AAAI: Adversarial Attack in Authorship Identification Against Deep Learning Based Model

*Note: Sub-titles are not captured in Xplore and should not be used

Tung-Yu Hsieh
Department of Computer Science,
National Tsing Hua University
Hsinchu, Taiwan

Tain-Yuan Liu
Department of Computer Science,
National Tsing Hua University
Hsinchu, Taiwan

Wei-Po Lin
Department of Computer Science,
National Tsing Hua University
Hsinchu, Taiwan

Zhao-Wei Hong
Department of Computer Science,
National Tsing Hua University
Hsinchu, Taiwan

Hong-Lin You
Department of Computer Science,
National Tsing Hua University
Hsinchu, Taiwan

***Abstract*—Deep learning has been widely use on many NLP (Natural Language Processing) tasks, such as text generation, text classification, question-answering, and so on. However, we've observed that some of the existing text classification model could be affected by a small amount of noises in the input data, and result in severe prediction error. In this paper, we test the vulnerability of current existing text classification model on authorship identification by using adversarial attack, which has been proposed in computer vision territory for a long time. By using white-box attack, we select partial phrases in the article and modify them using various kind of algorithms, while maintaining similarity above a threshold or as high as possible. Our best result shows that we are able to decrease the classifier's accuracy from 92.3% to 16.4% while remain more than 0.77 of semantic similarity.**

***Index Terms*—Adversarial Attack, Machine Learning, NLP, Author Identification**

I. Introduction

NLP (Natural Language Processing) is a thriving territory due to the appearance of ChatGPT. With the support of hardware acceleration, training process of a language model is no longer time-consuming. The invention of different machine learning architectures aids the development of new models. Authorship identification is one of the text classification problem which can be dealt with by state-of-the-art neural network architecture like LSTM (Long Short Term Memory) or transformers. Although most of existing models have been well tested, they will inevitably produce incorrect predictions under complicated real-world circumstances. It might due to the diversity of user scenario, or the limit of hardware specifications. However, there is possibility that prediction might be interfered by anonymous attacker manually.

CS460200 Introduction to Machine Learning Final Project, National Tsing Hua University, 2024.

Adversarial Attack is the type of attack which aims to confuse the classifier by modifying a small part of input data, makes it barely unable to notice the modification by human but successfully tricked the classifier. In image classification in the most intuitive example of adversarial attack. By adding some proper noise to the origin input image, the classifier will produce wrong output, although two input pictures looks almost identical. This will become a severe problem when adversarial attack has been used in realistic scenarios, such as obstacle detection on self-driving cars and facial identification.

In NLP territory, adversarial attack have not been widely tested yet. Thus, we decide to gather information about it and set up our experiment to testify the impact on text classification model which has the framework with high usage. There are multiple tasks for text classification, such as plagiarism detection, sentiment analysis, and grammar check. Here, we will focus on one of the text classification task: Authorship Identification.

Authorship Identification is the job to identify the author belongs to the input article. It can be used on copyright protection, criminal document analysis, and linguistics research. The whole process will start from preprocessing the input text with a tokenizer to separate words from a whole article. Then, word embeddings are built and transform each word to the embedding vector for further influence or training. The model will take the input vectors into the calculation with multiple layer of forwarding propagation on commonly used neural network framework, and result in a list of weights or possibilities belongs to each author. The model framework can be applied with different architectures, but neural networks are most widely seen, especially when LLMs (Large Language Model) are thriving for recent years.

We aims to verify the impact on the model when attacker has a thorough view, that is, we've set our attack scenario

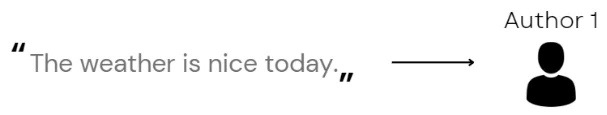

Fig. 1. Schematic of authorship identification

into **white-box attack**. White-box attack assumes the attacker are able to know the details of the model, which simulates a severe security breach. Under this circumstances, the model's hyper-parameters, architecture, training dataset, and the known author list is transparent to the attacker. With this setting, we are able to find out the vulnerability of existing architecture on text classification tasks, which has barely tested before.

We've aimed to attack the classifier in two phases. They represent different extent of attack as described below:

- **Phase 1: Confusion.** In phase one, our goal is to confuse the classifier so the modified articles should be identified as any other authors' creation excepts the original one. The attack in this phase is more simple and straight forward since any other author is acceptable, even if the predicted author is different each time. As long as the predicted author is not the original author, it will be treated as a successful attack.

- **Phase 2: Assignment.** In phase two, our goal is to make every article being identified as articles from a specific author. Meanwhile, the similarity between the modified articles and original articles will be measured as a metric to evaluate the effect of our attack. The specific author is chosen manually.

II. Evaluation

The flow of the attack will work as follow. First, we have to consider what words should we replace. In order to have an effective attack, the words we've picked should have great importance to represent the writing style or characteristic of an author. Then, we have to find suitable alternatives, which are the words to substitute picked candidates. These words have to be either inconspicuous to the model or distinctive for another author so the model is more possible to identify incorrectly.

A. Target Model

We select the article level LSTM model used in [2] as our opponent. As one of the common RNN (Recurrent Neural Network) framework, LSTM is a typical architecture for text classification. We re-train the model with some modification to libraries version in order to fix some library compatibility problem. For the rest of the part, all hyper-parameters and packages, remain the same as in [2]. we store the trained model weight to external directory, so we can keep the consistency of outputs.

B. Datasets

The dataset we've used here is Reuters 50_50 news dataset. This dataset collects articles from 50 Reuters journalist, each with 50 news reports. It is also the same dataset as the one used in [2] so it will be much easier to calculate the mis-classified rate comparing to the original articles.

C. Word Selection

In total, we have tried three different method to select candidates to be replaced for phase 1 and phase 2.

- **TF-IDF:** TF-IDF is widely used on text analysis, It uses two different metrics to evaluate the importance of a word in an article. First, TF stands for term frequency. It calculates the frequency of a word appears in an individual article.

  $$TF(w) = \frac{\text{Number of word } w\text{'s appearance in a document}}{\text{Total number of terms in the documents}}$$

  Frequent appearances will be considered as more important. However, we can't only use TF since stop-words will have incredibly high TF score. Thus, we combine it with IDF score. IDF stands for inverse document frequency. It's calculated by taking the logarithm of the number of documents in the corpus divided by the number of documents where the term appears.

  $$IDF(w) = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents with word } w \text{ in it}}\right)$$

  Then, the TF-IDF score is the product of TF and IDF:

  $$TF\text{-}IDF(w) = TF(w) \times IDF(w)$$

  The intuition behind TF-IDF is that terms that appear frequently in one document but not in many documents across the corpus are likely more relevant and characteristic of that document's content, hence giving them a higher score. This makes TF-IDF an effective way to filter out common and non-discriminative words in a corpus while preserving the important terms.

- **POS tagging:** POS stands for Part Of Speech. It's a category in grammar for those words having similar grammatical properties. For example, common English part of speech have noun, verb, adjective, adverb, pronoun, preposition, conjunction, etc. We leverage this property to find words having special part of speech and take them as candidates. By tagging all the words in an article and categorize them, select some certain categories like NN (Noun, Singular), NNP (Proper Noun, Singular), and DT (determiner), etc. All of the words in the selected categories will be considered as candidates.

- **Gradient norm:** In the setting of white-box attack, we are able to use the internal parameters as extra information. Thus, we retrieved the gradient on the word embedding of baseline model after training process is complete. These parameters is calculated from the back-propagation process while model is training. Affected by the shape of the embedding inside the model, each word has been transformed into a vector with a-hundred weight. Our goal is to find the words having the maximum gradient in it's word embedding. With larger gradient represents the higher possibility that the word has been treated as a crucial word to the model. We calculate the norm of each word vector by Frobenius norm.

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_ij|^2}$$

After the norm of gradient is calculated, we sort them in decreasing order to have the top k words which have largest norm. The number k can vary depends on our trial or algorithm.

D. Word Replacement

- **Misspelling:** In phase one, we only have to confuse the classifier. A quick and intuitive way is to spell the words incorrectly on purpose. We only have to make sure the words after misspelled do not exist in the word embedding, which makes sure those crucial words have been replaced into unknown tokens in classifiers sight.

- **Word Embedding Distance:** This method uses a straight forward concept, which leverage the same word embedding as the one in baseline, and calculate the euclidean distance between the candidates and all other words in word embedding. The word having minimum distance will be the best alternative. Since all the words are in the word embedding, it is guaranteed to find a word with minimum distance from the original one.

- **Masked Language Modeling:** MLM (Masked Languaed Modeling) is a task that add some mask token to a specific sentence, and the model has to predict the word which should be put into the mask. It's a NLP version of cloze test, and this scenario is fits the task we want. After we selected all the candidates in an article, we use a simple iterative function to replace all candidates into the mask token, and ask another language model to fill them back one by one. In order to confuse the classifier, this model needs to perform like another author. Luckily, with the aid of transformer, mask language modeling can be solved easily.
In order to train this fake author, we use `bert-base-uncased` transformer as our base model. This is a reproduced version of BERT from huggingface hub which is used for uncased MLM in English. The model is pre-trained with some external datasets, so we don't need to restart the training process entirely. We've picked articles from a specific author as our additional dataset, and fine-tune the model with it. This can help us test out it's performance in phase 1 and phase 2 attack.

- **TF-IDF:** In our work, we enhanced TF-IDF than general usage mentioned above. Refer to the thoght of the TF-IDF, using logarithm to deal with statistical data, we develop some improved IDF methods in the following:

$$\begin{aligned} AR &= \text{Total number of aritcles} \\ AU &= \text{Total number of authors} \\ AU(w,i) &= \text{Number of author[i]'s articles contain word w} \\ \varepsilon &= \text{hyper-parameter to filter the too high frequent words} \\ \alpha &= \text{hyper-parameter to penalize words appear in other authors' articles} \\ \omega &= \text{hyper-parameter to filter the word appearing times less than } \omega \end{aligned}$$

Then, we calculate TF-IDF metric in three different process: filtering, weighting, and polishing. In filtering process, we use a IDF metric for deleting frequency word: IDF_d. When IDF_d equals 0, it indicates word w is a very normal word like "the, is, am ,are...". Set larger ε can elevate the criteria for the degree of normal.

$$IDF_d(w) = \begin{cases} 0, & \log\left(\frac{AR}{AR(w)}\right) \le \varepsilon \\ 1, & \log\left(\frac{AR}{AR(w)}\right) > \varepsilon \end{cases}$$

In weighting process, we leverage a new metric for author[i]'s characteristic: IDF_c. Large

$$IDF_c$$

indicates word w is with author[i]'s feature. Set larger

$$\alpha$$

can penalize the situation where word w appears in other author's article more strictly.

$$IDF_c(w,i) = max(0, \log \left(\frac{AU}{min(1, AU(w,i)) + \alpha\sum_{j=1,j\neq i}^{AU} min(1, AU(w,j))}\right)) \quad \alpha > 1$$

Finally, in polish process, we use a similar metric to TF but with some modification. IFF_f represents author[i]'s frequency. Large IDF_f indicates word w is appear frequently in author[i]'s articles.

$$IDF_f(w,i) = \log(max(1, AU(w,i) - \omega))$$

Complete TF-IDF metric for word replacement will be the value of:

$$IDF(w,i) = IDF_d(w) \times IDF_d(w,i) \times IDF_f(w,i)$$

- **Genetic Algorithm:** Genetic algorithm is a method able to be used to solve constrained or unconstrained optimization problems that is based on natural selection. At each step, this algorithm will pick up some individuals to be parents of the population and create some "children" to be next generation. When reading a lot of paper of NLP adversarial attack, we noticed that a piece of paper used a similar algorithm, hybrid local search algorithm. Hence, we are inspired to give genetic algorithm a try. We refer to the paper and design a suitable genetic algorithm for this project.
  By TFI-DF, we can get the important words for every authors. We find that 20th author has the most number of important words by TF-IDF. Hence, we will attempt to modify all texts to be identified as the 20th author's text. For any modified text which has been able to be identified from 20th author, we hope to increase the similarity between this modified text and its original through genetic algorithm.
  1) Through TFI-DF, build up a table which contains the important words of 20th author. Classify these words by their POS(part of speech). For every POS of classification, according to the degree of importance of each word, leave the top 70% and eliminate the remaining 30% of words.
  2) Considering the long calculation time of the total process of genetic algorithm, we randomly pick up 500 texts from our dataset. For every word in every text, if the POS of this word (denote as `W1`) is in the table of words of 20th author, randomly choose a word (denote as `W2`) with the same POS in the table. The chance for each word in the table to be chosen randomly is according to the degree of importance for 20th author. There is 90% of chance that `W1` will be replaced with `W2`.
  3) For every original text, through the method of step2, get at most 5 modified texts as the initial generation. However, it is possible that the modified text cannot be identified from 20th author. For every original text, we will try the method of step2 for at most 100 times. If we still can find 5 modified texts which can identified from 20th author, let it be. Hence, some original texts have less than 5 modified texts (even 0 modified texts).
     The following steps are the evolution process of genetic algorithm.
  4) Selection of parents: Every individual is parent.
  5) Crossover: Every two pair of parents will generate a child. The way of crossover is "uniform crossover", which is that every word of the child has 50% of chance from father and 50% of chance from mother.
  6) Mutation: We hope that the algorithm can find a child which has better similarity after mutation. Hence, for every word of the child, there is 5% of chance to be replaced with the corresponding word of the original text.
  7) Fitness and selection of survivors of next generation: Sort the total text with the similarity between the current modified text and the original text. The similarity is the fitness function. And we use universal-sentence-encoder to calculate the cosine similarity between the original text and the modified text. In this part, we will pick up top $min(5, int(0.5 * len(generation)) + 1)$ texts to be next generation according to their similarity.
  8) Repeat the step4 to step7 for 20 iterations. For every original text, we will pick up a text which has the highest similarity to be the final modified text. If only 1 text survive in the generation, the process of evolution of 20 iterations will suspend and the remaining 1 text will be the final result.

E. Similarity

Cosine Similarity: Cosine similarity has been widely used on semantic analysis, which measures the distance between two vectors of an inner product space. This allows that the similarity value will only hold within -1 to 1, comparing to other method like euclidean distance or Manhattan distance, their output value will overflow easily in high dimension embedding and susceptible to the length of the text.

In our work, we use a third party pre-trained sentence BERT model as a fair measuring tool. Sentence BERT is a modification of BERT which has been used for measuring sentence similarity for a while.

III. Implementation

We use `tensorflow` and `pytorch` to implement attack models mentioned above. The baseline model is trained on Google Colab with a Nvidia Tesla T4. For the rest of the approaches we've tried, we use Kaggle, Colab, and Taiwania 2 interchangeably. On Kaggle and Colab, we train and testify our model with the support of a single Nvidia Tesla T4. As for Taiwania 2, we use a single Nvidia V100 to speed up the training process.

POS tagging is completed by the support of `nltk` package, and word embedding is built with the support of `torchtext` and `GloVe` libraries. All input articles is tokenized by `spacy` module with `en_core_web_sm` model.

IV. Results

Baseline: For comparison, we list the result of the baseline model with original articles as input. The overall result will be Accuracy ≈ 0.93, Loss ≈ 0.0064, and F1 Score ≈ 0.93.

Phase 1: In phase 1, we've used two different combinations to attack, which are (gradient norm + word embedding) and (POS tagging + misspelling).

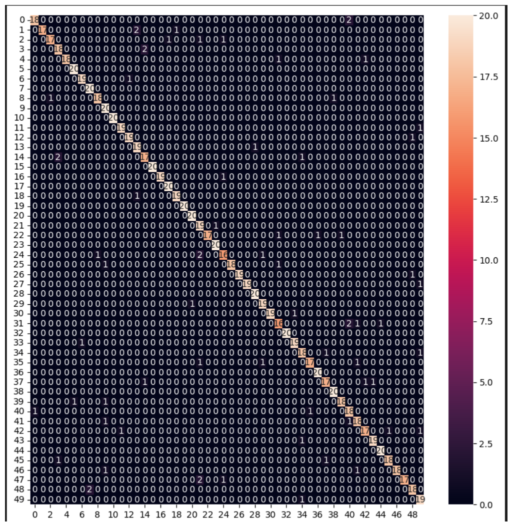
Fig. 2. Result of baseline model

The result of (gradient norm + word embedding) is not quite well, The diagonal line still exit in it's classification result, which means misprediction barely happens. In this case, in order to speed up the modification process, we only choose 1000 article as attacking sample.

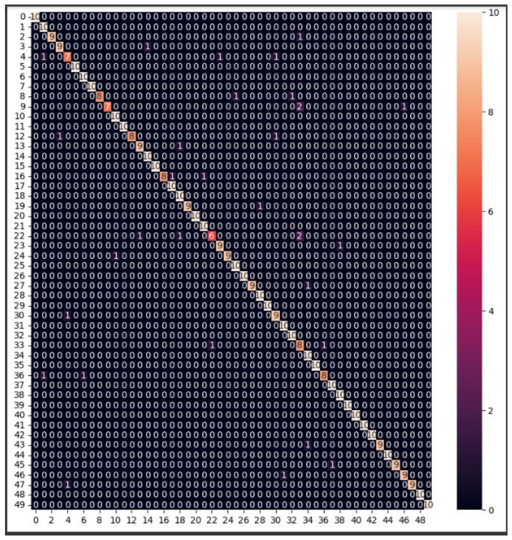
Fig. 3. Result with gradient norm + word embedding distance

As for (POS tagging + misspelling) combination, our best result shows that by selecting all words with {NNP, NNPS, NN, NNS} part of speech tag, we are able to confuse the model well by having severe accuracy degradation. The accuracy has only 0.164, with Loss ≈ 0.0976 and F1 Score ≈ 0.14, while the average cosine similarity remains for ≈ 0.77.

Phase 2: In phase 2, we've used four different combinations, which are (TF-IDF + TF-IDF), (gradient norm + MLM), (gradient norm + TF-IDF), and (POS tagging + Genetic Algorithm).

First, by using TF-IDF in both selecting and replacing, our

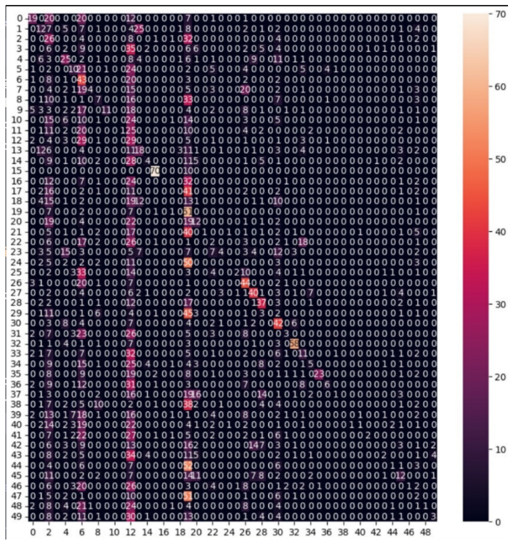
Fig. 4. Result with POS tagging + misspelling

classifier stands well. We select the top 60 word chosen by TF-IDF algorithm and replace them with enhanced TF-IDF. It only decreases the accuracy to ≈ 0.832, Loss ≈ 0.0124, and F1 Score ≈ 0.83, while the cosine similarity has ≈ 0.9 This method turns out to be failed due to ineffectiveness of confusing.

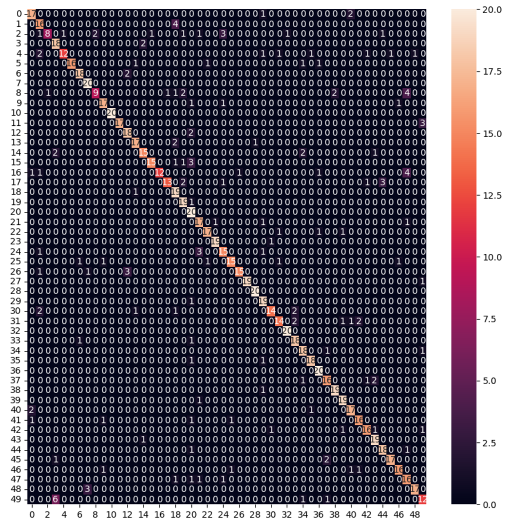
Fig. 5. Result with TF-IDF + TF-IDF

In the combination of (gradient norm + MLM), we use a fine-tuned BERT for masked language modeling, and trained with 764 sentences appeared in author 0's article, and trained with 10 epochs. We replace top 30 words having biggest norm to BERT token "[MASK]", and generate them using the model mentioned above. The result is disappointing, the best attack has accuracy ≈ 0.76, Loss ≈ 0.018, and F1 Score ≈ 0.75, while the cosine similarity only has ≈ 0.42. We can clearly see it's unable to confuse the classifier well, not to mention identifying them as articles form a single author.

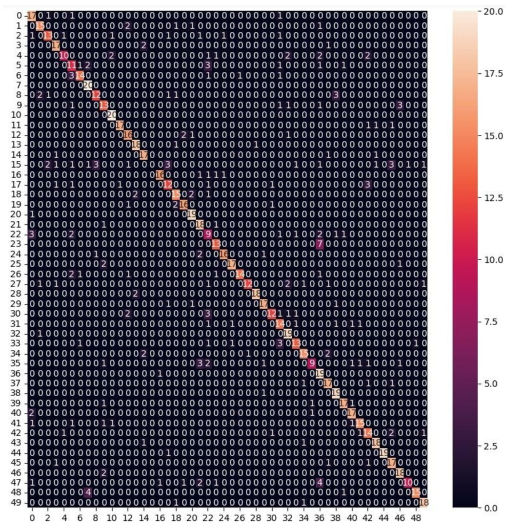
Fig. 6. Result with gradient norm + MLM

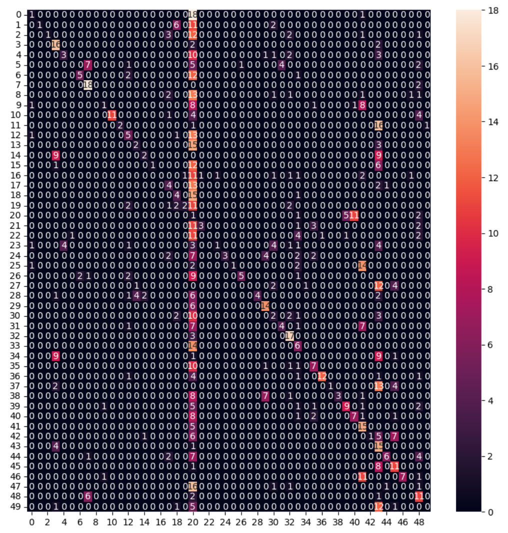
Fig. 7. Result with gradient norm + TF-IDF

In (gradient norm + TF-IDF) scenario, by selecting top 120 words having biggest norm, and replace them by enhanced TF-IDF algorithm, we can tricked classifier more by decreasing the accuracy to Acc ≈ 0.261, Loss ≈ 0.0878, and F1 Score ≈ 0.27, while the cosine similarity remains ≈ 0.78. By observing the heat map, we can clearly see most of the articles have gathered toward a specific author 20, who is the author we've picked.

In (POS tagging + Genetic Algorithm) scenario, we initially pick up 500 modified text which can be identified from 20th author. Hence, the heat map of Fig. 8. is almost a straight line aligned with 20th author. The impotance of genetic algorithm is the improvement of the similarity between the original text and the modified text. The average similarity of the initial generation is 0.450792. Ahter genetic algorithm, we improve the average similarity to be 0.484387.

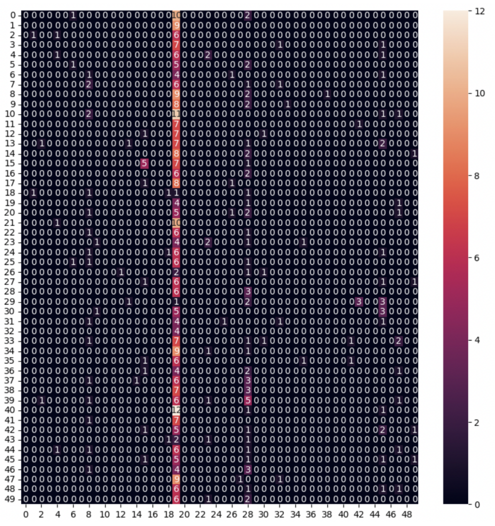
Fig. 8. Result with POS tagging + Genetic Algorithm

V. Conclusion

Adversarial attack on NLP is a relatively novel territory. Although our work shows the potential vulnerability of adversarial attack on language model, there are still a lot of room for improvement. We use a mathematical metric to calculate the similarity between articles. However, to achieve successful adversarial attack, the modified article should be able to trick humanity's eyes. Practically, we only retrieve 0.77 of semantic similarity, which still have a lot of space for further improvements.

Besides, our work in this paper focused on white-box attack. In practice, detailed information about existing model is almost impossible for attacker to obtain. Under this kind of circumstances, **black-box** attack which assume the detailed information about the opponent is concealed to attacked. This setting is more practice to real world scenario, and able to apply on far more existing models.

Data and Code Availability

All files related to this paper is in our public github repository can be found at https://github.com/Catking14/AAAI_ML23.

Author Contribution Statement

T. -Y. Hsieh (20%): research and implement baseline model, model design on some phase 2 method such as MLM BERT model.
T. -Y. Liu (20%): research on attacking methods and related papers, implement POS tagging model in phase 1 attack, research and implement similarity metric calculation.
W. -P. Lin (20%): research on attacking methods and related papers, implement genetic algorithm in phase 2, research and implement similarity metric calculation.
H. -L. Y (20%): produce presentation slides and final presentation, research on attack methods and related papers.
Z. -W. H (20%): research on attack methods and related

papers, implement TF-IDF method in phase 2.

References

[1] C. Qian, T. He, and R. Chang, “Deep learning based authorship identification,” Stanford archive, 2017. Available from https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1174/reports/2760185.pdf.

[2] A. Talati, A. Sharma, and R. Narayanan, “Deep learning based authorship Identification,” 2020. Available from https://github.com/arthtalati/Deep-Learning-based-Authorship-Identification/tree/master.

[3] J. Pennington, R. Socher, and C. Manning, “Glove: Global vectors for word representation,” in EMNLP, 2014, pp. 1532–1543.

[4] J. Li, S. Ji, T. Du, B. Li, and T. Wang, “Textbugger: Generating adversarial text against real-world applications,” *arXiv preprint arXiv:1812.05271*, 2018.

[5] P. -Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C. -J. Hsieh, “EAD: Elastic-Net Attacks to Deep Neural Networks via Adversarial Examples” *arXiv preprint arXiv:1709.04114*, 2017.

[6] A. Kurakin, I. J. Goodfellow, and S. Bengio, “Adversarial machine learning at scale,” in ICLR, 2017.

[7] Z. Yu, X. Wang, W. Che, and K. He, “TextHacker: Learning based Hybrid Local Search Algorithm for Text Hard-label Adversarial Attack,” *arXiv preprint arXiv:2201.08193*, 2022.