A CONTENT-BASED INTELLIGENT CHROME EXTENSION TO ASSIST READING TIME MANAGEMENT USING ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Richard Zhang[1], Ang Li[2]

[1]Oakton High School, 2900 Sutton Rd, Vienna, VA 22181
[2]Computer Science Department, California State Polytechnic University, Pomona, CA 91768

ABSTRACT

Oftentimes we lose track of the time we take to skim over a website or article online or we are simply curious about the time it might take for us to read over some text. We might also be curious about our attention span based on the length or difficulty of an article. This paper details the development process of an intelligent google chrome extension capable of gathering data from specific articles and processing the data to estimate the amount of time needed to read over an article based on the time it took to read similar or dissimilar articles [10]. This application takes into account the length, readability, average word size, and comparisons to other reading times in order to return the most accurate time predictions. The benefit of this application is improved time management as an accurate prediction of time will be provided.

KEYWORDS

Chrome-extension, Time management, Machine learning, Web scraping

1. INTRODUCTION

Time management is a skill that is essential for success in both school and life [11]. It is a way of organizing and planning one's day so that tasks can be accomplished in a timely and effective manner. By managing time wisely, an individual can achieve more in a shorter amount of time and improve their overall productivity. Time management is important for a number of reasons. First, it allows individuals to prioritize tasks, which can eliminate wasted time and energy [12]. This is especially important in school, where the learning process can be overwhelming. With good time management, people can focus on their most important classes and assignments first, without worrying about other tasks that can wait. Time management is also important for life outside of school. It can help individuals to stay on top of important deadlines in their professional and personal lives. Working adults can plan their days in a way that allows them to finish their work on time and still have time for their family and other commitments. This type of organization can also help to reduce stress, as there is less chance of missing something important due to a lack of planning.

This web extension assists in time management in studies as well as in information browsing boosting overall performance for reading tasks.

Some techniques and systems that have been implemented in existing websites to assist in time management have been the addition of a word count/article length. However, a word count is often an inaccurate measurement of the approximate time an article might take to read as there are many more factors than just article length that need to be taken into consideration before estimating reading time. For example, word difficulty, word length, level of reading, etc can all greatly affect the amount of time it takes for someone to read a section of text. More advanced sites provide an average reading time; however, the given prediction is often inaccurate for the same reasons article length is inaccurate. Because people read at different speeds and levels, it is inappropriate to generalize the time needed to read an article to all readers. No matter how websites attempt to provide article information, they fail to tailor an accurate prediction based on individuals and their individual abilities.

In this paper, we follow the same line of research by digging deeper into various factors that affect reading time and how they are accounted for in this web extension [13]. The goal is to provide the most accurate prediction of time required to read a specific web page given website information and user's performances on separate articles. The advantage of using a web extension is that every user can have tailored predictions based on individual performance. The second advantage of using a web extension is that it works across all websites for all users. The method is inspired by the endless capabilities of different google chrome extensions and the countless number of students who struggle with time management. Some features of this web extension are length of the article, average size of words, readability score based on the Gunning Fog Index formula (0.4 * ((average sentence length) + (percentage of Hard Words))), and an accurate time prediction based on comparisons between the current article and times required to read other articles using machine learning [1][2]. In addition, there is a timer feature that can save reading times for more accurate time predictions in the future. Because of these features, users will be much more informed on the approximate time it will take to read a specific article and will therefore be able to manage their time better.

Two experiments were devised to evaluate the accuracy of this web extension for making predictions on reading time. Firstly, multiple subjects trained the model by reading text off of ten randomly selected articles. The data recorded was as follows: reading time, article word length, article readability score, and average word length. Training the model usually requires hundreds of sample data for each user; however, due to practical limitations, the models were trained with fewer than twenty time entries per subject [1]. To easily compare the prediction results with the actual read times, the data was split into two sets: a training and a test set. Secondly, multiple subjects trained their models and read the same article to accurately test the accuracy for each subject. The data and the samples recorded were the same as experiment one. It was found that generally, the model does a better job at predicting times for users with more entries. In addition, the model's predictions are fairly accurate considering the limited entries.

The rest of this paper is structured as follows: Section 2 will detail the multiple challenges faced in this study and how they were overcome; Section 3 will describe the methodology and solution in greater detail; Section 4 details the experiments that were performed in this study, as well as a thorough analysis of the results; Section 5 will list any related works that have also been done regarding website reading time; and lastly, Section 6 will conclude the study and state any future work that may be done.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Web Scraping and Machine Learning Aspect

The python portion, or the backend of the application, was responsible for "reading over" or scraping the website the user was viewing to evaluate values such as article length, word size, readability, etc. This used the public python library beautiful soup (bs4) and required analysis of its documentation to accurately scrape each website for information. In addition, it required multiple layers of preprocessing before it was able to be looked over by other programs and by users. This preprocessing included grouping the data into a compact space or reformatting it so that it could be sent over to the javascript portion, the frontend, of the application. The most challenging portion, however, was the machine learning aspect which implemented sklearn, a python library specializing in machine learning, to compile given datasets using classification to provide an accurate prediction of the amount of time needed to read an article based on data on other articles [3]. This portion required research on the method of machine learning best suited towards time predictions. It also required research on the actual implementation of machine learning in python.

2.2. Google Chrome Extension Development

Developing a Google chrome extension allows for developers to create web applications for all users of Google chrome. This is extremely useful for connecting to all audiences and testing out an application in the real world. Creating a chrome extension was challenging because it was especially difficult to tailor my code to Google's requirements. For example, there were many aspects of my code that I had to revise or remove because of specific policies Google employs to protect user privacy and prevent malware. In addition, to create a chrome extension you must be familiar with their developer interface such as the new manifest v3 and the various functions it offers. On top of that, you must be familiar with HTML and CSS as well as a deep understanding of javascript to truly create an aesthetically pleasing and fully functional application. Lastly, publishing the chrome extension requires developers to go through a process of justifying the various permissions given to the web application in order to protect user privacy. For example, the ActiveTab permission allows an extension to view the current tab a user is viewing. This may lead to privacy concerns which are addressed in permission justifications.

2.3. Python Flask Server Connection With Chrome Extension

There were two components to this web application: the frontend, the Google chrome extension, and the backend, the python flask server. Connecting the python flask server with the chrome extension was particularly challenging because we had to familiarize ourselves with the various functions of a flask server, avoid the CORS (Cross Origin Resource Sharing) policy violation, and transfer data between the frontend and backend. Firstly, we walked through the python flask server documentation and researched the various ways it was possible to transfer data between javascript and python. Next, we implemented the most straightforward methods which led to the CORS policy violation meaning we could not publish our application. This led to further research on the methods to remove CORS from our program which limits some of its capabilities. Lastly, we attempted to connect the python flask server with the javascript program by transferring data in between the scripts. This step took trial and error but eventually we were able to successfully store data such as time, article length, and readability score.

3. SOLUTION

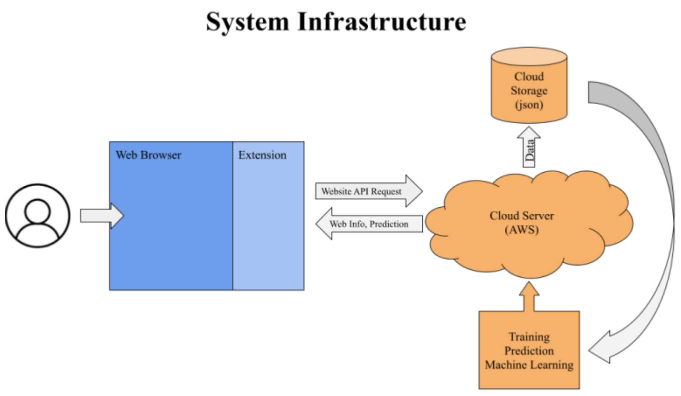
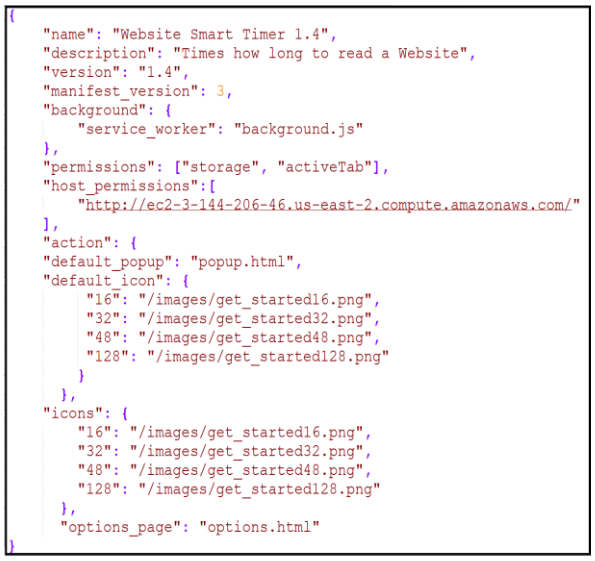


Figure 1. Overview of the solution

Above is a simple diagram to illustrate the connections between the various components of this web application. The web browser, the frontend, displays the timer and a time prediction as well as website information including average word size, number of worlds and readability score. Behind the display is the HTML, CSS, and javascript which make up the chrome extension. The chrome extension is created with manifest v3 (the code for which is below) and is capable of tracking the open active web tab. The chrome extension relays data to the web server, the python flask server, which sends data from the website to the chrome extension and stores data into a database. The database and web server are both hosted on AWS, Amazon web service, which is capable of running 24-7 waiting to respond to requests. This application is currently available on the google extension store for free download or use. The following text will describe each component in greater detail.



```
{
    "name": "Website Smart Timer 1.4",
    "description": "Times how long to read a Website",
    "version": "1.4",
    "manifest_version": 3,
    "background": {
        "service_worker": "background.js"
    },
    "permissions": ["storage", "activeTab"],
    "host_permissions":[
        "http://ec2-3-144-206-46.us-east-2.compute.amazonaws.com/"
    ],
    "action": {
    "default_popup": "popup.html",
    "default_icon": {
         "16": "/images/get_started16.png",
         "32": "/images/get_started32.png",
         "48": "/images/get_started48.png",
         "128": "/images/get_started128.png"
         }
      },
    "icons": {
        "16": "/images/get_started16.png",
        "32": "/images/get_started32.png",
        "48": "/images/get_started48.png",
        "128": "/images/get_started128.png"
      },
      "options_page": "options.html"
}
```

Figure 2. Screenshot of code 1

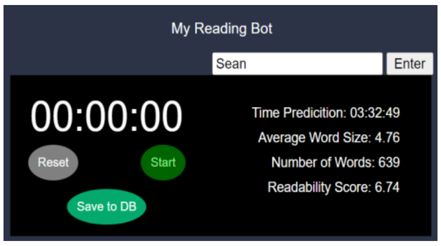
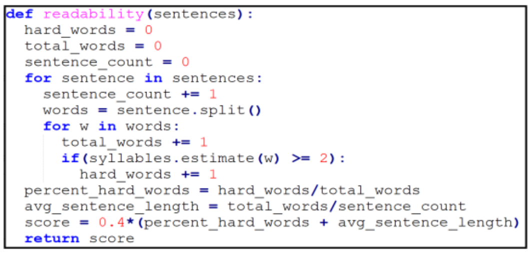


Figure 3. Reading bot

The frontend's display is created with HTML and CSS on a chrome extension popup which can be opened by clicking on the icon near the Google chrome extensions list on the top right of Google chrome. As shown above, the top portion of the popup allows users to input any username to save their specific data. The left side of the popup is a timer that tracks the time taken to read an article that will run with the popup opened or closed. Once finished, the user may click "Save to DB" which means save to database, saving their time and giving them a new time prediction. The right side of the popup displays the time prediction for a specific article, the average word size, the number of words, and its readability score based on the Gunning Fog formula.

Behind the display is the javascript which runs the timer component of the application. The timer feature displays a start/stop button, a reset button, and a save button. The reset button simply sets the time back to 0. The start/stop button interchange after every press starting and stopping the time. Because the initial start time is stored locally in chrome, the timer continues even after you close the popup. The save button, the most important component, takes the current time and sends it over to the python flask server component. This is done through flask's get and fetch methods which allows data to be sent over two devices. The javascript also updates the information display on the right side of the UI by providing the python server with website URL and receiving website information and a time prediction. In addition, the javascript will send a username to the python server so that each user's data can be unique and can be saved under one name. This ensures that each user's time prediction can be tailored to their reading speed and level.

Next, the web server and database components of this application are hosted on AWS (Amazon web service) [14]. The web server is written in python and consists of three main components: data storage, web scraping, and artificial intelligence. When the save to database button is pressed, the frontend sends data to the web server which will store values in two json files. The first json file is used to store username, websites, and website information so that common websites do not have to be web scraped twice, saving time. The second json file is used to store website difficulty and time taken so that the machine learning portion of the web server can be done more easily. When the popup for the chrome extension is opened, the frontend will send a request to the web server for website information and time prediction from the next two components of the python script. The website information is gathered by the web scraping script made with the python library Beautiful Soup. It takes a website URL provided by the frontend and separates the headings from the main text. It then counts the words, averages each word's size, calculates readability using the Gunning Fog formula, and packages it so that it can be sent back to the frontend.



```python
def readability(sentences):
  hard_words = 0
  total_words = 0
  sentence_count = 0
  for sentence in sentences:
    sentence_count += 1
    words = sentence.split()
    for w in words:
      total_words += 1
      if(syllables.estimate(w) >= 2):
        hard_words += 1
  percent_hard_words = hard_words/total_words
  avg_sentence_length = total_words/sentence_count
  score = 0.4*(percent_hard_words + avg_sentence_length)
  return score
```

Figure 4. Screenshot of code 2

Scikit-learn (also known as sklearn) is a Python library for machine learning that provides a variety of tools for tasks such as classification, regression, clustering, and dimensionality reduction [3]. It is built on top of NumPy and Pandas and is designed to be easy to use and efficient for large datasets [4][5].

Some of the key features of scikit-learn include:

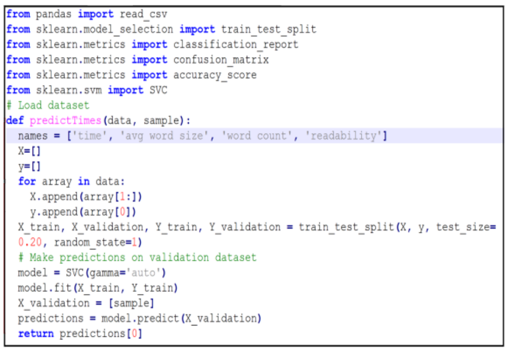
- A consistent interface for all models, making it easy to switch between different models and compare their performance
- A wide range of algorithms for classification, regression, clustering, and other tasks
- Support for both supervised and unsupervised learning
- Tools for evaluating the performance of models and selecting the best one
- Functions for preprocessing and transforming data, such as scaling and normalization.

This web application uses classification; a type of supervised learning in which a model is trained to predict a discrete label (e.g. "spam" or "not spam") for a given input. In scikit-learn, classification is implemented using a number of different classifiers, each with its own set of parameters and characteristics [15]. Some common classifiers in scikit-learn include logistic regression, decision trees, K-nearest neighbors, and support vector machines. This web extension adopted and implemented an SVM algorithm, which takes a set of labeled training data as input and outputs a classifier that can be used to predict the label of new, unseen data.
The key idea behind SVMs is to find the hyperplane in a high-dimensional feature space that maximally separates the different classes. This hyperplane is known as the maximum margin hyperplane, and the goal is to find the hyperplane that has the maximum distance from the nearest training data points of any class (these points are known as the support vectors).

To classify a new data point, the SVM simply assigns it to the class based on which side of the hyperplane the point falls on. If the point falls on one side of the hyperplane, it is assigned to one class, and if it falls on the other side, it is assigned to the other class.

SVMs have a number of advantages, including the ability to handle high-dimensional data and the ability to handle data that is not linearly separable by using the kernel trick. They are also relatively memory efficient, as they only require storing the support vectors, rather than the entire training set.

However, SVMs can be sensitive to the choice of hyperparameters and can be computationally intensive to train, especially for large datasets. They may also perform poorly on highly imbalanced datasets, where one class is much more prevalent than the other. The implementation can be seen below.



```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
# Load dataset
def predictTimes(data, sample):
  names = ['time', 'avg word size', 'word count', 'readability']
  X=[]
  y=[]
  for array in data:
    X.append(array[1:])
    y.append(array[0])
  X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_size=
  0.20, random_state=1)
  # Make predictions on validation dataset
  model = SVC(gamma='auto')
  model.fit(X_train, Y_train)
  X_validation = [sample]
  predictions = model.predict(X_validation)
  return predictions[0]
```

Figure 5. Screenshot of code 3

4. EXPERIMENT

4.1. Experiment 1

In order to effectively achieve the goal of improved time management, an accurate prediction must be generated by this model. In this experiment, the primary objective is to evaluate the effectiveness of this model. If the model can reasonably estimate read time given a limited number of entries, then this model is effective at helping with time management.

This experiment involved four subjects. These subjects were asked to train their personal models using the web extension and record its effectiveness on five different test articles of their preference. The model was trained with ten to twenty of their personal entries on articles of their choice with the only requirement being that they varied in length.

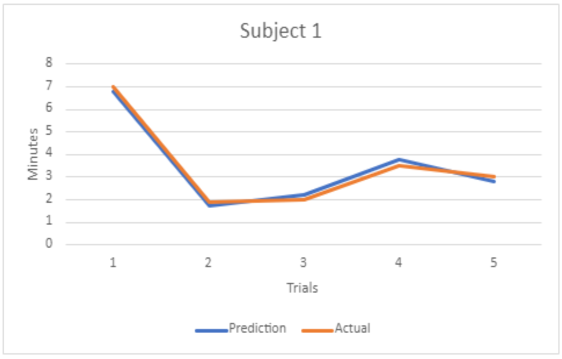


Figure 6. Prediction and actual of subject 1

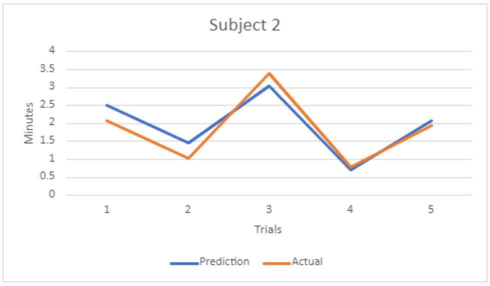


Figure 7. Prediction and actual of subject 2

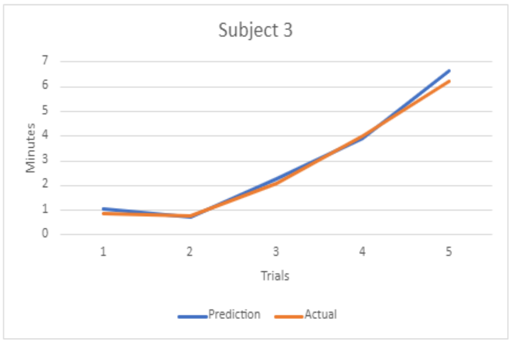


Figure 8. Prediction and actual of subject 3

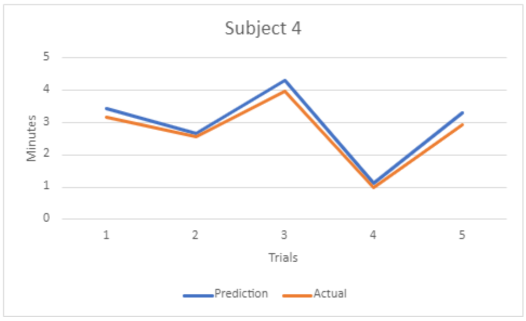


Figure 9. Prediction and actual of subject 4

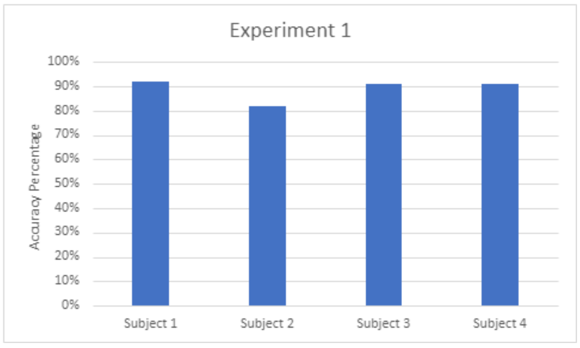


Figure 10. Result of experiment 1

As shown above, the model performed well considering the number of entries each subject inputted. The accuracy percentage for each subject was 92%, 82%, 91%, and 91% respectively resulting in an 89% percent accuracy in general. Subjects 1, 3, and 4 had predictions matching closely with actual time whereas there is a slight variance in subject 2’s predictions. The variance observed in subject 2 was likely due to the subject choosing to train the model with articles of small differences in length resulting in slightly skewed results when testing the model with large differences in lengths. Subjects 1, 3, and 4 had trained their models with entries ranging from one minute to six minutes, which means their models were better trained to adapt to large changes in length of articles giving them a more accurate prediction. It can be concluded that the model is effective for predicting read time given a limited number of entries and can be beneficial to time management.

4.2. Experiment 2

The next experiment was conducted to evaluate the model's effectiveness on a specific article dissimilar to the articles subjects used to train their personal models. The purpose of this experiment is to expose the model to different kinds of articles and evaluate the model's performance on a specific article across different individual users.

The article chosen had the following statistics that were taken into consideration when training the model: 5.21 average word size, 805 words, 8.02 readability score. Each subject had their models trained with personal data ranging from ten to twenty entries per subject

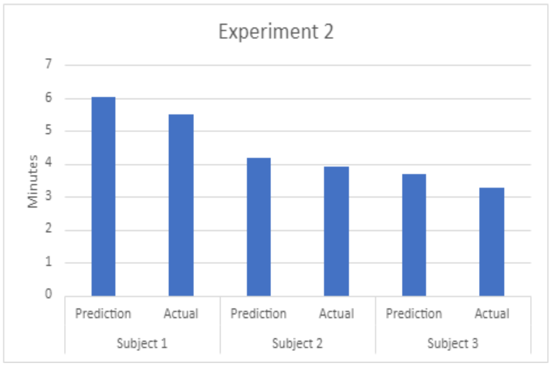


Figure 11. Prediction and actual of subjects in experiment 1

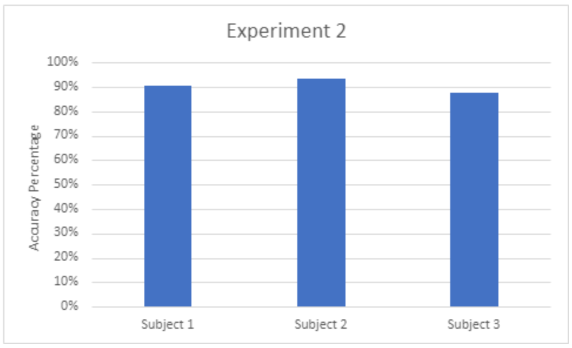


Figure 12. Result of experiment 2

As shown in the chart, the model was able to very accurately predict read time for a specific article given training data from different kinds of articles proving that the model is effective across every article. The accuracy percent was 91%, 94%, and 88% respectively resulting in a total of 91% accuracy in general.. All predictions, despite a small training set, resulted in differences between prediction and actual read times of no more than thirty seconds across all subjects. The reason for the larger than usual difference is likely due to the large difference in the type of article the subjects read. As shown in experiment one, variation was limited which was likely due to the fact that the subjects chose articles that were more similar to the articles they chose to train the model with. Overall, this experiment has proven that this model is effective across all types of articles and is capable of making accurate predictions despite change in article type.

These experiment results prove that this model is capable of both providing an accurate prediction for each user's articles as well as providing an accurate prediction for all kinds of articles. This addresses the challenge of time management by mitigating the risk of faulty time predictions. In experiment one, we have proved the accuracy of this model given a limited number of article entries [6]. We did this by having four subjects choose their own articles to use to train the model and test its effectiveness. This resulted in one significant finding: in order for the model to become more effective, it must be trained using articles of word size and readability. In experiment two, we have proved that the model is accurate for articles dissimilar with articles used to train the model further proving the effectiveness of using average word size, word count, and readability score in our model. We did this by having three subjects train their model with articles of their preference and having all subjects read the same article to compare predictions. Since all predictions were within thirty seconds of actual read time, it was concluded that the model is effective for all kinds of articles. Altogether, these experiments have shown that this web extension is effective for improving time management.

5. RELATED WORK

Irene Fernandez Monsalve, Stefan L. Frank and Gabriella Vigliocco introduced the surprisal theory into read time implementing machine learning in order to demonstrate accuracy [7]. They used lexicalized and unlexicalized models which both produced results hinting at a direct correlation between surprisal level and reading time. They concluded that surprisal had a significant effect on reading time. We used machine learning to attempt to guess rather than solve for reading time so in the short term, surprisal theory calculations will likely be more accurate and in the long term, machine learning will likely become more accurate.

Rui Liu, Huilin Peng, Yong Chen, and Dell Zhang presented a simultaneous news recommendation paired with a prediction time deep neural network capable of recommending news articles as well as providing a read prediction time [8]. They trained a neural network with large existing data sets and juxtaposed their HyperNews script with similar competitors concluding that their model outperformed general purpose models in news recommendation after the time prediction implementation. Because the model we implement concerns smaller datasets, our SVM (support vector machine) model will likely outperform HyperNews's neural network with respect to time prediction. Furthermore, our model is user specific and not generalized meaning on average, time prediction for each person should be fairly accurate. However, HyperNews's model is far stronger at predicting a general time for a collective group of people.

Srini Narayanan and Daniel Jurafsky implemented a Bayesian model, a system based on the Bayesian interpretation of probability, in order to accurately predict reading time expanding upon Narayanan and Jurafsky's 1988 hypothesis that human language comprehension can be modeled by treating human comprehenders as Bayesian reasoners [9]. Their fully trained model demonstrated that a Bayesian model of human sentence processing is capable of modeling reading time data from a syntactic disambiguation task. In general, Bayesian models are more commonly used for tasks that require a high degree of uncertainty or complexity, while SVMs, the system our model implements, are more commonly used for tasks that require a high degree of accuracy, such as text classification. Because of this, it is likely that an SVM model will outperform a Bayesian model when it comes to accurate reading time predictions.

6. CONCLUSIONS

Time management is important because it helps users make the most of their time and achieve users' goals. Our application addresses this problem by providing users with an accurate read time prediction based on article information and their historical read times. In order to achieve the best time management, an accurate prediction must be provided which is why in our experiments, two steps were taken to prove that our prediction model is accurate. The first step was to prove that the model is accurate for each individual user. We did this by letting subjects train and test the model with their personal data to evaluate the accuracy of the model. The second step was to prove that the model was effective for various kinds of articles outside of the ones used to train the model. We did this by letting subjects train their model with articles of their choice and having each subject test their model with the same article. The overall accuracy for both experiments was 90% indicating that this model performed well despite being given a small dataset and is therefore able to help with time management on a day to day basis.

In terms of accuracy, the biggest limitation to this application is its applicability in the early stages. It will usually take numerous inputs from the user in order to produce an accurate outcome. In addition, the model cannot make predictions after only a couple of inputs as the model would be too inaccurate. In terms of practicability, if the user shortens their reading speed naturally it might be difficult to produce accurate predictions based on old inputs; however since users can simply create another profile, this will not be a large problem. Lastly, it might become tedious to record every article users read using a start, stop, and save button; however it is necessary due to the fact that users may not be reading at all times and there is simply no other way to record article read time.

To address some of the limitations previously mentioned, the application may be adjusted to provide a generic time prediction during model training. The application may also be improved by switching to a stronger server where web scraping and time prediction can be done quicker. Finally, the UI could be changed so that the information displayed is customizable to the specific user.

REFERENCES

[1] Christanti, Viny, Dali S. Naga, and Cheria Benedicta. "Measuring Reading Difficulty Using Lexile Framework And Gunning Fog Index." Jurnal Teknik dan Ilmu Komputer (2017).

[2] Hearst, Marti A., et al. "Support vector machines." IEEE Intelligent Systems and their applications 13.4 (1998): 18-28.

[3] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

[4] Van Der Walt, Stefan, S. Chris Colbert, and Gael Varoquaux. "The NumPy array: a structure for efficient numerical computation." Computing in science & engineering 13.2 (2011): 22-30.

[5] McKinney, Wes. "pandas: a foundational Python library for data analysis and statistics." Python for high performance and scientific computing 14.9 (2011): 1-9.

[6] van de Schoot, Rens, et al. "Bayesian statistics and modelling." Nature Reviews Methods Primers 1.1 (2021): 1-26.

[7] Monsalve, Irene Fernandez, Stefan L. Frank, and Gabriella Vigliocco. "Lexical surprisal as a general predictor of reading time." Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics. 2012.

[8] Liu, Rui, et al. "HyperNews: Simultaneous News Recommendation and Active-Time Prediction via a Double-Task Deep Neural Network." IJCAI. 2020.

[9] Narayanan, Srini, and Daniel Jurafsky. "A Bayesian model predicts human parse preference and reading times in sentence processing." Advances in neural information processing systems 14 (2001).

[10] Carlini, Nicholas, Adrienne Porter Felt, and David Wagner. "An evaluation of the google chrome extension security architecture." 21st USENIX Security Symposium (USENIX Security 12). 2012.
[11] Macan, Therese Hoff. "Time management: Test of a process model." Journal of applied psychology 79.3 (1994): 381.
[12] Jackson, Valerie P. "Time management: a realistic approach." Journal of the American College of Radiology 6.6 (2009): 434-436.
[13] Graesser, Arthur C., Nicholas L. Hoffman, and Leslie F. Clark. "Structural components of reading time." Journal of Verbal Learning and Verbal Behavior 19.2 (1980): 135-151.
[14] Mathew, Sajee, and J. Varia. "Overview of amazon web services." Amazon Whitepapers 105 (2014): 1-22.
[15] Bisong, Ekaba. "Introduction to Scikit-learn." Building machine learning and deep learning models on Google cloud platform. Apress, Berkeley, CA, 2019. 215-229.