# One Character Is All You Need: Naive Black-box One-character Adversarial Attacks on Hate Speech Detection

**Tai Mai**
mai@cl.uni-heidelberg.de

## Abstract

**Content warning: Hateful language.**
**Due to the nature of the task tackled in this project, this report and the accompanying code and data contain hateful words and phrases that may be upsetting. To avoid confusion with text produced through methods introduced in this report, I opt to not censor these hateful terms. Reader discretion is advised.**

Adversarial attacks can be described as perturbations to data samples that change a model's predictions while being small enough to be disregarded by humans. This project explores the efficacy of a naive adversarial attack strategy that changes a single character in a text containing hate speech. After gathering adversarial examples in a dataset split, those can be used on another dataset split (e.g. test) with simple look-up substitutions to lower a hate speech detection model's recall on hate speech texts by half.

## 1 Introduction

While hateful conduct is not a recent phenomenon, it was presented with an optimal breeding ground through the introduction of the internet. The anonymity granted to users via avatars and arbitrary usernames, as well as a lack of moderation can allow bad actors to hide and engage in hateful language in a way they might not be able to in the real world. This creates a toxic and unsafe environment, especially for users targeted by that hateful language. With the sheer number of users on the internet, manual moderation is infeasible. To address this problem, efforts have been made towards automating the detection of written hate speech. The task of hate speech detection is a classification task that entails recognizing hateful language as such. It is imperative that such automatic hate speech classification systems are consistent and robust. However, experiments on visual classification systems have shown that they can be easily fooled by minimal adversarial attacks on their data (Su et al., 2019). If hate speech detection systems, too, are susceptible to these kinds of adversarial attacks, bad actors might look to abuse those loopholes and try to avoid being classified as hateful in order to circumvent a ban on online platforms. Thus, the questions arises of how clever you have to be to fool a hate speech detection system.

This project aims to showcase the efficacy of a naive and (for humans) minimally invasive attack strategy. It does not require knowledge of the internal workings of the victim hate speech detection system, other than the predicted probabilities assigned to the classifications. This method is evaluated on the hate speech detection dataset HateXplain (Mathew et al., 2021) and its accompanying hate speech detection model.

The code for the project can be found in an internal GitLab repository[1]

## 2 Related works

Waseem and Hovy (2016) proposed another simple hate speech detection dataset containing 16,914 Twitter posts, annotated as either *racism*, *sexism*, or *none*. In contrast to HateXplain (Mathew et al., 2021), the posts are listed by their post ID instead of containing the actual text in the post, requiring the use of the Twitter API. Therefore, we forewent this dataset and utilized HateXplain (Mathew et al., 2021) for ease of use. Furthermore, it is worth noting that HateXplain (Mathew et al., 2021) also contains more samples (20148) than Waseem and Hovy (2016). We conjecture that the methods presented in this project report might still be transferable to other datasets like Waseem and Hovy (2016) and the method's efficacy is more dependent on the type of model used. The current state-of-the-art model on the HateXplain (Mathew et al., 2021) dataset was proposed by Kim et al. (2022). According to

[1] https://gitlab.cl.uni-heidelberg.de/mai/adversarial-hatespeech

a leaderboard[2] for HateXplain, the performance improvement is marginal compared to the original HateXplain model proposed in conjunction with the dataset (Mathew et al., 2021). Since the HateXplain (Mathew et al., 2021) model has been officially uploaded to huggingface (Wolf et al., 2020), it was preferred over the state-of-the-art model for ease of use.

Adversarial attacks have been explored extensively for visual tasks like image classification. The work by Su et al. (2019) investigated how a single changed pixel can sway the prediction of an image classifier. This project takes an analogous approach for written language. Previous work on adversarial attacks on text by Yoo et al. (2020) and Berger et al. (2021) regard black-box adversarial attacks on a token level. While Yoo et al. (2020) take more sophisticated approaches, Berger et al. (2021) show that simple heuristics are sufficient for adversarial attacks. This project subscribes to the same philosophy but approaches it on a character level, instead of token level. The black-box strategy used in this project for choosing the specific character to attack is inspired by the token-level approaches taken by Gao et al. (2018) and Jin et al. (2020). Gao et al. (2018) determined the most important token in the sentence by observing the score difference when replacing each tokens (one at a time) with an `UNK` token. Instead of replacing it with an `UNK` token, Jin et al. (2020) deleted each token altogether to observe changes in the prediction scores. The approach taken in this project, too, observes the changes in the prediction scores but foregoes deleting characters and skips right to replacing characters.

## 3 Method

### 3.1 Approach

White-box adversarial attacks, like proposed by Goodfellow et al. (2014), utilize information from inside the neural classifier, such as the gradients, in order to modify data samples and mislead the classifier. Naturally, this allows for more informed and well-directed attacks. In contrast to that, the method presented in this project assumes the perspective of an end-user, looking to commit hate speech without being detected as such. Therefore, the hate speech detection system is treated as a black-box. The most interesting adversarial attacks are the ones that are strong enough to mislead neural models, yet weak enough to not interfere with human comprehension and judgment. Adversarial attacks as proposed by Goodfellow et al. (2014) work by adding an unnoticeable amount of noise to whole images. Su et al. (2019) take an almost opposite approach by adding a noticeable amount to a single pixel. Both approaches are weak enough for humans to dismiss. Previous work by Yoo et al. (2020) aims to achieve the same effect for text by substituting tokens with other, similar tokens such that the meanings of the entire sentences are not significantly distorted.

This project aims to take an approach inspired by one-pixel attacks (Su et al., 2019) by allowing arbitrary substitutions of a single character in the text. The intuition behind taking this approach is grounded in the fact that humans are often able to disregard typos in digital text. Modifying words by substituting characters with special characters is even done deliberately when censoring swear words, for example in "*f*ck*". Humans are typically able to fill in the blank and make out the meaning through context. While, initially, normal roman letters were included in the list of permissible character substitutions, upon inspection of the attacked texts, there have been instances where the character substitutions significantly altered the meaning of the word and subsequently the whole sentence. An example of such an occurrence is when the offensive word "*retarded*" was replaced with the rather positive word "*rewarded*", e.g. in the sentence "*why are you repeating yourself are you a little* ***retarded/rewarded***". For this reason, this project will primarily focus on substitutions with only special characters and numerical digits, akin to the aforementioned censoring.

### 3.2 LIME

To analyze and understand the model's reaction to the adversarial attacks, we utilize LIME (Ribeiro et al., 2016). LIME is a model-agnostic toolkit for obtaining a classifier's rationales behind its classification decisions. LIME approximates the model decisions by "learning an interpretable model locally around the prediction" (Ribeiro et al., 2016). Applying it to the HateXplain model yields a list of tokens and the individual score contributions that the model assigns to them.

[2] `https://paperswithcode.com/sota/hate-speech-detection-on-hatexplain`, last accessed: 2023-03-29

### 3.3 Finding adversarial examples

#### 3.3.1 Brute force

To find adversarial examples, every character in a text of length $l$ is replaced with every one of the $n$ permissible substitution characters, one at a time. The resulting $l \times n$ candidate texts are then evaluated with the hate speech detection model and ranked by their reduction in abusive probability. Of those $l \times n$ candidates, the top $k$ candidates are recorded. This procedure results in sentences such as "*why are you repeating yourself are you a little @etarded?*"

An attack is considered successful if the predicted abusive score $\in [0, 1]$ of an abusive text falls from $\geq 0.5$ to $< 0.5$. This method of finding adversarial examples is obviously very naive and costly but manageable considering the size of the dataset at hand. Further details about the dataset are discussed in Section 3.5.

#### 3.3.2 Educated guesses with LIME

This method works fundamentally the same as the brute force method but reduces the attack area from the entire text to a single token in the sentence. Using LIME on the original text yields the token with the highest contribution to the abusive score. The brute force method is then applied to only attack characters in this token instead of the entire text.

### 3.4 Applying the adversarial examples

Applying the methods above on the train or validation split of the HateXplain dataset (Mathew et al., 2021) allows the construction of a look-up substitution dictionary which can be used on the test split of the dataset. This substitution dictionary contains the most attacked victim tokens and their most successful "censored" counterparts. Furthermore, for each victim token and each substitution token, their numbers of occurrence among the top $k$ attacks is recorded in this substitution dictionary. An example excerpt of such a substitution dictionary can be seen in Table 2. These numbers of occurrence allow for making empirical guesses on which victim token to substitute during test time. We choose the one victim token in the sentence that was successfully attacked the most and replace it with its most successful substitution token.

### 3.5 HateXplain

Mathew et al. (2021) proposed the HateXplain dataset together with mechanisms and models for detecting hate speech. In contrast to the other, previously mentioned hate speech dataset by Waseem and Hovy (2016), HateXplain includes the judgments of three annotators per text. Furthermore, when annotators classify a text as abusive, their rationales are included in the dataset in the form of a binary list, indicating whether or not each token in the sentence was relevant for the abusive classification. These rationales are provided in an effort to introduce and train explainability, as well as improve performance with their classifiers. Their most performant model uses BERT (Devlin et al., 2018) together with their proposed attention mechanism which was trained with the annotator's rationales as ground truth attention.

While the dataset actually contains three different classes ("*normal*", "*offensive*", and "*hate speech*"), we assume the binary classification scenario of the dataset in which "*offensive*", and "*hate speech*" are combined under the umbrella term "*abusive*". To reconcile possible disagreements between the three annotators, their judgments are collapsed into a single label via majority vote. The HateXplain dataset (Mathew et al., 2021) contains 20148 social media posts from Twitter and Gab, a predominantly right-wing social media. The dataset is stratified into train, val, and test splits in a ratio of $8:1:1$, where the val and test splits contain 1922 and 1924 texts, respectively.

## 4 Experiments and Results

As mentioned in Section 3.5, the val and test splits of the HateXplain dataset contain 1922 and 1924 texts, respectively. Of those texts, 934 and 951, respectively, were labeled as abusive by both the model and the annotators. We only use texts that are correctly classified as abusive before any attacks to compare the model's recall on abusive texts before and after adversarial attacks. These 934 texts from the val split are used to find adversarial examples which are then used to attack the 951 texts from the test split.

### 4.1 Brute force

#### 4.1.1 Special characters and digits

We first discuss our findings when only using ASCII special characters and numerical digits as permissible substitution characters on the val split. Roughly 68% of the 934 abusive texts in the val split were vulnerable to at least one of the brute force adversarial attacks. We call the remaining

| Method | Permissible substitutions | Recall | |
|---|---|---|---|
| | | val | test |
| Brute force | special + digits | 0.32 | 0.46 |
| | special + digits + alphabet | 0.31 | 0.50 |
| LIME guess | special + digits | 0.34 | 0.46 |

Table 1: Recall of abusive texts after adversarial attacks.

32% resistant, meaning none of the attacks on those texts managed to lower the abusive probability to below 0.5. The mean text length of vulnerable texts is about 127 characters, while the mean text length of resistant texts lies at about 154 characters. This hints at a correlation between text length and vulnerability. On average, around 5% of all brute force attacks on a given text are successful.

After gathering the adversarial substitutions from the val split, the test split can be attacked. By simply replacing the empirically most attacked token in a test sentence with its most successful substitute, around 54% of the texts that were previously correctly classified as abusive were now falsely classified as not abusive with an average abusive score of 0.46.

#### 4.1.2 Special and alphanumeric characters

When adding the roman alphabet to the list of permissible characters, the numerical results look very similar. The success rate on the val split rises to about 69%. We observe the same correlation of text length and vulnerability with an average vulnerable text length of 114 characters and average resistant text length of 154 characters. On average, around 4% of all attacks on a given text have been successful. When applying the adversarial look-up substitutions found from the val split to the test split, roughly 50% of the abusive texts are falsely classified as not abusive. On average, around 3.2 of the top 5 attacks are successful.

Table 2 shows an excerpt of the substitution dictionary.

### 4.2 Educated guesses with LIME

Unfortunately, due to time constraints, only attacks using special characters and digits could be evaluated. Of the 934 abusive texts in the val split, about 66% had at least one adversarial example. When restricting the attack area to the token with the highest abusive contribution, about 2% of all attacks on that token are successful on average. On average, 3.1 out of the top 5 attacks on a text are successful. On the test set, the substitution dictionary gathered with educated guesses drops down to 0.46, the same value as brute force.

## 5 Analysis

Table 1 compares the recall on the val and test splits using different permissible substitutions and attack methods.

Analysis with LIME reveals the most influential tokens for the model's classification. We, again, take the text "*why are you repeating yourself are you a little retarded*" as an example. Not surprisingly, LIME attributes the highest contribution to the abusive score to the token "*retarded*" with a contribution of around 0.65 to the total score of 0.79.

### 5.1 Special characters and digits

Naturally, among of the tokens in this text, "*retarded*" was the one that was most attacked during the val split attacks. Its most successful substitution was the token "{*etarded*" with 23 successful attacks. After performing the substitution and analyzing the model's classification on the attacked text, we observe an abusive score contribution by the token "etarded" of around $-0.001$, which means the "{" character has been discarded from the attacked text. Therefore, we suspect that the actual weak spot that is responsible for the success of these adversarial attacks is the model's tokenizer. BERT-style models, like the HateXplain model (Mathew et al., 2021) used in this project, use a subword-level tokenization method called WordPiece (Wu et al., 2016). The special characters in the attacked text trigger the tokenizer to break the victim word up into smaller segments that make it unrecognizable to the model.

### 5.2 Special and alphanumeric characters

Due to the aforementioned behavior of the tokenizer, attacks using special characters proved to be

| Original | #Attacks | Substitution | #Subs |
|---|---|---|---|
| nigger | 555 | ni@ger | 62 |
| | | ni:ger | 55 |
| | | ni∼ger | 38 |
| kike | 419 | {ike | 31 |
| | | ∼ike | 31 |
| | | k:ke | 22 |
| retarded | 284 | {etarded | 23 |
| | | ∼etarded | 17 |
| | | :etarded | 16 |

(a) Special characters and digits.

| Original | #Attacks | Substitution | #Subs |
|---|---|---|---|
| nigger | 578 | bigger | 44 |
| | | Bigger | 38 |
| | | digger | 38 |
| kike | 421 | {ike | 27 |
| | | ∼ike | 25 |
| | | }ke | 17 |
| retarded | 284 | }etarded | 18 |
| | | @etarded | 11 |
| | | reta@ded | 11 |

(b) Special characters, digits, and alphabet

Table 2: Substitution dictionary excerpt, gathered from the val split using brute force. Shown are the 3 most attacked original tokens with their 3 most successful substitution tokens each.

more successful than attacks using alphabetic characters. Successful attacks using alphabetic characters tend to change the victim word to another existing word. Examples include: "*why are you repeating yourself are you a little* ***retarded*** → ***rewarded***" and "*fun fact did you know kardashian is armenian for* ***nigger*** → ***bigger*** *lover*".

### 5.3 Unsuccessful attacks

Upon inspection of the unsuccessful attacks, it becomes apparent that those tend to fail because the sentences contain multiple abusive words that can't all be censored. An example of such a text is "*always thought that* ***nigger***→***nigge@*** *was a faggot*" with an abusive score of 0.97 before the attack and 0.73 after the attack. The token "*nigger*" was chosen as the victim token since it has been attacked more often during the substitution gathering phase. However, interestingly, LIME actually attributes a higher abusive score contribution to the token "*faggot*" (0.33) compared to "*nigger*" (0.26). This suggests a possible alternative attack strategy that uses LIME during test time as well to identify the most vulnerable token to substitute. To deal with such texts containing multiple potential hate speech tokens, allowing multiple substitutions would be necessary.

### 5.4 Limitations of this project

A comparison with other models was beyond the scope of this project. Training the HateXplain model on the adversarial examples was attempted but proved unfruitful. While this project makes use of the HateXplain model uploaded to huggingface[3], the authors have also open-sourced their code[4]. However, it seems the instructions to use the code were incomplete and/or out of date. After getting the code to run, an attempt at reproducing the training results on the vanilla dataset using the provided instructions reported in the paper failed with a performance no better than chance.

## 6 Discussion

LIME was instrumental in understanding the behavior of the model in reaction to the adversarial examples. I believe that interpretable machine learning will and should play an important role in making machine learning trustworthy. I maintain the position that for the general population to welcome machine learning systems in sensitive areas, they need to be consistent, robust, and trustworthy. While HateXplain is not the state-of-the-art in this discipline anymore—OpenAI's ChatGPT[5] was unsurprisingly not vulnerable to the attacks generated in this project and it should be safe to assume that the newest closed-source Large Language Models (LLMs) by OpenAI and Google outperform other current methods of the open research community—it is nonetheless surprising that a model close to the state-of-the-art of the open research community was vulnerable to such a primitive attack strategy. With LLMs regrettably going closed-source, I find it conceivable that an NLP or machine learning

[3] https://huggingface.co/Hate-speech-CNERG/bert-base-uncased-hatexplain-rationale-two, last accessed: 2023-03-30

[4] https://github.com/hate-alert/HateXplain, commit hash 01d7422

[5] https://chat.openai.com, last accessed: 2023-03-30

equivalent of a white-hat hacker might have to play a bigger role in the future. I hope that work on adversarial attacks can continue in the future in order to minimize the attack surface for bad actors.

## References


Nathaniel Berger, Stefan Riezler, Artem Sokolov, and Sebastian Ebert. 2021. Don't search for a search method–simple heuristics suffice for adversarial text attacks. *arXiv preprint arXiv:2109.07926*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 50–56. IEEE.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is bert really robust? a strong baseline for natural language attack on text classification and entailment. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 8018–8025.

Jiyun Kim, Byounghan Lee, and Kyung-Ah Sohn. 2022. Why is it hate speech? masked rationale prediction for explainable hate speech detection. *arXiv preprint arXiv:2211.00243*.

Binny Mathew, Punyajoy Saha, Seid Muhie Yimam, Chris Biemann, Pawan Goyal, and Animesh Mukherjee. 2021. Hatexplain: A benchmark dataset for explainable hate speech detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 14867–14875.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144.

Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. 2019. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841.

Zeerak Waseem and Dirk Hovy. 2016. Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In *Proceedings of the NAACL student research workshop*, pages 88–93.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Jin Yong Yoo, John X Morris, Eli Lifland, and Yanjun Qi. 2020. Searching for a search method: Benchmarking search algorithms for generating nlp adversarial examples. *arXiv preprint arXiv:2009.06368*.