Group 11 - Project Final Report: *Is This Website Safe for Kids?*

Authors: Chen Anqi, Chen Su, Joshua Chew Jian Xiang
Email: {e0324117, e0323703, e0406350}@u.nus.edu.sg

Abstract

This project investigates several machine learning models to determine which is the most appropriate classifier for a classification task to determine the degree of safety of a website to kids by only looking at the URLs. We place greater importance on the identification of URLs that lead to websites that are unsafe for children, like pornography or extreme violence. This project uses Naive Bayes (NB) and Logistic Regression (LR) classifiers as baseline models, and compares their performance with Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) models. Variants of the models are implemented using TF-IDF and word embeddings as features. The performance of each model is evaluated using the AUC-ROC score with One-vs-One configuration for multi-class targets. We have empirically found RNN with BiLSTM to be the highest-performing model for this particular task.

Introduction

Children, at an age of formative growth and possessing high impressionability, should be protected from websites that have content potentially harmful to their mental well-being. In this information age where kids are flooded with information on the Internet from a young age, it is necessary to have protective applications for them to avoid potential traps and viewing of inappropriate online content. Traditionally, this is achieved through blacklists which block out the webpages identified to be unsafe. However, this method is neither flexible nor exhaustive enough to detect newly emerged websites. Hence, we feel that a machine-learning classifier would be very helpful in filling up the gap.

Content-based classification is a possible solution, but the web page content can change dynamically which increases the difficulty of correct classification. Also, the entire HTML file together with the objects it contains need to be fetched in order to perform content-based classification, which can be slow and expensive. Hence, we decide to look into URL-based website classification, because it need not wait for the full page to load, thus would be faster as compared to content-based classifiers. Time-efficiency is a much desired property, as the categorization of visited websites should be completed as soon as possible to ensure a smooth Internet surfing experience. With a well-designed model and fine-tuned hyperparameters, URL-based classifiers can be widely applied in browsers and parental-control softwares to detect potentially unsafe websites, achieving both fast speed and high accuracy.

For the purposes of this project, we categorize all web pages into 4 categories: *Adult*, *Potentially Unsafe*, *Safe*, and *Targeted at Kids.* We aim to build a machine learning model that can discern web pages into correct categories based on URL only. In particular, the correct detection of the *Adult* pages is a more important task compared to the other three categories since the negative impact of letting even a single adult website to pass and be shown to kids is extremely huge. Therefore, we focus the most on improving the accuracy of predicting *Adult* websites, aiming to minimize false negatives in this class.

Related Work

Classification of education-related sites using URLs. Kan's paper on web-page categorization without the web page focuses on accessing the feasibility of using URLs for classification [3]. He makes use of education-related labels like *course*, *faculty*, *project* and *student*, which is a different area of focus from our study. The paper concludes that the performance of URL-based features exceeds some document-based features. This gives us confidence that URLs can be applicable for our context as well.

Language classification based on URLs. Baykan et al.'s study [1] provides a comprehensive evaluation of techniques for feature engineering and machine learning algorithms to identify the language of a web page based on the URL. The paper tries out features such as word, n-grams and other custom-made features (e.g. occurrences of words in various dictionaries), and algorithms such as Support Vector Machine, Decision Trees, Naive Bayes, and Maximum Entropy. Again, this paper proves that there is a significant amount of information that we can extract from URLs. Although the objective of the study is different, its analysis of features and models have been useful in our project's selection of models to be tested for the context of the safety of websites to children.

Identifying kids-specific web pages using URLs. Rajalakshmi et al.'s paper [6] performs a task that is similar to this project, but focuses only on binary classification of URLs to identify websites that are kids-specific. This paper arrives at the conclusion that CNN combined with Bidirectional Gated Recurrent Unit (BGRU) gives the highest accuracy for this classification task. While this paper suggests several useful neural network models, we feel that such binary classification is more relevant in web page ranking or recommendation systems. For tasks like safety censoring, such categorization is too blunt, and those models may not perform well on detecting unsafe websites for kids. Therefore, in our study, more categories are introduced and the focus of detection also changes.

Detecting malicious websites through URLs. Le et al.'s paper [5] proposes a CNN model to detect malicious URLs which host unsolicited content and are used to perpetuate cybercrimes. It makes use of both characters and words from URL strings to capture semantic features, yielding better performances than Bag-of-Words features. This study focuses on finding websites that commit cyber-crime, which is again different from our project's focus on finding websites that are harmful but are not necessarily illegal. Therefore, the same CNN model does not necessarily perform well for our tasks. However, inspirations of possible models and features can be drawn.

Methodology

Selection of Dataset

We retrieve the URL dataset [8] from Kaggle which is adapted from the DMOZ directory. It contains 1.56 million URLs divided into 15 categories. This dataset provides a fairly comprehensive URL coverage for our model training.

Relabelling and Re-proportioning of Data

The existing labels are not tailored to the purpose of our study, and thus they are recategorized into the 4 labels we define as shown in Fig. 1. The remapping is based on common knowledge.

Original Category	**New Category**
Adult	Adult
Arts, Games, News, Shopping, Society, Recreation	Potentially Unsafe
Business, Computers, Health, Home, Reference, Science, Sports	Safe
Kids	Targeted at Kids

Fig. 1. Remapping of labels

After remapping, there are 35,325 instances in the *Adult* class, 765,105 instances in the *Potentially Unsafe* class, 716,366 instances in the *Safe* class, and 46,182 instances in the *Targeted at Kids* class.

Three versions of training and test dataset are prepared via randomized downsampling, two of which are unbalanced datasets with the ratio of number of instances in each class to be 1:15:15:1 and 1:5:5:1. The other version is balanced with an equal number of instances in each class. In the balanced dataset, all other classes are downsampled to the scale of the smallest *Adult* class, resulting in a total of 35,325 × 4 = 141,300 instances in training and test data.

Preliminary experiment is conducted for Naive Bayes and Convolutional Neural Network models to determine which version of the dataset gives better accuracy. The results show that the balanced dataset performs better for both models and thus is used for all other models. Details of the preliminary experiment is in the **Evaluation** Section.

URL Token Parsing

Each URL in the dataset is splitted into a list of its constituent tokens. The method used for tokenization is adopted from the recommendations from Kan's paper [4].

First, a baseline tokenization is conducted by splitting a URL by its non-alphanumeric characters. For example, an input URL "http://www.thebookrecycler.bizland.com/" will be split into ['http', 'www', 'thebookrecycler', 'bizland', 'com'].

Notice that the URL above also contains words concatenated with one another without any space in between , like 'thebookrecycler'. We make use of *recursive segmentation* using entropy reduction to split such tokens into their constituent words.

We regard the entropy of a token to be equivalent to its Information Content (IC). To assist in the calculation of IC, we adopt a Kaggle dataset [7] based on the *Google Web Trillion Word Corpus* which provides the counts of the 333,333 most commonly used English words in the World Wide Web. If a token is found in the corpus, the IC of the token is calculated as

$$IC\,(token) \;=\; -\,log\,(\frac{count(token)}{totalcount})$$

where *totalcount* refers to the sum of the counts of all the words in the corpus. If the token is not found in the corpus, the IC of the token is made very large and proportional to the length of the token, allowing the parser to favour smaller partitions.

A token can be split into a set of partitions if the sum of the partitions' IC is lower than the token's entropy. Hence, we searched each token for an index such that partitioning by that index lowers its entropy. The resultant partitions are treated as individual tokens and recursively partitioned in a similar way. Such an algorithm might not guarantee a partitioning with the global minimum entropy, but it provides a satisfactory local minima in $O(n\ log\ n)$ time.

After processing the URL through the above mentioned partitioning scheme, the resultant list of tokens is ['http', 'www', 'the', 'book', 'recycler', 'biz', 'land', 'com'], where 'thebookrecycler' has been successfully split into its constituent words.

Training and Testing of Models

The Train-Test Split ratio is 8:2. In total, five models are evaluated, which include Naive Bayes (NB), Logistic Regression (LR), Convolutional Neural Network (CNN), (Bidirectional) Recurrent Neural Network (RNN), and a combination of CNN and RNN. Two types of features are tried out: Term-Frequency Inverse Document Frequency (TF-IDF) and Word Embeddings.

Hyperparameters of each model is tuned using: 5-fold cross validation with grid search. Early Stopping is adopted to mitigate overfitting.

Area under the receiver operating characteristic curve (AUC-ROC) score with One-vs-One (ovo) configuration for multi-class targets is used as the evaluation metrics. We use ovo AUC-ROC for two main reasons. First, it is more suitable for balanced datasets like the one we use. Second, the ovo variant computes the average AUC-ROC of all possible pairwise combinations of the four prediction classes, hence is more comprehensive.

A model is considered to have better performance if it obtains high AUC-ROC scores on test data across all four classes. In particular, a high score for the *Adult* class is prefered.

Word Embedding Preprocessing

Global Vector for Word Representation (GloVe) [2] is an unsupervised learning algorithm to obtain vector representation of words that keeps similar words closer to one another. In our project, we use GloVe Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50-dimensional) pre-trained embeddings. To link parsed URL tokens to their respective word vectors, we first map the distinct tokens to a continuous range of integers, and then build a V×k embedding matrix (whereby V = cardinality of corpus vocabulary, i.e. number of distinct tokens in all input URLs plus <PAD> and <UNK>; k = embedding dimension, which is chosen to be 50) with row indices corresponding to the tokens' mapped integers.

Index 0 of the embedding matrix is reserved for <PAD>, which is represented as a zero vector. Index 1 of the embedding matrix is reserved for <UNK>, which is represented as the average of all GloVe word vectors. We do not use a zero vector for <UNK> despite it being a common practice, because we feel it is more accurate to distinguish <UNK> from <PAD> as <UNK> also carries some unique information.

After this step, we are able to achieve a decent coverage of 89.04% of the vocabulary in the training corpus with our embedding matrix built from the above-mentioned GloVe 50d embeddings.

Evaluation

Preliminary Experiment on Unbalanced v.s. Balanced Datasets

To decide which proportion distribution of the 4 classes is the best to be used in subsequent training and testing of our language models, we conduct preliminary experiments (i.e. before fine-tuning) on each of the three versions of datasets we have prepared (see the section **Relabelling and Re-proportioning of Data**). Models used for preliminary experiment are the TF-IDF variant of the Naive Bayes model and Convolutional Neural Network model (each model will be explained in greater detail in subsequent sections).

The AUC-ROC scores for two models on each version of dataset are shown in Fig. 2 and 3:

Label	Adult	Potentially Unsafe	Safe	Targeted at Kids
1:15:15:1	0.6275	0.7355	0.7433	0.5137
1:5:5:1	0.7453	0.7196	0.7469	0.6083
1:1:1:1	**0.8692**	**0.7175**	**0.7539**	**0.8076**

Fig. 2. AUC-ROC scores of NB for each dataset version

Label	Adult	Potentially Unsafe	Safe	Targeted at Kids
1:15:15:1	0.8917	0.7975	0.8121	0.8120
1:5:5:1	0.9004	0.7681	0.8011	0.8225
1:1:1:1	**0.9186**	**0.7911**	**0.8443**	**0.8627**

Fig. 3. AUC-ROC scores of CNN for each dataset version

We find that in both cases, the 1:1:1:1 dataset (i.e., the dataset with equal label counts) results in the best overall performance. Hence, for the subsequent evaluation of each language model, we use this balanced dataset for training and testing.

Preliminary Experiment on Whether to Include Automatic Spelling Correction

We understand that tokens in the URL need not be meaningful, and may often be misspelled or shortened to achieve certain figurative effects. Therefore, we reckon that automatic spelling correction of the parsed tokens might be helpful for increasing the accuracy of our classifier. Hence, we conduct experiments to compare two versions of implementation, both using the CNN model (before fine-tuning of hyperparameters) on the balanced dataset, but with an extra auto spell correction feature added to one of the versions. The auto correction is implemented on each parsed URL token, using `Speller` from the `autocorrect` Python library.

The AUC-ROC scores on both versions are shown in Fig. 4:

Label	Adult	Potentially Unsafe	Safe	Targeted at Kids
With Auto Correction	0.8685	0.7301	0.7955	0.8148
Without Auto Correction	0.9186	0.7911	0.8443	0.8627

Fig. 4. AUC-ROC scores of CNN with/without auto spell correction

We find that contrary to our initial guess, the version without auto spell correction actually gives better performance. We explain this phenomenon as the way how websites misspell words in the URL can reflect their degree of casualness, thus to some extent their safety to kids. Hence, we do not include auto spell correction in the subsequent evaluation of each language model.

Naive Bayes (NB) Models

The NB language model is implemented using `sklearn.naive_bayes.MultinomialNB`. Our first variant of NB makes use of TF-IDF of tokens in URL inputs to calculate the log-likelihood probabilities. The conversion of each tokenized URL into the TF-IDF vector was done using the `CountVectorizer` and `TfIdfTransformer` modules imported from `sklearn.feature_extraction.text`.

The optimum hyperparameters found are:
`alpha=0.3, fit_prior=True, ngram_range=(1,2)`

Fig. 5 below reflects the performance of the NB model on each of the labels.

Label	Adult	Potentially Unsafe	Safe	Targeted at Kids
AUC-ROC	0.8754	0.7286	0.7703	0.8129

Fig. 5. AUC-ROC scores of NB model using TF-IDF

We have also implemented a second variant of NB, where each URL is represented with an aggregate vector of each token's word embeddings. Our chosen aggregation method is to obtain a vector of the same length as each embedding, where each element represents the coordinate-wise maximum of the embeddings of all the tokens. Other aggregation methods include coordinate-wise minimums, coordinate-wise arithmetic means, or the concatenation of above-mentioned minimum and maximum vectors are also explored, but they yield lower AUC-ROC scores.

We train this variant of the NB model with the following optimum hyperparameters found:
`alpha=0.3, fit_prior=False`

Fig. 6 below reflects the performance of the second model on each of the labels.

Label	Adult	Potentially Unsafe	Safe	Targeted at Kids
AUC-ROC	0.6114	0.5208	0.5550	0.6076

Fig. 6. AUC-ROC scores of NB model using an aggregate of embeddings

As seen from the above tables, the TF-IDF variant of NB classifier performs much better than the word embedding variant. However, the scores for both models are not high enough.

Logistic Regression (LR) Models

The LR model is implemented using the library `sklearn.linear_model.LogisticRegression`. Similar to the previous baseline model, our first variant makes use of TF-IDF vectors as the feature.

The model is trained with the optimum hyperparameters found:
`C=100, penalty= 'l1', solver='saga', ngram_range=(1,2)`

Fig. 7 below reflects the performance of this model on each of the labels.

Label	Adult	Potentially Unsafe	Safe	Targeted at Kids
AUC-ROC	0.7671	0.6293	0.6610	0.7162

Fig. 7. AUC-ROC scores of LR model using TF-IDF

The second variant makes use of an aggregation of each URL token's word embeddings. More specifically, each URL token is converted into a *GloVe* word embedding vector, and an aggregate coordinate-wise mean of all the URL's token embeddings is used to form a feature vector. The optimal hyperparameters for such a model are:
`C=100, penalty='none', solver='sag'`

Fig. 8 below reflects the performance of the LR model on each of the labels.

Label	Adult	Potentially Unsafe	Safe	Targeted at Kids
AUC-ROC	0.6426	0.5593	0.5920	0.6292

Fig. 8. AUC-ROC scores of LR model using an aggregate of embeddings

Similar to NB models, the TF-IDF variant of LR classifier performs much better than the word embedding variant. However, the scores for the TF-IDF variant is much lower than that of the NB model.

Convolutional Neural Network (CNN)

The CNN model is implemented using `tensorflow.keras.Conv1D`. The entire model is built sequentially with a layered structure as detailed below, with fine-tuned hyperparameters alongside each layer.

Embeddings. The first step is to use pre-trained embeddings to convert URL tokens to their corresponding word vectors. We use an optimum padding maximum length of 19, with embedding dimension 50 using *GloVe* pre-trained embeddings. As such, each URL is represented by a 19×50 matrix.
1D Convolution. The embedding matrices formed by URLs are then passed through a 1-dimensional convolutional layer, with optimum number of filters as 512 and a kernel size of 3, using 'relu' activation.
Max Pooling with Dropout. The output of convolution is regularized using a max pooling layer of pool size 3, together with a 0.2 dropout rate to mitigate overfitting.
Flattening. The 2-dimensional output from the previous stage is then flattened into a 1-dimensional array.
Dense Layers. The flattened array is passed through three sequential fully-connected layers with 128, 64 and 32 units respectively using 'relu' activation. This compiles the data extracted by previous layers to form the final output.
Softmax Output. A final softmax layer to output corresponding classification probabilities for the 4 labels.

The model is then trained using the training dataset with the following hyperparameters:
```
epoch = 20, batch_size = 32, optimizer = ‘adam’,
early stopping patience = 5, monitor = ‘val_auc’,
loss = 'categorical crossentropy',
metrics = 'roc_auc'
```

Fig. 9 below reflects the performance of the CNN model on each of the labels.

Label	Adult	Potentially Unsafe	Safe	Targeted at Kids
AUC-ROC	0.9385	0.8222	0.8705	0.8873

Fig. 9. AUC-ROC scores of CNN model

We can see that the scores of all four classes improve significantly from the two baseline models. The score for the *Adult* class is particularly high, which aligns with our objective of discerning unsafe websites.

Recurrent Neural Network (RNN)

The RNN model is implemented with `tf.keras.layers.LSTM`. LSTM cells learn what information is relevant to retain or forget. We also try BiLSTM which encodes information from previous and future sequences.

The structure of the RNN model is similar to the CNN model. The optimum hyperparameters of each layer are as follows:

Embeddings. Same implementation as CNN Embedding layer with optimum padding maximum length of 20.
LSTM/BiLSTM. The embedding matrices are passed through this recurrent layer which contains 256 units, with dropout as 0.2, recurrent_dropout as 0.2.
Dense Layers. The output of the recurrent layer is passed through three sequentially fully connected layers with 128, 64, 32 units respectively. The activation function is 'relu'.
Softmax Output. A final softmax layer to output corresponding classification probabilities for the 4 labels.

The model is then trained using the same model compiler hyperparameters as mentioned in the CNN section. Fig. 10 below reflects the performance of the two RNN models on each of the labels.

Label	Adult	Potentially Unsafe	Safe	Targeted at Kids
LSTM	0.9450	0.8356	0.8784	0.8954
BiLSTM	0.9444	0.8380	0.8802	0.8969

Fig. 10. AUC-ROC scores of RNN models

We can see that the performance of the RNN models are better than the CNN models in all four classes. It is interesting to note that the simple LSTM model gives the highest score for the Adult class, outperforming all other models. However, the BiLSTM model gives the best overall performance across the four classes, with little drop in the Adult class and increase in the remaining three classes.

Combining CNN and RNN

This model adds a LSTM layer after the Max Pooling layer in CNN. The optimum hyperparameters found by grid search are the same as those for individual CNN and RNN respectively. The model is then trained using the same model compiler parameters as mentioned in the CNN section. Fig. 10 below reflects the performance of the combined CNN and RNN models on each of the labels.

Label	Adult	Potentially Unsafe	Safe	Targeted at Kids
CNN + LSTM	0.9442	0.8316	0.8775	0.8939
CNN + BiLSTM	0.9441	0.8341	0.8789	0.8964

Fig. 10. AUC-ROC scores of CNN+RNN models

We can see that the addition of the LSTM/BiLSTM layer improves the score across all four classes compared to the simple CNN model. However, the scores are still not as high as any of the RNN models. In other words, the convolution layer is holding back the performance of the recurrent layer.

Discussion

Insights on Baseline NB and LR Models

As a discriminative language model, LR generally performs better than the generative model NB in most use cases. This phenomenon is observed for our aggregate word embedding variants. However, for our TF-IDF variants, our empirical results show that the NB model outperforms the LR model, which goes against the general phenomenon.

For both baseline models, we find that input representation using TF-IDF causes significantly better performance than their aggregated word embedding variants. This is to be expected, because of the information loss from the aggregation of embedding vectors across each tokens. On the other hand, there is minimal information loss in the TF-IDF representation because information of all unigrams and bigrams are captured.

Insights on Neural Network Models

Firstly, the performance of all Neural Network models are significantly higher than the two baseline models. RNN with LSTM gives the highest score on the *Adult* class while RNN with BiLSTM gives better overall performance across all four classes. One possible reason accounting for the improvement on overall performance for BiLSTM could be that URLs are not formal and structured sequences of language, and hence word tokens could be arranged in varying orders while expressing the same meaning. Hence, information from both directions need to be encoded. The drop in *Adult* class may signal that the BiLSTM focuses more on improving the generalization across all classes and therefore the score of a particular class might be sacrificed.

We think that RNN with BiLSTM is the highest-performing model since the improvements on the other three classes are significant while the drop in Adult class is very little compared to LSTM model, and the score for *Adult* class is already high.

In terms of training time efficiency, CNN takes around 20s to train per epoch, which is 5 times faster than RNN (~100s per epoch), and 10 times faster than bidirectional RNN (~200s per epoch).

Besides, we find that after parameter tuning, the optimum kernel size for CNN is 3, indicating that information in the input URLs is best captured by *trigrams*. Furthermore, there could be some information loss incurred in the convolutional and pooling operations which do not preserve local sequence. This is potentially why CNN is outperformed by RNN.

In addition, though not shown in the tables, the discrepancy between validation and test scores is the smallest (~0.03) for the *Adult* class, while the difference is greater (~0.05 to 0.1) for the remaining 3 classes. In particular, the discrepancy differs the greatest for *Potentially Unsafe* and *Safe* classes. One reason could be the downsampling process, where we only retain all entries in the *Adult* category from the original dataset. The randomized downsampling of the other 3 categories could lead to information loss, especially for *Potentially Unsafe* and *Safe* classes where the sizes are only 1/20 of their original sizes. Another reason could come from the relabelling stage. Since *Potentially Unsafe* and *Safe* classes contain multiple categories of the original dataset, the instances are more diverse, and thus are harder to generalize.

Lastly, past research by Rajalakshmi et al. [6] claims that CNN with bidirectional RNN gives the best performance on a similar website classification task. However, we are unable to reproduce this result in our experiment. There could be several reasons. First, we just use a single convolutional layer with a fixed kernel size, where more complicated CNN structures may give better performance. Second, a different evaluation metric is used in that study, resulting in different outcomes. Lastly, the objectives of the classifiers are different. In our study, we focus more on the correct detection of the unsafe websites.

Conclusion

Summary of Findings

We have empirically found RNN with BiLSTM to be the most accurate model among those we have tried for discerning the safety of web pages based on their URLs. However, the training of RNN is noticeably longer, indicating that RNN is more computationally expensive due to the numerous weights to be tuned during the forward computation and loss backpropagation.

Limitations of Study

One limitation of our research is that we could not find proper datasets tailored specifically for our task, hence have to manually transform and regroup the website categories from existing datasets. However, the relabelling method we adopt may be too general and subjective. In particular, *Potentially Unsafe* and *Safe* categories are constructed by looking at the original categories only, without the examination of the website content. This process could incur noise. This may also be the reason for lower scores in *Potentially Unsafe* and *Safe* classes across all models.

Secondly, we perform downsampling to balance the four categories in the original dataset *before* splitting into train and test data. This means that our test data is actually seen and touched during the preprocessing stage. Therefore, the test data (with equal proportions of the four categories) may not reflect the distribution of incoming websites in real-world situations (where *Unsafe* websites are likely to be a minority).

In addition, the URL classifier alone may not be a strong enough defender if it were to be deployed in parental control software. For sites categorized as *Safe* by our model, we recommend further inspecting the HTML content to be more reassured about their validity. This would serve as an additional layer of protection. Nevertheless, the URL classifier does serve as a quick and simple preliminary detector for potentially unsafe websites.

Areas for Future Research

In the current implementation, the CNN model only has one convolution layer with a kernel size of 3. Concatenation of parallel convolution layers with different kernel sizes (eg. 2, 3, 4) can be further explored to capture the interpolation of bigrams, trigrams and 4-grams. This may improve the complexity of our features and thus make our model more robust.

Due to URLs' short length and frequent usage of abbreviations, we can also consider sub-word level features, e.g. character embeddings, which might be a more suitable feature than word tokens in some cases.

In the current dataset preprocessing, we downsample the larger classes to construct a balanced dataset. However, the downsampling may lead to information loss especially when the original class size is large (e.g. (eg. *Potentially Unsafe* and *Safe* classes are about 20 times larger than the smallest *Adult* class). Therefore, upsampling smaller classes is another option for data balancing to preserve the critical information in the larger classes.

References

[1] Baykan, E., Henzinger, M., & Weber, I. (2011). A Comprehensive Study of Techniques for URL-Based Web Page Language Classification. ACM Transactions on the Web.

[2] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

[3] Kan, M.-Y. (2004). Web page categorization without the web page.

[4] Kan, M.-Y., & Thi, H. O. (2005). Fast webpage classification using URL features. Singapore.

[5] Le, H., Pham, Q., Sahoo, D., & Hoi, S. C. (2018). URLNet: Learning a URL Representation with Deep Learning for Malicious URL Detection.

[6] Rajalakshmi, R., Tiwari, H., Patel, J., Kumar, A., & Karthik.R. (2019). Design of Kids-specific URL Classifier using Recurrent Convolutional Neural Network. International Conference on Computational Intelligence and Data Science (ICCIDS 2019). Chennai: Elsevier.

[7] Tatman, R. (2017). English Word Frequency. *Kaggle*. Retrieved March 31, 2021, from https://www.kaggle.com/rtatman/english-word-frequency/metadata

[8] Sayad, A. (2020). Website classification using URL. *Kaggle*. Retrieved March 31, 2021, from https://www.kaggle.com/shaurov/website-classification-using-url/metadata