Digital Detox Assistant: A Deep Learning and NLP Based Digital Life Tracking System

Santosh Vasa, Rahul Reddy Baddam
vasa.s@notheastern.edu, baddam.ra@northeastern.edu
Khoury College of Computer Sciences, Northeastern University

Abstract— The time tracking software currently available to consumers, such as Google's Digital Wellbeing and Apple's Screen Time, do not fully picture the time a user spent digitally. They categorize "spent time" based on the bare mapping of each application to a class without processing the content the user consumes. This paper presents a novel and a more sophisticated approach to this problem by applying modern Natural Language Processing (NLP) strategies. It discusses how various techniques of NLP can be used and examines the results on self-prepared datasets. It concludes by describing the impact, scope and applications of this approach.

Keywords: Natural Language Processing, Term Matching, Semantic Similarity, Word Representations.

I. INTRODUCTION

We are all aware that humans are getting increasingly dependent on technology. As the pandemic took the world by a storm, most professions switched to digital/online mode. This switch during the pandemic resulted in blurry lines between productivity and leisure, due to which people started to lose accountability for their time spent online. Busy schedules, combined with distractions, can get in the way of finishing tasks. Providing the users with insights into their digital life can help them manage their time better.

Digital time tracking software such as Google's Digital Well Being and Apple's Screen Time provide basic information on how a user spends his time. Although this is useful, the software categorizes the just time based on the application used and its type. For example, Instagram is always classified as social or recreational regardless of the user's profession. A marketing lead, content creator's work might involve working with Instagram on a daily basis.

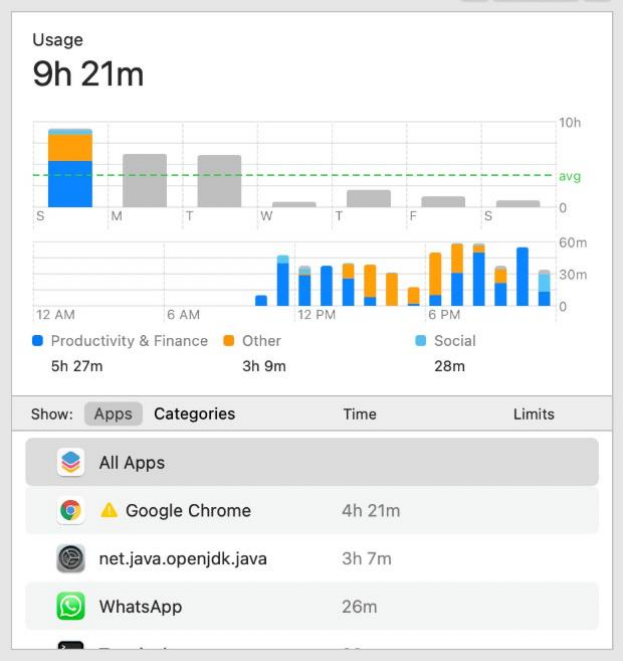

Fig 1. The present example of consumer software for time tracking. Shows how the software categorizes time based on the type of application.

The former can be improved by factoring in the user's content using more sophisticated and modern techniques. Our paper discusses a novel approach of applying Deep Learning-based natural language processing techniques to understand a user's retrieved web data personalized to the user's skillset.

A user's profession can be better understood by taking their portfolio as input. A portfolio such as their LinkedIn profile or their CV can be used to extract key skills and keywords. This extracted information is later fed to Wikipedia to get more text information related to the user's profession.

A user's web usage can be monitored by deploying a browser extension that retrieves scraped text information of every web page the user visits along with a timestamp.

We use modern NLP strategies to compare the relevance between the prepared user's knowledge base and the scraped text data each time a user visits a webpage. This relevance, along with the time spent on the webpage, helps understand a user's digital life productiveness. The information can later be formatted and shown to the user in a well-structured graphical representation.

Section two and three discuss the different approaches we experimented with, and factors weighted to determine a feasible method for the mentioned problem. We observed that uncommon tokens or words hold the most semantic information associated with the user's profession or the topic of the web page. These uncommon words are later fed to deep neural networks that consider the context for vectorizing tokens into embeddings. The output embeddings of both knowledge base and fetched web data are later matched using feature engineering, a similarity metric and a thresholding factor. The whole process can be termed as a binary document classification/ relevance problem uniquely applied to this application.

Section four of this paper talks about the experiments and analysis observed. We used a contextualized word representation model trained on a 1-billion-word dataset, whereas our classification model is analyzed on multiple carefully self-prepared datasets. We experimented with the web usage of two individuals with different professions and displayed the results.

II. RELATED WORK

Named Entity Recognition: Named Entity Recognition or NER tags named entities in a given sentence. Entities are nouns in a sentence such as organizations, people or locations. Modern Deep Learning models such as BERT and GPT series achieve promising results in the Glue Benchmark. Although these methods are encouraging, they require a huge corpus of labeled data to train and perform skill or keyword extraction on text. These methods also overfit and do not consistently perform when used for larger and wider skillsets of professions.

Text Summarization: A text summarization system compresses large text data into a few summary lines. This summary ideally retains as much semantic information as possible. It addresses the problem by capturing the semantic density in the text and tags the sentences with the most information. Sequence to Sequence encoder-decoder architectures reconstructs the complete text data into new sentences instead of tagging the existing sentences. We observed this technique is not useful as it discards keywords that give key insights into the user's profession.

Word Representations: As discussed earlier, words/text cannot be processed by computers in the raw format. These words must be converted into numbers for computers to process and understand.
One-hot vectorization can represent the words by choosing a vocabulary/ dictionary with a limited size.

Libraries like TF-IDF and count vectorization also use similar approaches to represent words. These approaches are static and do not represent any semantic or polysemic information of the word.

The conversion approach of the terms plays a key role in the further process as the numbers should semantically represent the word to capture and make decisions based on meaning. Apart from the simple representation approaches mentioned above, the other techniques can be categorized into two methods.

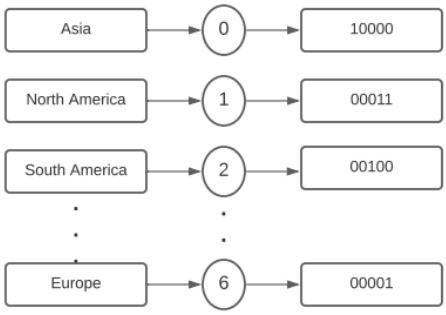

Fig 2. One Hot Representations

Statically Distributed Word Representations: These word representations are trained on a huge corpus of text data to retain the semantic information of the words by looking at their usage in sentences in the corpus. The following example shows how the representations capture the similarities of two different words.

Asia is the largest continent.
Antarctica is the coolest continent.

The words Asia and Antarctica are similar in their usage. The word representation algorithms capture this information. One of the most popular approaches to this problem is Word2Vec by T. Mikolov. Word2Vec uses two algorithms, the Skip Gram model and the CBOW - Continuous Bag of Words model. The Skip Gram model is trained to predict surrounding words when given a central term, whereas a CBOW model is trained to predict the main word when given the surrounding words.
The following shows how the two words in the above example are represented using the distributed word representations.

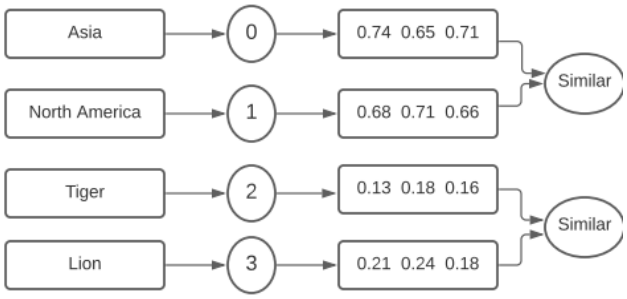

Fig 3. Static Distributed Word Representations

GloVe vectors and FastText bought more improvements to this process. Although these methods capture the similarities among the words, it fails to capture the context during the inference. The static distributed representation methods do not factor in the surrounding word information to represent the words. Not factoring in the context introduces us to the problem of not addressing polysemy.

Dynamic Deep Contextualized Word Representations: Context-based word representations address polysemy problems by considering the surrounding tokens even during the inference. Vectorizing and distinguishing the same spelled tokens based on their usage in the sentence retains more semantics after the conversion. If a contextualized word representation is used, the embeddings generated for the same word "minute" in the following sentences will have a large Euclidean distance between them.

Take a minute before asking any doubts.
They are asking a minute amount of money compared to others.

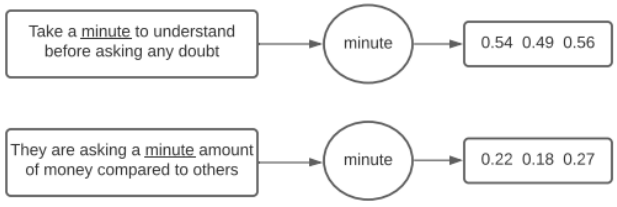

Fig 4. Dynamic Deep Contextualized Word Representations.

III. ARCHITECTURE

The architecture of the data flow can be majorly divided into three segments:

1. Information retrieval
2. Feature Extraction
3. Classification (Matching)

3.1 Information Retrieval:

The system requires data from multiple unstructured text sources, which is later fed to cleaning and processing to extract features. The process can be further categorized into two subparts named after their processed outputs - Knowledge Base and Fetched web text.

Knowledge Base: This is the auto-prepared system information on the user. This part takes a LinkedIn profile, a Curriculum Vitae or a list of keywords related to the user's profession as input. It generates more associated text by extracting passages of text from the internet. The generated text acts as a knowledge base to identify relevant web pages in the further process.

In our code, we take a list of skills and keywords as input from the user and feed it to Wikipedia extraction. These keywords can be anything significant and explain the user's profession. The system then extracts each keyword's overviews of text data from this information web page.

The overviews are then concatenated into a single information base called knowledge base. The knowledge base is crucial, unique, and associated with each user and the reason for making personalized decisions of relevance.

$$Knowledge\ base = \Sigma_i\, W(S_i)$$

W is a function that takes a skill word and extracts a summary (Overview). S is the list of keywords enumerated using i.

Fetched Web Text: The web history of each individual consists of a lot of web pages that a user visits. Retrieving the text data from the web pages and understanding the information will facilitate the system to make decisions on its relevance to the user's profession. The retrieval process can be done using publicly available free-to-use scraping libraries such as Beautiful Soup and Selenium Automated Testing Software. We observed that Selenium extracts more text data compared to Beautiful Soup and recently released a headless browser version. The process involves h tags, p tags, span tags, title and URL extraction associated with a weight matrix. Title, h tags are given more importance (weight) compared to span and p tags. The retrieved data is then saved along with its weight matrix and associated with the web page.

The system now has all the information to decide if the fetched web text and the knowledge base. The webpage will be classified as relevant if both the data match by some degree. Feature Extraction facilitates the process of understanding the relevance.

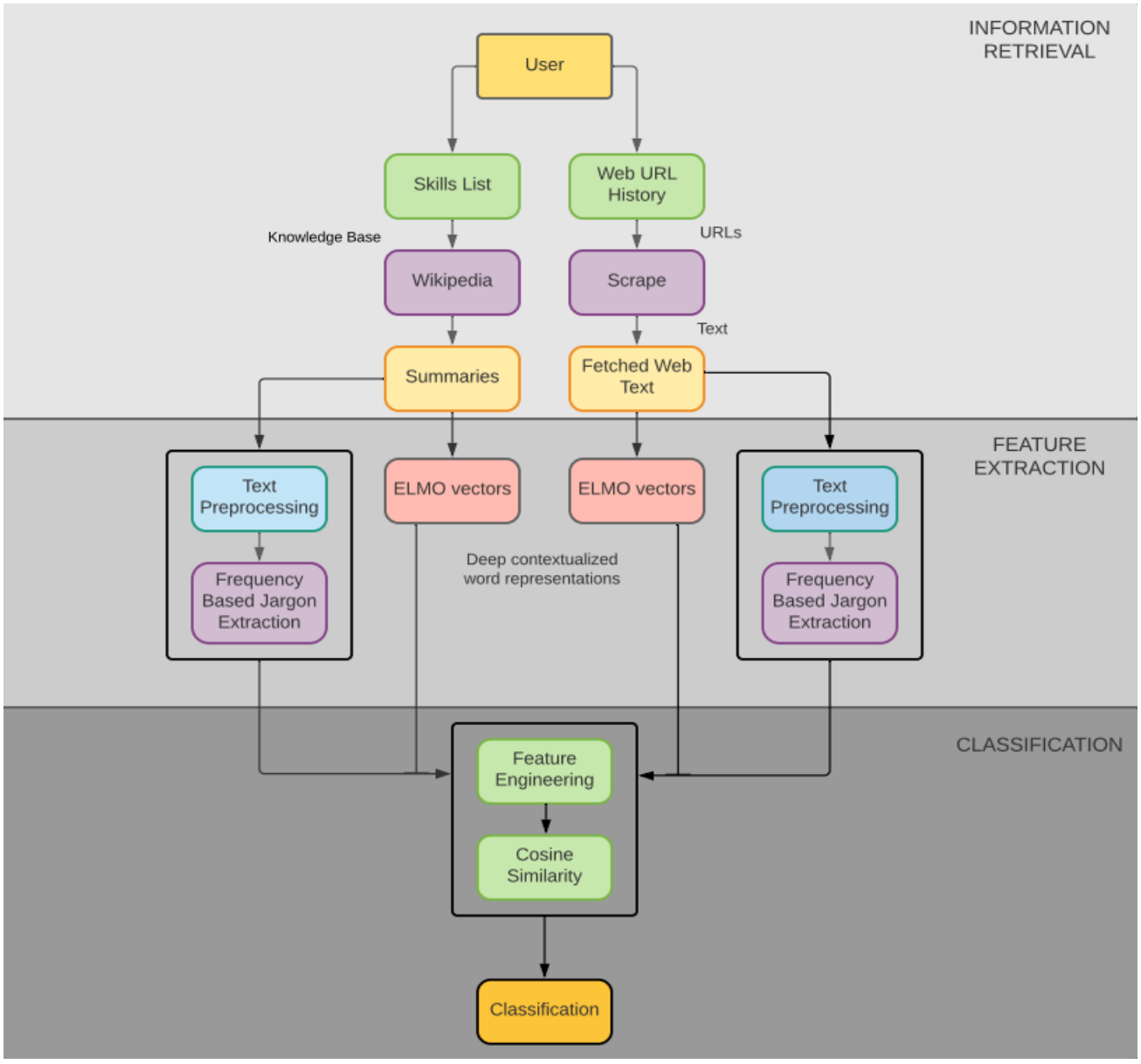

Fig 5. Model Architecture and Flow.

$$\text{webpage text data} = \Sigma_i \, T[i] * (ex[i])$$

T represents the weight matrix of each tag; ex represents the complete text scraped from a webpage; i represents the extracted tag.

3.2 Feature Extraction:

This process involves three sub-parts - Text Cleaning, Term matching and Word Representation.

Text Cleaning: The text is filtered out in the first phase by removing the punctuation and converting it into lower case text. The resultant text is then tokenized into words. The words are then lemmatized or normalized to their root words which facilitates better term matching. This process is carried out both fetched web text and knowledge base summaries.

Frequency-based uncommon word extraction:
We observed that uncommon words (Based on frequency they occur in general text corpus) give the most insight into the web page's topic or the user's profession. The most crucial part of this whole system is to identify the words that most represent the user's profession and the words that most describe the topic of a web page. This process can be termed as skill word or keyword extraction. We used the vocabulary generated by BERT - Google. The vocabulary index contains more than 30,000 tokens ordered by the frequency of their occurrence in general texts. BERT orders the frequency of words based on text from all of Wikipedia and Brown Corpus. For example, the root word "Programming" is uncommon and lies in a high index, whereas the root word of "walking" will be more common and will lie in a lower index in the BERT vocabulary. Both the resultant text cleaned knowledge base and the web text data are passed through this block, as shown in figure 5. The result is two sets of words that are uncommon and hold information on a person's profession and the web page's topic.

Term Matching: After text cleaning and uncommon words extraction, we are left with words that are semantically dense in core information. One approach is to match these words and factor the ratio of terms matched in the knowledge base and fetched web text to all terms. This approach leads to encouraging results (more in chapter 4). Although this approach is simple and achieves good results, it is evident that it fails to interpret a few patterns the text language holds. We introduce you to the problems that exist in this approach. Semantic similarity and Polysemy are not taken into account while matching them solely based on text.

Semantic similarity is where two different words have a similar meaning. For example,

"Your kid is very good at soccer." and
"Your child is playing in the garden."

the underlined words are represented by different characters but have a semantic similarity. Simple term matching will fail as the words are not the same.

Polysemy is where two words or phrases represented with the same characters have different semantics. As explained in chapter 2, the polysemy problem needs surrounding word tokens to understand how the word is used in the sentence. For example,

"He won the second prize."
"He took 100 seconds to finish the race."

The underlined words are represented with the same characters but have different semantics. Both Polysemy and Semantic Similarity are not addressed when using basic term matching.

Embeddings from Language Model:

Using ELMo addresses both issues. ELMo outputs a contextualized feature representation by taking the input word sequence as input.

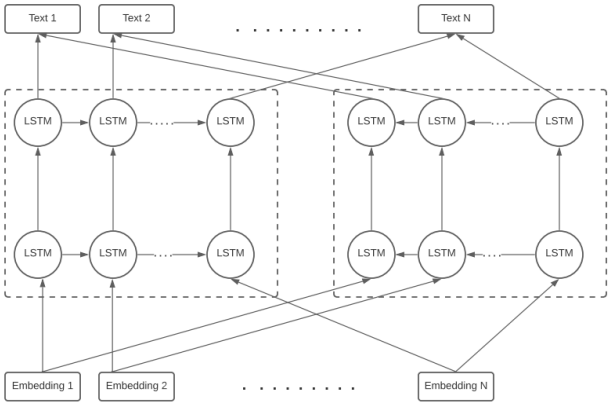

Fig 6. ELMo Architecture

The figure above shows that ELMo is trained on two-layered bidirectional LSTM layers. In the bi-directional LSTM layer, also called as BiLM layer, the forward pass is executed as just another language model by considering the previous tokens of the sentence. In contrast, the backward layer computes tokens from the future. This way, ELMo acts as a function of the complete input sequence and vectorizes the word based on it. ELMo can be used for all NLP applications that require word representations as inputs. It is widely used for tasks such as sentiment analysis, text translation, question answering, named entity recognition and more.

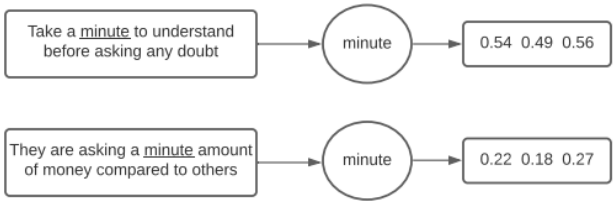

Fig 7. ELMo embeddings example

ELMo outputs a 1024-dimensional vector, also called as embedding, for each toke given in the input sequence. Using ELMo vectors instead of one-hot representation or

distributed word representation helps us solve the problem of semantic similarity and polysemy to an extent.

3.3 Feature Engineering and Classification:

After Text Cleaning and Feature Extraction, the resultant data is a vector representation (1024 embedding) of cleaned uncommon words found in the knowledge base and fetched web text. As shown in fig 5, the data is sent to the next block, which outputs a similarity score.

This block processes the data outputted by the ELMo network. All the matched terms and the terms with similar ELMo representation are taken and computed in a function to retrieve a relevance score. The score is weighted on the following formula where the word frequency of the term in the knowledge base and the fetched web text is considered and weighted.

$$match\ score = f1 * w1 + f2 * w2$$

f1 – represents the count and frequency of words matched from the knowledge base. f2- represents the count and frequency of words matched from fetched word text. w1 and w2 are two weights to balance the importance in the score.

The similarity metric used here is cosine similarity. It measures the angular value/angle between two n-dimensional vectors (1024 dimensions in our case). Lower the angle, more the similarity.

$$similarity(v1, v2) = \frac{v1.v2}{||v1|| * ||v2||}$$

$$similarity(v1, v2) = \frac{\sum_{i=1}^{n} v1_i\ X\ v2_i}{\sqrt{\sum_{i=1}^{n} v1_i^2}\ X\ \sqrt{\sum_{i=1}^{n} v2_i^2}}$$

The vectors v1 and v2 represent the two resultant ELMo vectors outputted by the knowledge base and fetched web text inputs.

We get the binary classifier by thresholding the output given from feature engineering and cosine similarity.

$$Relevance\ Score = \frac{matched\ score}{total\ number\ of\ fetched\ uncommon\ words}$$

IV. EXPERIMENTS AND ANALYSIS

We have self-prepared two datasets with around 130 entries of different professions: each spread around various time periods in a day. The data is a synthetic representation of how a user might visit web pages in a day at particular time slots.

The two datasets represent a day's web history of two individuals with different professions, namely a computer science engineer and a medical practitioner.

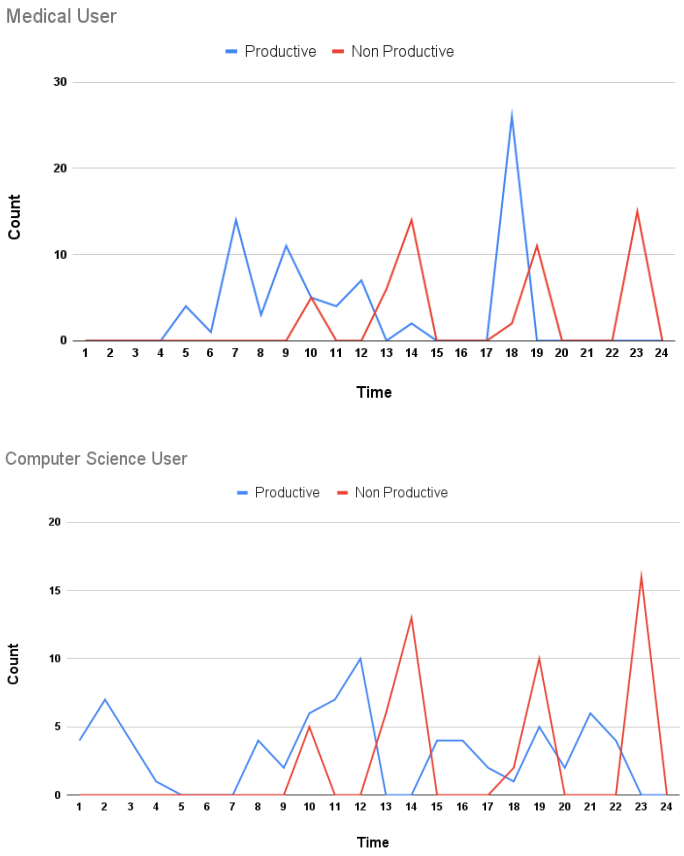

Fig 8. Results on two selected professions. (a). Medical User's productivity chart. (b) Computer Science User's productivity chart.

The two datasets represent a day's web history of two individuals with different professions, namely a computer science engineer and a medical practitioner.

Each dataset is associated with a list of skills/keywords prepared accordingly based on their profession. The following snippet shows the list of skills given as inputs.

medical user = ['patient' ,'health', 'pandemic', 'chemotherapy' , 'therapy' , 'cancer', 'disease' , 'diagnostic' , 'vitals' , 'vaccination', 'Pathology', 'Anesthesiology' , 'physicians' , 'pharmacy' , 'Cardiology', 'cardiovascular' , 'Surgery' , 'nurses', 'doctor' ,'Allergy', 'Clinical', 'Immunology', 'Dermatology', 'Diagnostic' ,'Radiology', 'diagnose', 'Emergency Medicine' , 'Endocrinology', 'Metabolism', 'diabetes', 'menopause', 'hypertension', 'thyroid diseases', 'cholesterol', 'Gastroenterology', 'pharmacology', 'biochemistry', 'hospital', 'genetic diseases' , 'Microbiology' , 'Nephrology' , 'Neurosurgery', 'Orthopedic Surgery', 'Psychiatry' ,'Oncology']

Computer Science Engineer = ['Deep Learning', 'Convolutional Neural Networks', 'TensorFlow', 'Machine Learning', 'Algorithms', 'Computer Vision', 'Deep Reinforcement Learning', 'Computer Science', 'Python', 'MATLAB', 'Recurrent Neural Networks', 'Django', 'Java', 'Keras', 'PyTorch', 'Neural Networks', 'Edge Computing', 'OpenCV', 'Scikit-Learn', 'Natural Language Processing', 'Object Detection', 'Reinforcement Learning', 'Generative Adversarial Networks']

These skills and entries in the dataset are passed through the Information Retrieval, Feature Extraction and Classification pipeline. The binary classification output categorizes each entry, and the results on both datasets are shown in fig 8.

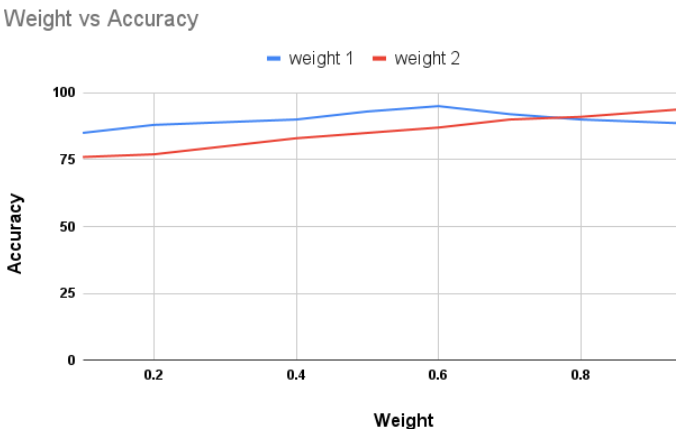

Fig 9. Shows hyperparameter tuning of two weights used in the feature engineering process.

As mentioned above, the following function finds the score and the weight parameters can be tweaked.

$$match\ score = f1 * w1 + f2 * w2$$

f1 – represents the count and frequency of words matched from the knowledge base. f2- represents the count and frequency of words matched from fetched word text. w1 and w2 are two weights to balance the importance in the score.

The results show that 0.6 (fetched web text) count and 1.0(Knowledge base text) works best with our datasets.

Accuracy:

The two datasets contain 130 entries each. We have labeled them manually and have evaluated the accuracy. We were able to get about 95% on the Computer Science skill set and an accuracy of about 93% on the medical skill set. The entries we made to the dataset mostly contain medium to large text passages.

Dataset	User Profile	URL entries	Key words in Skill Set	Accuracy
1	Computer Science Engineering	123	23	95%
2	Medical professioonal	134	45	93%

Fig 10. Table with Accuracy numbers.

We observed that the classification fails when the webpage has very little text. Fewer text data makes it very difficult to understand the topic on which the webpage is based. Although we factor in the number of words on the web page while calculating the score, fewer keywords and uncommon words might not even match the words existing in the knowledge base. The performance on these web pages can be improved by using a database that contains information on the type of websites. The website name can be extracted from the URL, which can further be checked in the manually prepared dictionary map of popular websites.

Generating summaries from all the skills in the skill list/keyword list is also difficult considering multiple pages have the same name on Wikipedia. This can be improved by improving the search on Wikipedia by understanding the relevance between each skill. A computer science engineer would more like have python programming as a skill and has nothing to do with the real python (animal).

The pipeline also fails when the website needs user authentication to display text. This wouldn't occur in a real-time application as the user mostly uses a web page after logging in to the site.

Latency:

The current system uses ELMo to generate word embeddings. ELMo uses two layered Bidirectional LSTM layers which process input words serially. This results in slow processing as the time steps require outputs generated in the previous time steps. Output embedding generation thus requires serial processing of all the time steps.

Processor	Network	Embedding Size	Tokens per second
Intel i7 (8500U)	ELMo	1024	~63

Fig 11. Latency table

Transformer and attention-based networks process words parallelly, exploiting the high computing power of General or Graphic Processing Units and Tensor Processing Units.

The model we used is neither quantized nor pruned. It involves computing with float32, which doesn't add any accuracy gains. As we did not perform any pruning activity, The model currently conducts many redundant operations. By Quantizing and Pruning, the network's latency can be further improved by order of multiple times.

Clustering:

Each web page should have a fixed size embedding to cluster into multiple meaningful groups. The ELMo network gives an embedding for each uncommon word present in the fetched web text. As the number of uncommon words on a web page is not fixed, we have followed the following approach. We limited the uncommon word extraction to 50, leaving a fixed embedding size of (50,1024) vector outputted by the ELMo network. The top 50 most frequent words are added to the webpage embedding by performing average pooling on word embeddings with multiple word occurrences. The Fig 10. Shows the clustering flow we used.

We used DBSCAN and K-Means to perform the clustering. We tweaked the Epsilon values and the min sample size, but the process showed no promising results. One of the reasons is that the size of each web page embedding is too large (50 * 1024 = 51200). Using a smaller last layer size can help us compress the differentiating features into smaller vector sizes. Ideally, we expected DBSCAN to form 4 clusters with work, shopping, entertainment, and travel on the datasets we have given. This was not reflected in the results as it made clusters even inside the four subdivisions we expected.

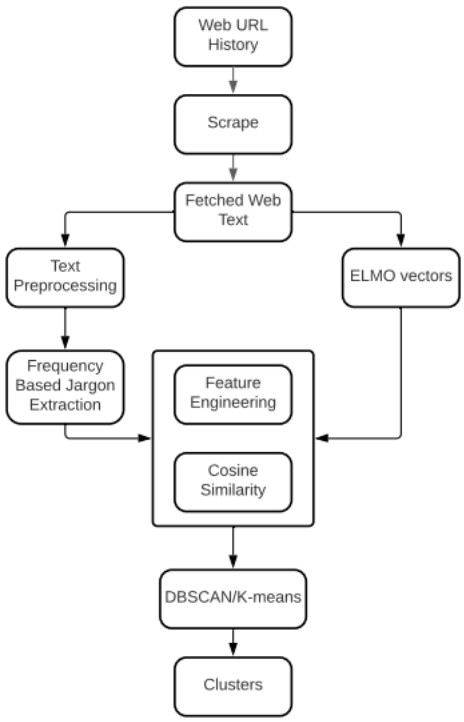

Fig 12. Clustering Flowchart

V. CONCLUSION

The uncommon words present in the knowledge base and fetched web text is semantically dense in explaining the topic of the content. Exploiting this text data results in high accuracy on the digital time tracking binary classification task. The neural network and overall flow's latency and throughput can be further improved by using better neural architectures such as transformers, quantizing and pruning the network. Clustering the web pages does not result in expected results with the embedding size of 51200. This can be improved by using fewer nodes in the last layers. The binary classification outputs alone work as a full-fledged time tracking system by classifying each web page's relevance to the user visits.

VI. CITATIONS

1. Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018, March 22). *Deep contextualized word representations*. arXiv.org. Retrieved December 15, 2021, from https://arxiv.org/abs/1802.05365

2. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019, May 24). Bert: Pre-training of deep bidirectional Transformers for language understanding. arXiv.org. Retrieved December 15, 2021, from https://arxiv.org/abs/1810.04805

3. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013, September 7). Efficient estimation of word representations in vector space. arXiv.org. Retrieved December 15, 2021, from https://arxiv.org/abs/1301.3781

4. Glove: Global vectors for word representation. (n.d.). Retrieved December 15, 2021, from https://nlp.stanford.edu/pubs/glove.pdf

5. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017, December 6). Attention is all you need. arXiv.org. Retrieved December 15, 2021, from https://arxiv.org/abs/1706.03762

6. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020, July 22). Language models are few-shot learners. arXiv.org. Retrieved December 15, 2021, from https://arxiv.org/abs/2005.14165

7. Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017, June 19). Enriching word vectors with subword information. arXiv.org. Retrieved December 15, 2021, from https://arxiv.org/abs/1607.04606

8. Staudemeyer, R. C., & Morris, E. R. (2019, September 12). Understanding LSTM -- a tutorial into long short-term memory recurrent neural networks. arXiv.org. Retrieved December 15, 2021, from https://arxiv.org/abs/1909.09586

9. Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018, March 22). Deep contextualized word representations. arXiv.org. Retrieved December 15, 2021, from https://arxiv.org/abs/1802.05365

10. Yadav, V., & Bethard, S. (2019, October 25). A survey on recent advances in named entity recognition from Deep Learning Models. arXiv.org. Retrieved December 15, 2021, from https://arxiv.org/abs/1910.11470

11. Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assef, Saeid Safaei, Elizabeth D. Trippe, Juan B. Gutierrez, Krys Kochut, Text summarization techniques: A brief survey - arxiv. (n.d.). Retrieved December 15, 2021, from https://arxiv.org/pdf/1707.02268.pdf

12. Wang, D., Lu, X., & Rinaldo, A. (2019, December 4). DBSCAN: Optimal rates for density based clustering. arXiv.org. Retrieved December 15, 2021, from https://arxiv.org/abs/1706.03113

13. Xie, Y., Shekhar, S., & Li, Y. (2021, October 10). Statistically-robust clustering techniques for mapping spatial hotspots: A survey. arXiv.org. Retrieved December 15, 2021, from https://arxiv.org/abs/2103.12019

14. Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large, annotated corpus for learning natural language inference. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics.

15. Jason Chiu and Eric Nichols. 2016. Named entityrecognition with bidirectional LSTM CNNs. In TACL.

16. Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2020, February 9). Albert: A lite bert for self-supervised learning of language representations. arXiv.org. Retrieved December 15, 2021, from https://arxiv.org/abs/1909.1194