foreground application exists within the list of predicted foreground applications. Using `n_foregrounds = 3` for our testing dataset, we achieve a final accuracy of 70.05%.

Task 1 Approach 2. Next-App Prediction: Long Short-Term Memory

Though our Hidden Markov Model implementation was fairly successful, we wanted to approach the task from a different angle, so we built a Long Short-Term Memory Recurrent Neural Network (LSTM RNN) model as well.

Because of the higher complexity of LSTMs compared to HMMs, we developed our model using Keras, a deep learning API built on Python that allows for a streamlined pipeline of model creation. By using Keras, we could manipulate various values quickly and easily, allowing us to test many variations of the models in a short period of time, without having to rewrite any significant part of the code. Keras also contains many preprocessing and testing functions that allow us to prepare our data and test the model accuracy easily.

We decided on a RNN model because RNNs are a type of neural network that uses previous inputs to make new predictions. This was perfect for our case, as our data is time based, and we were trying to make future predictions based on past actions. Furthermore, out of the many implementations of RNNs, we decided it was best to use a Long Short-Term Memory (LSTM) model. This is because LSTM models are more tailored to situations where the durations between events vary. In our case, the durations between each foreground application switch were different, meaning that the timestamps of each point of data were spread unevenly.

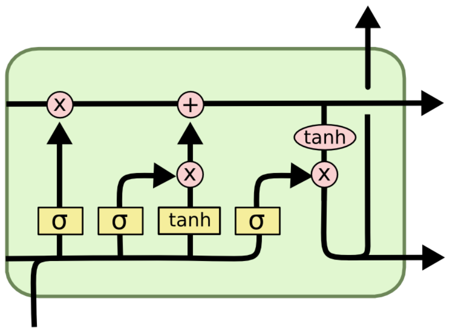

Figure 4: *LSTM cell*

In order to use our data to create the model, we first needed to process it into the correct format so Keras could read it in. First, we needed to select the features to use for our model. In other words, the model would use these properties to make the prediction. We used the time of foreground change, the previous application used, and the time the user spent on the previous application right before the foreground window change. Even with the features selected, however, we had to do further preprocessing to make the data usable. First, we extracted just the hour out of the timestamp. This is because we wanted to see if app usage depended on time of day, and removing non-repeating elements (such as month and day) and removing elements too minute (such as minutes and seconds) seemed appropriate. Second, we one-hot encoded the previous application used. As we found that some of the applications found in the dataset only appeared once or twice (installers, for example), we found the top ten most used applications, and renamed the rest to "Other". Though initially we did not do this, after multiple rounds of testing we found that removing the lower outliers improved our model accuracy tremendously. Finally, we created the output column by shifting the application name column by one. After preprocessing, we split the data into a 70:30 ratio, in which 70% of the data was used for training, and 30% was used for testing.

The training dataset was used to create the LSTM model in Keras. Our final model consisted of four layers; an LSTM layer, a dropout layer, another LSTM layer, and a dense layer with a softmax activation function. Thanks to the flexibility and ease of use of Keras layers, we were able to experiment with various configurations, and we found that using the ADAM optimizer, the categorical cross-entropy loss function, and 100 epochs created a fairly accurate model of our training set.

In order to use our data to create the model, we first needed to process it into the correct format so Keras could read it in. First, we needed to select the features to use for our model. In other words, the model would use these properties to make the prediction. We used the time of foreground change, the previous application used, and the time the user spent on the previous application right before the foreground window change. Even with the features selected, however, we had to do further preprocessing to make

the data usable. First, we extracted just the hour out of the timestamp. This is because we wanted to see if app usage depended on time of day, and removing non-repeating elements (such as month and day) and removing elements too minute (such as minutes and seconds) seemed appropriate. Second, we one-hot encoded the previous application used. As we found that some of the applications found in the dataset only appeared once or twice (installers, for example), we found the top ten most used applications, and renamed the rest to "Other". Though initially we did not do this, after multiple rounds of testing we found that removing the lower outliers improved our model accuracy tremendously. Finally, we created the output column by shifting the application name column by one. After preprocessing, we split the data into a 70:30 ratio, in which 70% of the data was used for training, and 30% was used for testing.

The training dataset was used to create the LSTM model in Keras. Our final model consisted of four layers; an LSTM layer, a dropout layer, another LSTM layer, and a dense layer with a softmax activation function. Thanks to the flexibility and ease of use of Keras layers, we were able to experiment with various configurations, and we found that using the ADAM optimizer, the categorical cross-entropy loss function, and 100 epochs created a fairly accurate model of our training set.

Similarly to our HMM, our LSTM returned a list of the top four most likely applications to follow the current foreground application, ordered in levels of confidence. Accuracy was calculated with the same function (where a single observation was labeled accurate if the true future foreground application appeared in the list of predictions). Our LSTM model had a test accuracy of 68.60%, a value similar to our HMM accuracy.

Task 2. Next-App Duration Prediction: Long Short-Term Memory

Because of the flexibility of the LSTM model, we decided to focus on a LSTM solution for task 2, predicting next-application durations, as well.

Our task 1 LSTM model used a "look-back" value of one previous foreground application in order to predict one future foreground application, where a "look-back" is defined as the number of previous events a single input will use in order to generate the next output. In order to raise accuracy, our task 2 LSTM used a look-back value of five. In other words, the model uses the previous five data points to predict the next. The task 2 model architecture is similar to our task 1 model, with the four layers in the same order. However, we used a MAE loss function and 25 epochs instead. After training, our model made predictions with a mean absolute error of 0.74 minutes. With our testing dataset's mean foreground duration of 1.73 minutes, on average our model predicted foreground durations accurate within approximately 44 seconds.

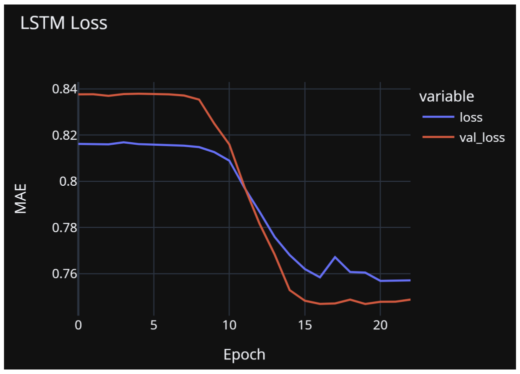

Figure 5: *2nd LSTM's training and validation loss*

4. Results

Our project addressed two predictive tasks: (1) next-app prediction and (2) next-app duration prediction.

Task 1 was explored via two machine learning models with our First Order Hidden Markov Model yielding an accuracy of 70.05% using a prediction list of three and our Long Short-Term Memory Model yielding an accuracy of 68.60% using a prediction list of four.

Task 2 was explored through a single Long Short-Term Memory Model with an average error rate of 42.77%, or 44.4 seconds.

5. Conclusion

Developing a data science project from start to finish has certainly been a great challenge and learning experience. By leveraging C programming to custom code data collectors, we were able to efficiently collect user data while minimizing extraneous information gathering. Additionally, we have grown to accept primary data sourcing as an essential aspect of machine learning practices; striving towards becoming a full-stack data scientist will reap benefits by giving one a greater, holistic view of all stages of a data science pipeline.

Although we have successfully solved our two project tasks of (1) next-app prediction and (2) next-app duration prediction, there exists improvements in several realms. Our original HMM was based on a First Order HMM, and therefore, only used single, previous applications to predict future applications. By spending more time working towards a Second, Third, or higher Order HMM, we can hope to achieve greater model performance. Additionally during our data collection phase, there was heavy overlap of distinct time periods, such as Fall Quarter 2021, Winter Break, and Winter Quarter 2022 (based on a college quarter system). In the future, we may aim to clearly demarcate data collection boundaries to ensure quality data that is independent from each distinct time period.

Our project work does not stop here: with the aforementioned model performances for our (classification-based) next-app prediction task and (regression-based) next-app duration prediction task, we hope to implement our data science work for other relevant applications. These other applications include mobile device machine learning, cloud machine learning, and repurposing of obsolete hardware.

Acknowledgements

Jamel Tayeb and Bijan Arbab, Intel; Intel DCA Team