SMART TAB PREDICTOR: A CHROME EXTENSION TO ASSIST BROWSER TASK MANAGEMENT USING MACHINE LEARNING AND DATA ANALYSIS

Brian Hu[1], Evan Gunnell[2], and Yu Sun[2]

[1]Arnold O. Beckman High School, Irvine, CA 92602
[2]California State Polytechnic University, Pomona, CA, 91768

ABSTRACT

The outbreak of the Covid 19 pandemic has forced most schools and businesses to use digital learning and working. Many people have repetitive web browsing activities or encounter too many open tabs causing slowness in surfing the websites. This paper presents a tab predictor application, a Chrome browser extension that uses Machine Learning (ML) to predict the next URL to open based on the time and frequency of current and previous tabs. Nowadays, AI technology has expanded in people's daily lives like self-driving cars and assistive-type robots. The AI ML module in our application is more basic and is built using Python and Scikit-Learn (Sklearn) machine learning libraries. We use JavaScript and Chrome API to collect the browser tab data and store it in a Firebase Cloud Firestore. The ML module then loads data from the Firebase, trains datasets to adapt to a user's patterns, and predicts URLs to recommend opening new URLs. For Machine Learning, we compare three ML models and select the Random Forest Classifier. We also apply SMOTE (Synthetic Minority Oversampling Technique) to make the data-set more balanced, thus improving the prediction accuracy. Both manual tests and Cross Validation are performed to verify the predicted URLs. As a result, using the Smart Tab Predictor application will help students and business workers manage the web browser tabs more efficiently in their daily routine for online classes, online meetings, and other websites.

KEYWORDS

Machine Learning, Chrome extension, Task Management.

1. INTRODUCTION

The lockdown after the Covid-19 pandemic has caused almost all schools to switch to online classes such as Google classroom or Zoom, and businesses to work from home and use online meetings such as WebEx. Using Internet browsers constantly becomes a daily route for many people like high school or college students, individual workers, and company employees. Quite frequently, web browsing activities are repetitive. For example, a student needs to go to a classroom link for each period and click through multiple links to get assignments and class material. In this case, a student's class schedule is somewhat fixed during school hours, meaning that the same URL may be open at the corresponding bell time. Similarly, a businessman could have the same URLs for a daily status report or access repeated websites for work communication. Having a utility that can generate those repetitive URLs automatically will make it more convenient for people who use Internet browsers a lot daily.

David C. Wyld et al. (Eds): NLP, MLTEC, CLBD, SEAPP, NeTIoT, VLSIE, ITCS, ARIA, DMS - 2021
pp. 227-240, 2021. CS & IT - CSCP 2021 DOI: 10.5121/csit.2021.112318

Moreover, a common issue when using a web browser is to have many tabs opened simultaneously, making it difficult to see the title of each website and quickly lose track of which tab is for what purpose. Also, many open tabs will consume more computer resources such as memory, leading to slower page rendering time and sometimes the crash of the browser. When this happens, a person usually feels frustrated and has to close the browser and start over. A survey with our friends and teachers indicates that they have similar issues. It is desirable to reduce the number of open tabs to reduce memory usage and not lose track of different tabs.

The above situations raise a question: is it possible to create an application that can automatically predict URLs and open a tab based on the history of web browsing activities? Browsers like Google Chrome have an Auto-fill feature, which automatically fills the address, user login, or other info on forms and websites when turned on. However, there seems to be no such predictive software akin to Auto-fill for web links. This has motivated us to develop the project presented in this paper, intending to help improve user productivity with fewer clicks and reduce stress. There are some key requirements to such an application. For instance, the prediction accuracy should be high enough for a user willing to use it; The application must be tailored for each user and segregate each user's data from each other; The algorithms should be able to handle randomized and possibly large data-set, etc.

Artificial Intelligence (AI) has gained more usage in real-world applications in recent years, such as robots that can help elders in their daily life, including opening doors, moving boxes and furniture, and performing health checks, etc. Using AI Machine Learning (ML) is a natural way to connect between a user's web browser usage pattern and a heuristic prediction of the next URL to open. Many studies used Machine Learning algorithms in recommendation systems [1]. Some researchers used content-based filtering [2], collaborative filtering [3], and hybrid or extended model [4][5][6] to provide personalized web recommendations. Most of these studies are suitable to be used in a back-end search engine rather than on the client for individual users. Several papers researched URL-based classification and prediction for categorizing web pages to predefined types of websites [7][8]. Some other papers built models based on browser history to either visualize the browser history [9][10], provide interactive prediction [11], or assist revisiting web pages [12][13]. Our goal is to have an auto suggestion of tab URL for front-end users.

In terms of ML algorithms, regression, classification, and clustering are three major types [14]. Regression is predicted based on a math function (Linear or Polynomial) and gives a result which is a scale of a specific type of data. Classification uses input data examples as a training dataset, finds matches, and outputs a prediction based on the existing labeled data. Clustering is an unsupervised classification to separate unlabeled data into several subcategories based on similarity. For the tab prediction in our study, the regression would not work well as the URLs are random, and there is no pattern to build an equation to predict a good result on the function graph. Clustering may not be the ideal model either because some URLs the user accesses could have the same root URL, and these URLs are likely grouped into the same cluster. Classification fits the problem best as the data-set is a collection of URLs, and each one has its class label. By inputting all website URLs a user visits into a database, the classification model can find the best map of the input data and predict where he will go next as every URL has its prediction. In this paper, we explore three ML classification techniques for tab prediction: SVM Support Vector Classifier (SVC), Passive Aggressive Classifier (PAC),and Random Forest Classifier (RFC) [15]. Through executing manual tests and Cross Validation for each Classifier, we demonstrate that the Random Forest Classifier is the winner based on both prediction accuracy and the ability of handling an unbalanced data-set.

In addition, we remove over-fitting by applying the SMOTE [16] oversampling method to maintain a better accuracy and customize each prediction to an individual user's web browsing

history. Our approach is implemented using open source Scikit-Learn (Sklearn) [17] machine learning libraries. For user interaction we develop the application as a Google Chrome extension [18] that a user can download and install. The Chrome browser extension is a plug-in running on the Google Chrome browser for customized features. We leverage JavaScript, Chrome API, Python, Flask and Flask API [19] to build the connection and communication between various components like the chrome extension, back-end database, web server, and the ML module. We utilize Firebase Cloud Firestore [20] to store encrypted tab history data for each user to protect users' privacy. The ML module loads data from the Firebase, trains datasets, and predicts a URL, which will be returned to the user to open. Using the Tab Predictor Chrome extension application will help users save time, need less clicks thus more efficient, making it easier for people to manage their browser tabs. Further work includes the research and application of unsupervised machine learning algorithms, as well as modifying the application for scalability so that it may handle a larger number of users and higher volume of web history data.

The rest of the paper is organized as follows: Section 2 lists the challenges that we met during the design and development of the Tab Predictor Chrome extension. Section 3 describes the methodology, solutions, and details of software modules. Section 4 presents the test results and data analysis on the results. Section 5 summarizes related work. Section 6 provides the conclusion and future work of the project.

2. CHALLENGES

There are various challenges we encounter during the development of the application. This section describes three challenges, related to the machine learning algorithms, user data customization, and dealing with unbalanced data-set.

2.1. Selecting proper Machine Learning models to get a high accuracy

Machine Learning models only learn from what they know and are used purely to predict a result based on their testing data. Even with a lot of data, machine learning models may not reach a high level of accuracy. Depending on how the dataset is organized, certain ML algorithms are better suited than others. Because our application is directly interfacing with the user consistently, trying to ease their web lookup, we need a higher level of accuracy to maintain utility. Otherwise, the program itself seizes to be helpful and will ultimately not be used. Our application involves multiple elements to maintain a high accuracy: removing overfitting by using the SMOTE oversampling method, customizing each prediction to individual user's web history, and selecting a specific machine learning model. Due to the high amount of data gathered for each specific user, the best machine learning model was the Random Forest Classifier. It is easily able to handle all the different large data it is given and can make accurate predictions based on the time tabs were opened.

2.2. Tailoring the Machine Learning prediction for individual users

The Tab Predictor is a user interaction application where it requires specific data for each user. The machine learning algorithm needs to be able to pull data from a specific user's web history. Because each user is unique, prediction data must be tailored to them, or else the predictions would not be useful. In addition, since we are trying to predict every webpage they go to frequently, we are not only going to have unique datasets for individual users but also the relatively large data for each user. This requires us to set up a non-relational database where each user has their own subcategory of web history, allowing us to easily categorize them. In this paper we utilized Firebase to organize all users into a specific area, with each user having

separate data collections. When a user runs a prediction, it pulls from their data collection specifically without interfering with other user's datasets.

2.3. Dealing with unbalanced datasets to improve prediction accuracy

The problem domain lends itself to an imbalanced data-set which is naturally weighted towards one specific result. Due to how frequently users browse the web, most of our data from users do not have an immediate follow up tab to be opened. This leads to most of the predicted results being wrong and having a false accuracy where many of the predicted results are blank. The data-set needs to be modified so that most of the predicted results are not biased towards a singular prediction. We used the Synthetic Minority Oversampling Technique, or "SMOTE", which increases the number of minority classes. By increasing the quantity of less common data elements, the accuracy of our model increased as the data-set became more balanced.

3. SOLUTION

Figure 1 shows the overview of the Tab Predictor application. It mainly contains four components:

- Chrome extension: the front-end interface with users, a plug-in running on the Chrome browser
- Firebase Cloud Firestore: the backend cloud database used for storing the Chrome tab events and web browsing history for individual users.
- Predictor engine: the main Machine Learning modules use the open source Scikit-Learn (Sklearn) Machine Learning libraries, running in a Python Repl.it.
- Flask web server: the bridge between the Predictor engine and the Chrome extension, which will fetch the predicted results from the Machine Learning modules and send back the prediction to the Chrome extension.

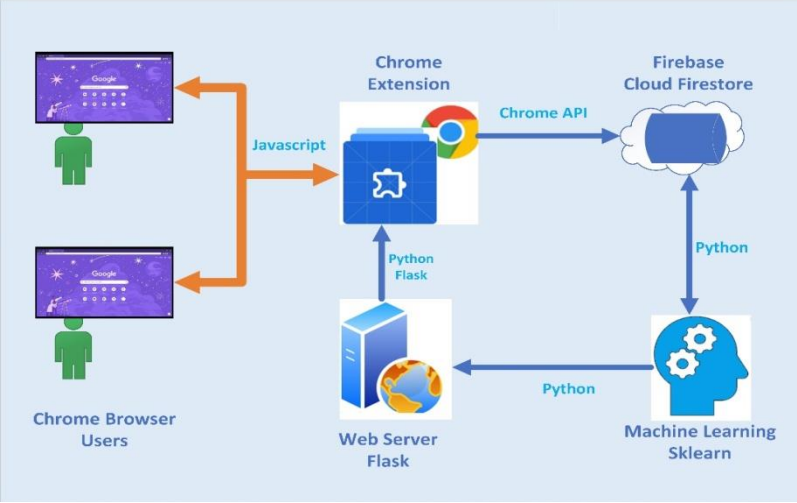

Figure 1. System Overview of the Tab Predictor

We use a code collection of JavaScript, Chrome API, Python, and Python Flask API to develop the Chrome extension and integrate various components. The entire program starts at the Chrome browser tab with a user opening a tab and typing in the search box or address bar. If the user signs in and runs the Chrome extension, the Chrome extension will add the websites or tab

history the user accesses into the Firebase, then the Machine Learning module pulls the web history from the Firebase, iterates through all the websites, and runs its ML algorithms to make the prediction. The predicted URL will be returned to the extension through the Flask web server and prompt for the user to either open or ignore the recommendation. If the ML engine does not find a valid prediction for the extension, then it would return -1, telling the web server not to return any URL. The more accurate prediction should relieve users' searching and typing actions in the browser but not overwhelming them with URLs they don't want to open.

Extensions are software programs that enables users to customize the browsing experience. We use JavaScript and Chrome API to create the Chrome extension as the user interface of the Tab Predictor application. Figure 2 shows the Tab Predictor Chrome extension on the browser:

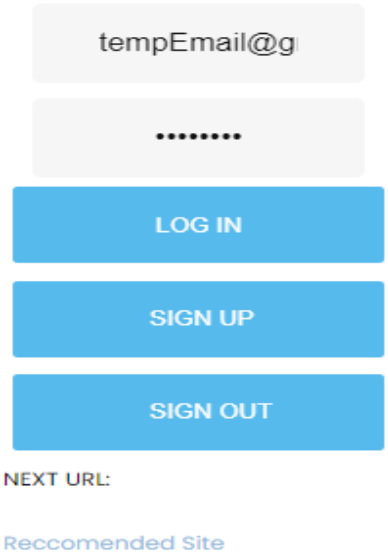

Figure 2. Tab Predictor Chrome Extension User Interface

There are mainly three different areas in the Chrome extension: Back-end scripts, popup script, and content scripts.

The popup scripts handle the user login with a popup window asking for a user's email and password, see the screenshot shown in Figure 3 for stored user login info. The popup scripts are triggered by browser button click action. When the popup is created, it will send a handshake request to tell the background scripts that the popup window is live.

Identifier	Providers	Created ↓	Signed In	User UID
tester1@gmail.com	✉	Sep 15, 2021	Sep 15, 2021	Jv4xlU0CbpTd7t6nvH7ek6dhgNJ2
testemail2@gmail.com	✉	Sep 12, 2021	Sep 12, 2021	OVkqvuLivTcmXwjzmuUPPLJYn9r2
tempemail2@gmail.com	✉	Sep 8, 2021	Sep 8, 2021	2qUtVr6LQ1S3de73URgbEcOyOgK2
tempemail@gmail.com	✉	Jul 1, 2021	Nov 1, 2021	u1TumlM0LLdiSUAeKngP98ZRuq...
testemail3@gmail.com	✉	Jun 24, 2021	Jun 24, 2021	bh2UIh2QvqWNu2h7PHyrgiZu9012

Figure 3. Tab Predictor Chrome Extension User Login Sample

The back-end scripts initialize the connection to the back-end Firebase Cloud Firestore, collect the tab information such as tab Id, tab URL, date, and time, combine the tab info and user login data, then store them into the Firebase. The history is organized in a way that the most recent tabs are always at the bottom, and we know the exact time it was opened. In addition, the back-end scripts listen for events and will send information back to the popup when getting the handshake from the popup window.

The content scripts run whenever a web page is loaded and are between the web page and back-end scripts. They communicate with the back-end scripts through messaging.

One of the paper's essential parts is to define where to save the browser events and history. There are many cloud databases available, such as Firebase Cloud Firestore, Parse Open Source back-end Platform, Back4app, Heroku, etc. [21]. Firebase is selected as it can hold a lot of data while being simple to implement into the project. It provides certain free features and has large data libraries and commands to pull data from or push data into it. Also, it was capable of handling user accounts, allowing for each user to have their own history saved separately.

We organize all users into a specific user data collection where each user has their own subcategory of data. Each user is identified and segregated by a login token, typically an email address and password, and the web history for a user is stored as a child collection of the root. This way individual users have their own section of history URLs. Figure 4 is a sample of the Firebase structure:

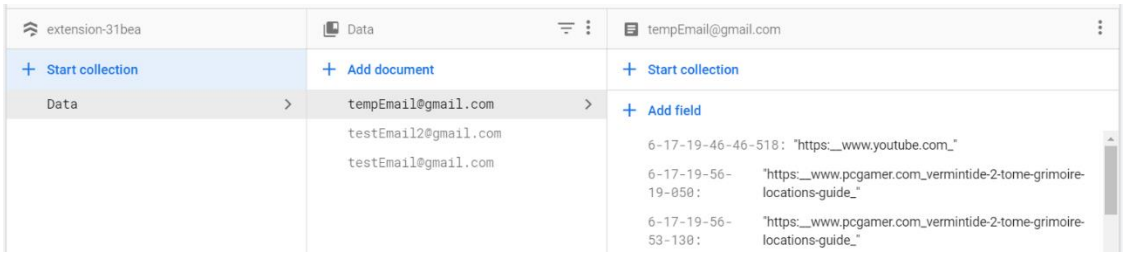

Figure 4. Firebase Cloud Firestore for Saving Tab History Info for Individual Users

Python code is used to access the Firebase and pull data from it. Figure 5 demonstrates the code snippet of connecting and loading data from Firebase: First the user logs in to the Firebase via stored credentials, next the machine learning module reads the data collection from the Firebase for a particular user identified by the user's email, then sorting the data, and at last encoding the dataset for the Machine Learning Classifier.

```
from flask import Flask
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

cred = credentials.Certificate("ServiceKey.json")
firebase_admin.initialize_app(cred)

db = firestore.client()

@app.route("/takeInputOutput/<month>/<day>/<hour>/<minute>/<url>/<email>/")
def takeInputOutput(month, day, hour, minute, url, email):
  try:
    print("Reading from " + email)
    input_data = []
    output_data = []

    my_date = []
    url_data = []
    print("Setting up Encoders....")
    inputEncoder = LabelEncoder()
    outputEncoder = LabelEncoder();
    print("Getting data from Firestore.. ")
    doc = db.collection('Data').document(email).get()
    if(doc.exists):
      doc=doc.to_dict();

    URLs = list(doc.values())
    my_date = list(doc.keys())
    print(my_date)
    if(len(my_date) < 5):
      return "Not enough data to predict."

    print("Sorting Firestore Data")
```

Figure 5. Code Sample for Accessing and Pulling Data from Firebase

The Machine Learning module is built using Python running on a boosted Repl.it platform. Repl.it is a browser-based online development environment for interactive programming.

There are many open-source Machine Learning (ML) libraries. Some popular ones are TensorFlow, Scikit-Learn, Theano, Caffe, Keras, and PyTorch [22]. We select Scikit-Learn(Sklearn) as it contains a robust and easy-to-use library capable of handling most predictions. Importing Sklearn in Python is simple, and there are no external programs that need to be downloaded. Sklearn also has a built-in cross-validation library, which we can use to test the accuracy of the Machine Learning algorithms.

Figure 6 shows the code section for the Machine Learning module. We encode the dataset from the Firebase to the format that the ML classifiers can read, call one of the ML classifiers from the Sklearn libraries to perform the prediction, then decode the predicted result to a human readable format.
The python machine learning algorithm is refitted each time it is used. Since there are multiple users, each with their own unique search history, the machine learning algorithm must learn the patterns of each database every time a new user logs on and types in a new URL.

```
print("Saving Data into DF and encoding it")
df = pd.DataFrame(input_data,
                  columns=["Month", "Day", "Hour", "Minute", "url"])
df["url"] = inputEncoder.fit_transform(df["url"])
input_encoded = df.values.tolist()
output_encoded = outputEncoder.fit_transform(output_data)
x = input_encoded
y = output_encoded
oversample1 = SMOTE(k_neighbors = 2)
x,y = oversample1.fit_resample(x,y)

print("Selecting Model...")
model = RandomForestClassifier(class_weight = "balanced")
print("Fitting model...")
model.fit(input_encoded, output_encoded)
url = inputEncoder.fit_transform([url])[0]
print("Predicting result...")
result = model.predict([[int(month), int(day), int(hour), int(minute), url]])
print(result)

print("Our result is ", result)
print("Our result with output encoded is ", output_encoded[result])
print("Our output_encoded list\n")
final_result = outputEncoder.inverse_transform(output_encoded[result])
print("HERE IS THE 0th ELEMENT")
print(str(final_result[0]))
print("Outputting Result!")
print (str(final_result))
return str(final_result[0])
```

Figure 6. Code Sample for Prediction Using Machine Learning Algorithm

The Flask server is one of the most popular Python web application frameworks, easy to get started with, and has good readability [19]. We leveraged the Python Flask to fetch the prediction outputs from the Machine Learning Module, send results back to the Chrome extension, and present the recommended URL to the user. Figure 7 shows the code section for the Python Flask server. Figure 8 is the screenshot of presenting the recommended URL.

```
from flask import Flask
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

cred = credentials.Certificate("ServiceKey.json")
firebase_admin.initialize_app(cred)

db = firestore.client()

print("ready")

@app.route("/takeInputOutput/<month>/<day>/<hour>/<minute>/<url>/<email>/")
def takeInputOutput(month, day, hour, minute, url, email):

 try:
   print("Reading from " + email)
   input_data = []
   output_data = []
```

Figure 7. Code Sample for Python Flask Server

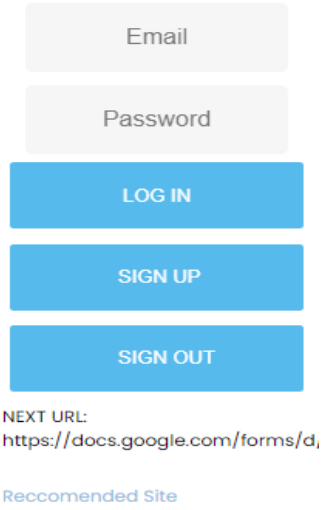

Figure 8. Screenshot of Presenting the Recommended URL

4. EXPERIMENT

Three Classification ML algorithms have been explored: SVM Support Vector Classifier (SVC), Passive Aggressive Classifier (PAC), and Random Forest Classifier (RFC). The purpose is to identify a classification model that supports better prediction. Sklearn K-fold cross-validation is used to calculate the accuracy of the different learning algorithms. Figure 9 and Figure 10 show the Cross-validation (CV) score for each test and the final average. The SVM machine learning model creates vectors with large margins in between the data. It essentially creates sections of fit for all the data points graphed. A Passive Aggressive model is similar to SVM in that it is also a vector classifier; however, it will strongly adjust itself whenever it is incorrect. It will keep doing this until it is correct. Random Forest is a decision tree machine learning model that uses slight random variance when it splits a decision. Random Forest in particular uses this to its advantage and creates multiple slightly different decision trees and averages the result.

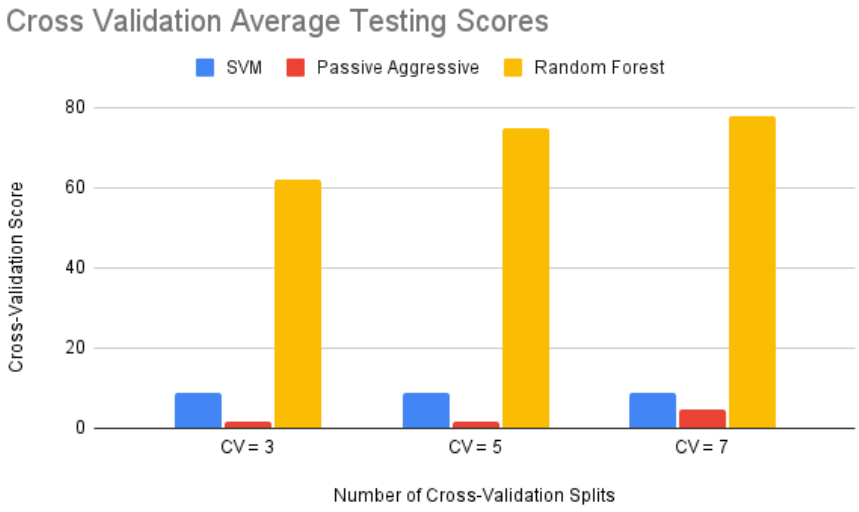

Figure 9. Average Cross-Validation Score for Three Machine Learning Algorithms

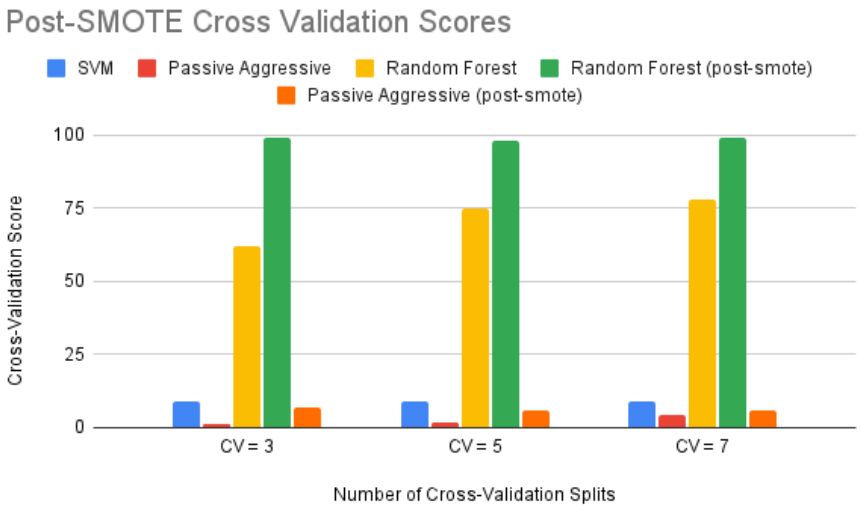

Figure 10. Average Cross-Validation Score for Three Machine Learning Algorithms post SMOTE included

As can be seen from the charts, the CV scores indicate that the Random Forest has the highest accuracy among the three ML algorithms. In some cases, the Passive Aggressive has a similar accuracy score as Random Forest, but it is not as consistent. Random Forest is the best decision for the current datasets not only because it has the highest accuracy but also because it is designed to handle imbalanced datasets very well. Even then post-SMOTE it did a good job. Order of events: 1. Our data was 99% accurate because it was so imbalanced. 2. We used random forest to help with the balanced data-set, but it still resulted in a “no tab” prediction 99% of the time. 3. We then decided to use SMOTE to balance our data-set. This got us a more realistic accuracy of around 65%. As we ran the extension over time we gathered a much larger data-set that helped with the initial imbalances we had. Because of this, with SMOTE, our accuracy shot up to 98% accuracy. With random forest, SMOTE, and now a much broader data-set, it is likely that the model is returning such a high result because it is over-fitted. This over-fitting, while normally a problem, is not a negative outcome because the project is associated with your personal browsing habits and should not base its prediction on anything else. As we now have a much larger data-set, SMOTE is not as necessary. And if we remove it, we get less over-fitting with an accuracy of about 77% accuracy.

Cross Validation results were the determining factor (as well as our own testing) with ~60% accuracy using the Random Forest Classifier. The prediction accuracy without SMOTE is around 99% accurate, as the data was unbalanced, with most results being the same link. It would return “No Tab” as a prediction and get it correct most of the time as most predictions had “No Tab” after it as nothing was opened after a 2-minute period. The SMOTE filled the imbalanced data set, helping the cross-validation produce a more accurate result of 60%.

SVM Feature Importance is utilized to determine how relevant the data is when it comes to predicting the next URL. It checks how many features or properties are in the data, calculates the importance rank of a feature by changing the value of that feature to see the impact on the end results, and returns a percentage indicating the weight of a feature for the prediction. Figure 11 shows the importance of Month, Day, Hour, Minute, and Second collected for Chrome Tab info, followed by the detailed table.

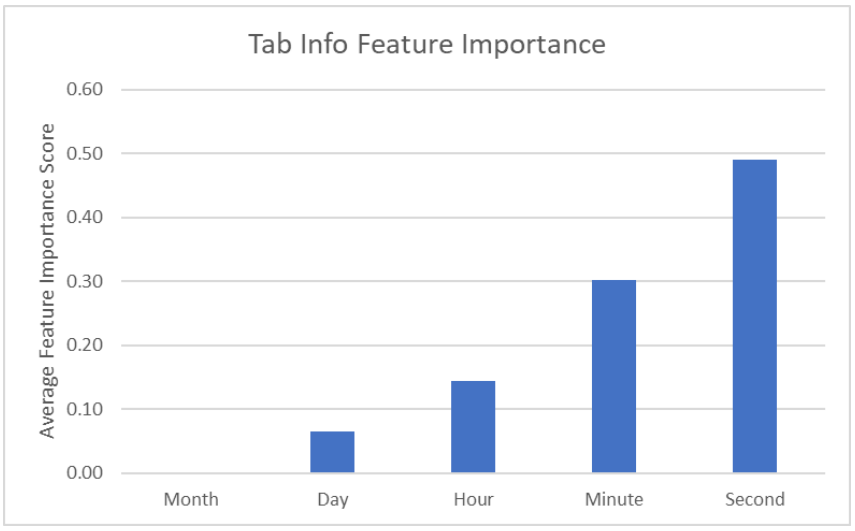

Figure 11. Feature Importance for Chrome Tab URL Time Info

Table 1. Feature Importance for Chrome Tab URL Time Info

Feature Importance	Month	Day	Hour	Minute	Second
#1	0.00000000	0.06086918	0.14408149	0.30362221	0.49142712
#2	0.00000000	0.06527741	0.14247498	0.30155574	0.49069188
#3	0.00000000	0.06473688	0.14882411	0.30013972	0.48629929
#4	0.00000000	0.06289494	0.14125969	0.30368546	0.49215991
#5	0.00000000	0.06487327	0.14333041	0.29936943	0.49242689
#6	0.00000000	0.06380527	0.14345727	0.30237818	0.49035928
#7	0.00000000	0.06240662	0.14423776	0.30371136	0.48964426
#8	0.00000000	0.06759765	0.14065858	0.30333982	0.48840396
#9	0.00000000	0.06714231	0.14755350	0.30054375	0.48476044
#10	0.00000000	0.06529274	0.14443936	0.30105877	0.48920913
Average	**0.00000000**	**0.06448963**	**0.14403172**	**0.30194044**	**0.48953822**

As can be seen from the chart and table, the Second and Minute have the highest feature importance, indicating they are weighted more when finding the best URL for the next prediction. The reason why the Feature Importance of the month is 0, is because the data used to calculate the feature importance was all in the same month. The reason it is not removed is that the algorithm will eventually incorporate months into its predictions.

Before feeding into the Machine Learning Classifiers, we organize the browser tab history data from the Firebase into sorted URLs based on months, days, hours, minutes, seconds, milliseconds, and user email. We go through each instance of the sorted URLs and create input datasets for the ML module, as well as generate the output data-set based on the input data-set.

In order to find the best time interval for the prediction, we executed tests with various levels of intervals such as 30 seconds, 1 minute, 2 minutes, 5 minutes, and 10 minutes. 2 minutes was picked because it is a relatively realistic time for users to reopen something related to their own work. The ML model checks if two URLs were opened within 2 minutes of one another, and if they were, the prediction for that site would be the one opened second. The months and days are used to check for consistency between the URLs that are being predicted. Milliseconds are used to make sure that two URLs are not opened within the same second.

Since our application is targeted for students and people who have repetitive tab opening activities for their daily online classes or work assignments, a lot of times these users do not have an immediate follow up tab to be opened. Also, the tab activities vary between users. These behaviors lead to large, broad, and overly thin datasets in the Firebase. Due to these imbalanced datasets, the ML model tends to predict a blank page or singular prediction. To relieve this type of biased prediction, we utilize the SMOTE technique to balance the datasets. SMOTE is a type of data augmentation by randomly increasing minority class examples by replicating them. We set the k nearest neighbor value to 2, meaning each minority instance will find its 2 nearest neighbors using Euclidean distance [17]. By adding multiple of the minority classes, the data-set becomes more balanced. Results reveal higher accuracy rates after applying SMOTE.

5. RELATED WORK

Shawon and Zuhori propose two methods to make a web browser more intelligent [9]. They used linear regression to calculate website access frequency based on first visit, last visit, and URL counts, sorted the frequency, and then predicted the web link with highest frequency when a user typed in the address bar. They also provide content recommendation by classifying the URLs into several categories (Computers, Arts, Business, Games, etc.) using optimized Naïve Bayes Classifier and then suggesting a list of websites for a particular category. Instead of frequency, our study uses the time frame of the web history URLs as the input data for the Machine Learning module.

Kotapalle and Kandala built a cross-browser plugin, a client-side browser history management extension, that saved the user browsing history and the relationship of tabs in a tree mode using IndexedDB and local storage on the client [10]. They provided a front-end visualization of users' browsing behavior in a tree view or linear structure. Their extension was not yet connected to any back-end Machine Learning system. Our application focuses on Google Chrome extension, and it is integrated with a Cloud data-store and Machine Learning engine.

Woo and Lee proposed a web interaction profiling framework for real time interaction prediction [11]. They developed an event tracing tool to collect both users' navigation and click events using JavaScript event handlers. They adopted Gated Recurrent Unit (GRU) deep learning with URL grouping and Web embedding techniques for the interaction prediction. Their approach collected users' browsing activities at a very detailed level. It seems their framework fits better for commercial web applications to provide recommendations during user interaction with the websites. Our application uses a JavaScript event listener to collect the browser tab URL information only and leverage Random Forest classification.

6. CONCLUSIONS

In this paper we developed a Google Chrome extension tool that uses Machine Learning to predict future URLs to open and prompt the recommendation to the front-end users. We use JavaScript and Chrome API to create the Chrome extension, store tab events and history into the Firebase Cloud Firestore, and integrate the Chrome extension, Firebase, Machine Learning module through the Python Flask web server. The Machine Learning module is built using Sklearn libraries and SVM in Python. Various machine learning models such as Support Vector, Passive Aggressive, and Random Forest were executed and cross-validated to identify the method with a more accurate prediction. Feature Importance is calculated for the time granularity of Month, Day, till Second. In addition, we stacked SMOTE technique and Random Forest classification to make the thin and imbalanced data-set more balanced and achieve better

accuracy. With the current solution model, the system is able to output the predicted URL at a decent prediction accuracy.

The limitation of our study is currently we experiment with a small number of users. Future work involves increasing the Machine Learning prediction performance as accuracy is always the king, testing, and enhancing the solution to be salable to a larger number of users. This could be done via modifying our back-end to incorporate parallel processing and adding more powerful servers. Moreover, we also want to add some adaptive modules using unsupervised Machine Learning techniques so that whenever the user opens or ignores the recommended new tab, we will update the machine learning model with that result. This could be added into our extension's background scripts, with the model being updated once or twice a week.

REFERENCES

[1] Portugal, Ivens, Alencar, Paulo, and Cowan, Donald. "The use of machine learning algorithms in recommender systems: A systematic review." Expert Systems with Applications, Volume 97, 2018, Pages 205-227, ISSN 0957-4174. https://doi.org/10.1016/j.eswa.2017.12.020. (https://www.sciencedirect.com/science/article/pii/S0957417417308333)

[2] Gangurde, R. and Kumar, B. "Web Page Prediction Using Genetic Algorithm and Logistic Regression based on Weblog and Web Content Features," 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), 2020, pp. 68-74, doi: 10.1109/ICESC48915.2020.9155634.

[3] Yu, X., Chu, Y., Jiang, F., Guo, Y., and Gong, D. "SVMs Classification Based Two-side Cross Domain Collaborative Filtering by inferring intrinsic user and item features. "Knowl. Based Syst. 141 (2018): 80-91.

[4] Wanaskar, U., Vij, S., and Mukhopadhyay, D. "A hybrid web recommendation system based on the improved association rule mining algorithm." arXiv preprint arXiv:1311.7204 (2013).

[5] Geetha, G., Safa, M., Fancy, C., and Saranya, D. "A hybrid approach using collaborative filtering and content based filtering for recommender system." Journal of Physics: Conference Series. Vol. 1000. No. 1. IOP Publishing, 2018.

[6] Bhavithra, J. and Saradha, A. "Personalized web page recommendation using case-based clustering and weighted association rule mining." Cluster Comput 22, 6991 –7002 (2019). https://doi.org/10.1007/s10586-018-2053-y

[7] Hernández, I., Rivero, C. R., Ruiz, D., and Corchuelo, R. "CALA: ClAssifying Links Automatically based on their URL." Journal of Systems and Software, 115(2016), 130-143

[8] Rajalakshmi, R.and Xaviar, S. "Experimental Study of Feature Weighting Techniques for URL Based Webpage Classification." Procedia Computer Science, Volume 115, 2017, Pages 218-225, ISSN 1877-0509. https://doi.org/10.1016/j.procs.2017.09.128.

[9] Shawon, A., Zuhori, S. T., Mahmud, F., and Rahman, M. "Web Links Prediction and Category-Wise Recommendation Based on Browser History." arXiv preprint arXiv:1902.08496 (2019).

[10] Kotapalle, G., Kandala, H., and Gade, K. "Extracting relationship between browser history items for improved client-side analytics and recommendations." 2018 3rd International Conference on Contemporary Computing and Informatics (IC3I), 2018, pp. 141-146, doi: 10.1109/IC3I44769.2018.9007258.

[11] Joo, Minwoo and Lee, Wonjun. "WebProfiler: User interaction prediction framework for Web applications." IEEE Access 7 (2019): 154946-154958.

[12] Papadakis, G., Kawase, R., Herder, E. et al. "Methods for web revisitation prediction: survey and experimentation." User Model User-Adap Inter 25, 331 –369 (2015). https://doi.org/10.1007/s11257-015-9161-7

[13] Uddin, I., Khusro, S., Ullah, I., and Rauf, A. "Semantic History: Ontology-Based Modeling of Users' Web Browsing Behaviors for Improved Web Page Revisitation." In Proceedings of the Computational Methods in Systems and Software (pp. 204-215). Springer, Cham, 2018. https://doi.org/10.1007/978-3-030-00184-1_19

[14] Müller, Andreas C., and Guido, Sarah. "Introduction to machine learning with Python: a guide for data scientists." O'Reilly Media, Inc.", 2016.

[15] "Learning model building in Scikit-learn: A Python machine learning library." (2019, August 06). Retrieved February 28, 2021, from https://www.geeksforgeeks.org/learning-model-building-scikit-learn-python-machine-learning-library/
[16] "ML | Handling Imbalanced Data with SMOTE and Near Miss Algorithm in Python." Retrieved June 28, 2021, from https://www.geeksforgeeks.org/ml-handling-imbalanced-data-with-smote-and-near-miss-algorithm-in-python/
[17] Hackeling, Gavin. "Mastering Machine Learning with scikit-learn." Packt Publishing Ltd, 2017.
[18] Berke-Williams, G. "How to make a chrome extension." (2019, March 23). Retrieved February 28, 2021, from https://thoughtbot.com/blog/how-to-make-a-chrome-extension
[19] Grinberg, Miguel. "Flask web development: developing web applications with python." O'Reilly Media, Inc., 2018.
[20] Hajian, Majid. "Deploying to Firebase as the Back End." 10.1007/978-1-4842-4448-7_2, 2019.
[21] "What are some alternatives to Firebase? (n.d.)." Retrieved February 28, 2021, from https://stackshare.io/firebase/alternatives
[22] Dr. Garbade, Michael J. "Top 8 open source AI technologies in machine learning." (May 15, 2018). Retrieved February 28, 2021, from https://opensource.com/article/18/5/top-8-open-source-ai-technologies-machine-learning