Deepfake tweets detection
Project Proposal for NLP Course, Winter 2023

Adam Frej
Warsaw University of Technology
01151392@pw.edu.pl

Adrian Kamiński
Warsaw University of Technology
01151387@pw.edu.pl

Piotr Marciniak
Warsaw University of Technology
01151428@pw.edu.pl

Szymon Szmajdziński
Warsaw University of Technology
01151438@pw.edu.pl

supervisor: Anna Wróblewska
Warsaw University of Technology
anna.wroblewska1@pw.edu.pl

Abstract

This research project aims to address the growing concern of deepfake text generation, specifically focusing on detecting deepfake tweets. Deepfakes, generated using advanced machine learning techniques, have the potential to spread misinformation, impersonate individuals, and facilitate malicious activities. Detecting such deepfake content is crucial to protect society from deception and harm. The project leverages the TweepFake dataset, which contains tweets from humans and bots. It also explores various text representations and preprocessing techniques.

The research questions revolve around the development of a reliable deepfake detection algorithm. The project hypotheses involve exploring patterns, such as the use of emoticons, mentions, and misspelled words, that may indicate machine-generated tweets. Different machine learning and deep learning models were used to maximize detection accuracy while maintaining precision and recall balance.

The proposed work intends to improve state-of-the-art deepfake tweet detection and contribute to more trustworthy online interactions. Ultimately, the project seeks to enhance users' safety, trust, and confidence in their online experiences.

1 Introduction

The goal of our project is to determine if the content of a tweet is a deepfake. The term "deepfake" is a portmanteau of "deep learning" and "fakes" (Vincent, 2018). It refers to a type of synthetic content that is created using advanced machine learning techniques, particularly deep learning algorithms. Deepfakes are typically associated with manipulated videos, but they can also involve audio, images, and text. The core characteristic of a deepfake is that it convincingly alters or generates content to make it appear as if it is authentic, even when it is not. This work focuses on deepfake related to text corpora, in particular, to tweet data. This depicts human interaction in a short, dynamic form. Texts tend to be shorter and contain less context, which poses more challenges.

There are several reasons to tackle the problem of deepfake tweet detection. One of them can be protection against misinformation. Deepfake tweets can be used to spread misinformation, disinformation, and fake news, which can have serious real-world consequences, such as influencing public opinion, election outcomes, and even inciting violence. Detecting and mitigating deepfakes is crucial to protect society from being misled by false narratives. Moreover, an increasing number

of synthetic tweets can lead to a lack of trust in information shared on social media platforms.

Another reason for tackling the problem of deepfake tweet detection is the protection of individuals against impersonation, privacy violations, and identity theft. Deepfake tweets can be used to impersonate both public figures and private citizens, and by addressing this issue, we can safeguard people's rights and personal information.

Deepfakes can also be exploited for malicious purposes, including extortion, fraud, and harassment. They can be used to create a network of fake users who encourage other users to use some services. Detecting deepfakes is necessary to prevent these harmful actions.

Detection of deepfake corpora can also lead to exposure of flaws in existing text-generating algorithms, which can help to improve the language models in the future. This means creating texts that are more exciting and semantically plausible to read. Furthermore, comparing the models to human benchmarks can identify the types of errors in generated sequences that humans tend to notice and make text appear unnatural.

The proper detection of deepfake tweets can increase the trust, safety, and confidence of users in online interactions and content consumption. It can contribute to a more positive online experience.

2 Related works

The TweepFake dataset was created in (Fagni et al., 2021), where authors compared multiple approaches in both corpora encoding and algorithms to detect fake tweets. What is unique about this dataset is that the data used there was generated by various algorithms and was extracted directly from social media. The following setup can provide more generic results when evaluating models. Their results suggest that transformer-based models offer better results. They also found out that all approaches struggled with tweets generated by GPT-2 (Radford et al., 2019), and the best results were obtained using an RNN decoder using character encoding.

One of the main difficulties in natural language processing is text representation. Related works investigate several solutions to this problem in the context of feeding data models detecting machine-generated corpora. The TweepFake (Fagni et al., 2021) article proposes 4 options. One of them is a popular method, bag-of-words (BoW), with features weighted using TF-IDF function (Sebastiani, 2002). The output of this methodology was processed by either logistic regression (Cox, 1958), random forest (Ho, 1995), or SVM (Cortes and Vapnik, 1995). However, this approach suffers from the curse of dimensionality, as the features are very sparse and require a lot of data. Another drawback is that the algorithm misses the semantic context of the words. The article got the worst result here, hovering around 0.80 accuracy.

Another option is to encode text using a high-level language model, which overcomes these limitations. The TweepFake used BERT (Devlin et al., 2019) to provide contextual embeddings that include words context and can be encoded into a vector representing specific text. Yet again, this representation is later processed by classifiers.

The third approach operates on the character level. A vocabulary of characters is mapped to the internal embedding matrix, which is passed to the selected deep learning networks. This results in a surprisingly good score, up to 0.85 accuracy, and can be useful in cases without pretrained models available, for example, another language.

However, the most successful approach is utilizing pre-trained models. They take raw input directly and, with fine-tuning on a given dataset, solve the classification problem of labeling a sentence as human or machine-generated. This way, the models operate on the complex sequence-based understanding level. The TweepFake tested several language models, all related to BERT. Besides the original BERT, XLNet (Yang et al., 2020) and RoBERTa (Liu et al., 2019) were utilized as they can produce 15% better results thanks to architecture modifications and a bigger training dataset. The article also tested DistilBERT (Sanh et al., 2020), which tries to keep performance while simplifying the architecture and halving the number of parameters. Generally, the results get up to 0.90 accuracy, the best being RoBERTa, which indicates that the most complex representations are most effective. They are also well-balanced in terms of precision and recall.

One of the works tried to detect generated text by exploiting situations when humans are fooled by it (Ippolito et al., 2020). They also investigated several text representations. Similarly, the primary one is a fine-tuned BERT (Devlin et al., 2019). Again, this approach far surpassed other

methods depicted in the article.

Another representation in this work is a simple BoW. This time, the GPT-2's $50{,}000$ token vocabulary (Sennrich et al., 2016) is used to count the occurrences of tokens in text sequences. This embedding allowed for training logistic regression binary classifier, which achieved the next best result.

The article also proposed Histogram-of-Likelihood Ranks. As in GLTR (Gehrmann et al., 2019), they created an energy-based deepfake text decoder by calculating the probability distribution of the next word given the previous words in the sequence according to GPT-2 language model. They ranked the words by likelihood and then binned them either into 4 groups or uniformly over the whole vocabulary. Such histograms served as input for logistic regression binary classifiers. Unfortunately, this approach resulted in worse scores despite successes reported in GLTR, which can be explained by training data selection.

Another deepfake detection performed on social media texts was conducted on Amazon reviews in (Adelani et al., 2019). Fake news were generated using the GPT-2 text generation model (Radford et al., 2019). Authors, in order to adjust the model to new corpora, adapted the original GPT-2 model to Amazon (He and McAuley, 2016) and Yelp reviews (Zhang et al., 2015). As for detection algorithms, they used Grover (Zellers et al., 2019a), GLTR (Gehrmann et al., 2019), and OpenAI GPT-2 (Solaiman et al., 2019). They also tried combining those models using logistic regression at the score level. One of the experiments they performed was selecting a real review out of 4, of which 3 were fake. Human participants tended to randomly guess which review was real since they were right about 25% of the time. However, the models were not much better. The best configuration achieved 20% accuracy on this task.

In (Guo et al., 2023), authors proposed HC3 dataset consisting of questions and their corresponding human/ChatGPT answers. Based on this dataset, they conducted a comprehensive human evaluation and linguistic analysis as well as developed several detecting models. They used the GLTR model with logistic regression, RoBERTa, and RoBERTa-QA - a Question Answering version of the model that supports a text pair input format, where a separating token is used to join a question and its corresponding answer. In their work, they came to the following conclusions:

- The robustness of the RoBERTa-based-detector is better than GLTR.
- RoBERTa is not affected by indicating words (characteristic words for ChatGPT).
- RoBERTa is effective in handling Out-Of-Distribution scenarios, whereas we can observe a significant decrease in performance on GLTR's when testing on data in first-seen format.
- Detecting ChatGPT-generated texts is more difficult in a single sentence than in a full text.

3 Datasets

Several datasets were created to build a solution to detect content created by deep learning algorithms. Some of them are presented below.

1. The TweepFake dataset (Fagni et al., 2021) - it is a dataset which contains $25{,}572$ tweets half human and half bots generated. They used 17 human accounts, which were imitated by the 23 bots. Some of the fake accounts imitated the same human profile. These bots were using different technologies, and for almost all of them (except one bot account), the used technology is known.

2. The GPT-2 output datasets (Ippolito et al., 2020) - it is a group of several datasets in which deepfakes are generated by GPT-2 models (Radford et al., 2019). The datasets differ because different decoding strategy settings and different sizes of models are applied during generation. Each dataset contains $500{,}000$ training and $5{,}000$ validation and test samples, which are evenly spread across classes (human-generated excerpts of web texts or GPT-2 generated).

3. The HC3-English datasets (Guo et al., 2023) - it is a group of datasets from different sources, in which for each question, there is provided at least one human, and ChatGPT3.5 answer. The questions and answers by the human experts come mainly from publicly available question-answering datasets. There is also an additional source in which Wikipedia is treated as a human expert who is asked questions based on concepts in crawled data.

In most papers (Zellers et al., 2019b; Bakhtin et al., 2019), datasets containing generated and human texts are not provided because big corpora are used to build a generative model in which descriptors act as detectors of deepfakes. Later, they test their descriptors on unused parts of corpora and text produced by generators to see if the descriptors are working in the correct way.

4 Approach & research methodology

Our research on detecting deep fakes of tweets was performed on the TweepFake dataset. We also tried to use one of the GPT-2 output datasets to improve the detection of deep fakes prepared by the GPT-2, which caused the most problems in the original work. To test our solution, we used the split provided by the authors of the dataset. They performed a stratified split into 3 datasets (training, validation, test), which roughly reminds the 80%-10%-10%. By doing it this way, later, we can compare our results with the ones achieved in the article.

In our project, we stated several research questions, which set the direction of the experiments:

- Can we build a reliable deepfake detection algorithm? By reliable algorithm, we mean the model that will maximize accuracy on a balanced dataset while remaining well-balanced in terms of precision and recall. That means detecting generated tweets while avoiding assigning false positives.
- What are the most effective embeddings for deepfake detection in tweets?
- Are there any patterns that indicate the model-generated tweet content?

There are also some hypotheses which we tested:

- The use of emoticons may be higher in human-generated content.
- The use of mentions of other users may be higher in human-generated content.
- There will be more misspelled words in content generated by bots.
- The impact of different URL encoding, e.g., encoding all URLs to a single token vs extracting the basepath of the URLs.

In order to investigate and answer these hypotheses, we conducted explanatory data analysis (EDA). This also allowed us to gain insight into data and conduct further preprocessing steps. We assessed the most basic data properties. All splits are balanced, both in terms of human - non-human content and different generative methods in a non-human class. The tweet lengths are also similar across different classes. Using the Natural Language Toolkit (NLTK) (Bird et al., 2009), we tokenized the tweets and counted word occurrences. As expected, the most frequent are stop words (e.g. *the*, *to*, *a*), which we removed for later research. Lastly, the sentiment analysis of tweets reports quite balanced and focused around neutral values polarity and subjectivity in different classes with regard to splits. For this, we used *SpacyTextBlob* from spaCy package (Honnibal et al., 2020).

Regarding the hypotheses, we checked them by simple aggregations performed on texts.

1. **Emoticons**: most tweet do not contain any. However, if there appeared some, they were mostly human generated, which roughly confirms the hypothesis. Importantly, the main generative methods gpt2 and rnn were least likely to use any. There were some algorithms in *other* category which did use emojis.

2. **Mentions**: again, most tweets do not contain them. Here, the difference is more apparent. Humans are almost the only ones to use mentions, directly confirming the hypothesis.

3. **Misspells**: they are mainly introduced by humans. However, the situation is more balanced. The tweets without mistakes are not dominant and are mainly made by bots. Generally, the more mistakes in a tweet, the more likely it is to be created by a human. The hypothesis appears to be untrue. We used pyspellchecker (Barrus, 2019) with basic English dictionary *en_core_web_sm* from spaCy package (Honnibal et al., 2020). Perhaps the human texts contain more slang and informal speech, which technically contains misspells.

4. **URLs**: similarly to mentions, most tweets do not contain them and humans generate most URLs. Only rnn tried to introduce some. This makes the hypothesis obsolete, as the

impact of encoding does not matter when generated texts can not be encoded.

Generally, excluding misspells, all the discussed properties rarely appear. Usually, the tweets do not contain them. Therefore, despite assessing the hypotheses as true or not, they are not very significant to our research.

We applied some simple preprocessing steps to the texts of the tweets. Using again NLTK and other basic Python libraries, we tagged URLs and mentions. As reported in EDA, they did not contain meaningful information. Then, we tokenized the tweets and removed stop words. We also created two versions of the data, with applied stemming or lemmatization. Both are later tested.

We explored five different approaches to finding the best model for this task. The first approach is based on text representation with a bag of words encoded with the TF-IDF function. Next, tweets encoded this way were processed by machine-learning algorithms. In this project, we use five well-known classifiers: Light Gradient Boosting Machine (LightGBM) (Ke et al., 2017), eXtreme Gradient Boosting (XGBoost) (Chen and Guestrin, 2016), Random Forest (Ho, 1995), Logistic Regression (Cox, 1958), Support Vectors Machine (SVM) (Cortes and Vapnik, 1995).

As the first approach often suffers from the curse of dimensionality because feature space after encoding is sparse and does not take into account the word order, in the second approach, we prepared the feature space using BERT (Devlin et al., 2019). BERT provides contextual embeddings, i.e. fixed-size vectors representing words which depend on the context in which the word occurs. We averaged these embeddings to obtain a fixed-size vector of a tweet text, which depends on context and word order. As in the previous scenario, the tweers encoded by BERT have been later used to learn the same set of classifiers.

In both the first two approaches, the best hyperparameters for our ML pipeline (we optimized parameters also for the TF-IDF Vectorizer) were found with the help of the Optuna package (Akiba et al., 2019). As we wanted to get the best hyperparameters for our problem in these approaches, we merged our training and validation dataset to obtain a more extensive dataset. Later, during the search for hyperparameters, this bigger dataset was split into five stratified folds. Optuna was trying to optimize the mean of the balanced accuracy metric calculated on the test fold, which, with the stratified approach, should have very close results to simple accuracy. We set some restrictions within Optuna. Each ML pipeline takes 20 minutes, and maximally, 100 tries to find the optimal hyperparameters. The hyperparameters and their value space are presented in Table 4.

In the third and fourth approaches, we used deep learning techniques to build the models to solve our problem. In the third scenario, we encoded text by working at the character level. We mapped the text of the tweets to lowercase and tokenized the letters. After tokenization, we padded sequences to the maximum length of 320 characters.

In the fourth scenario, we applied token encoding of the text. We mapped the text of the tweets to lowercase and removed punctuation. The tokenizer was using the $15,000$ the most occurring tokens and the maximal length of the sequence was 100 tokens.

In both DL approaches, we used similar architectures of deep neural networks. They differed only in the number of filters in the convolution layer and the number of units in the GRU layer. The approach with tokens, in most cases, used the double of filters/units in their architectures. We checked three types of architectures:

- CNN - in this architecture, convolutional layers on the activation matrix coming from the embedding layer are used; after this layer, we performed global max pooling and used a dense layer to classify,
- GRU - in this architecture, we used only a bidirectional GRU layer on the activation matrix; after this layer, we performed global max pooling and used a dense layer to classify,
- CNN+GRU - in this architecture, we combined the strength of both architectures; we concatenated the outputs from the CNN layers and GRU layer to perform the classification.

In all DL approaches, we shared the following configuration:

- batch size: 256
- epochs: 200 with early-stopping (10 epochs patience)

- save: model from best (in regard to loss function) and last evaluation step
- optimizer: Adam
- learning rate: 1e-4
- loss: CrossEntropy

In the last approach, we focused on larger models. Finally, we tried the following architectures:

- xlm-roberta-base (Conneau et al., 2019) - XLM-RoBERTa-Base is a cross-lingual pre-trained language model based on the RoBERTa architecture, designed to understand and generate text in multiple languages with improved performance and efficiency.
- distilbert-base-uncased (Sanh et al., 2020) - DistilBERT-Base-Uncased is a distilled version of BERT, optimized for speed and efficiency while retaining much of the original model's performance.
- gpt2 (Radford et al., 2019) - GPT-2, or Generative Pre-trained Transformer 2, is a powerful autoregressive language model that excels in generating coherent and contextually relevant text.

The experiments shared the following configuration (unless otherwise stated in the method description):

- batch size: 8
- epochs: 10 with early-stopping (2 epochs patience)
- save: model from best (in regard to loss function) and last evaluation step
- weight decay: 0.01
- optimizer: AdamW
- loss: CrossEntropy
- seed: 1

We started with XLM-RoBERTa and in this approach, we trained the network 3 times, each time switching hyperparamers. First, we tried freezing layers that were not responsible for final classification (this approach will be denoted by XLM0) and without it (XLM1). After obtaining results, it was clear that freezing so many layers did not help model performance, so we discarded this idea from further training configurations. For XLM-RoBERTa, we additionally tried another value learning rate hyperparameter. In XLM1, the value $2 \cdot 10^{-5}$ was used and in new approach XLM2, we tried lower value i.e. $2 \cdot 10^{-6}$. The second approach proved to be better, and that value was used in all later experiments.

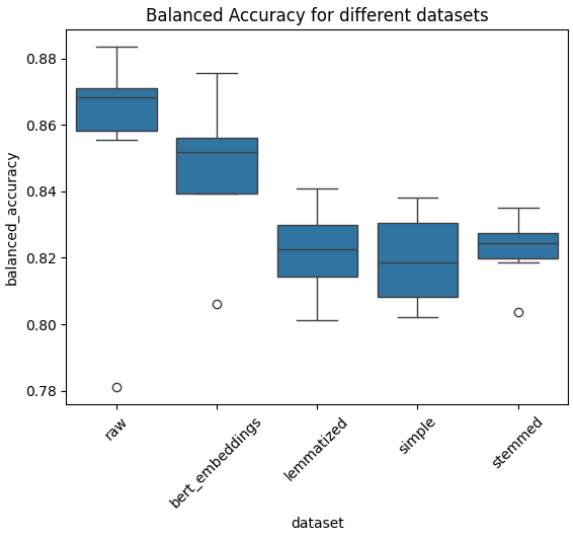

Figure 1: Balanced Accuracy for different datasets

After that, we moved to the DistilBERT model. Since this model is relatively small and fast, we decided to check how the learning process would look when we included the GPT2 output dataset to pretrain the model. In the pretraining process, we had to limit the number of steps (evaluation of the whole dataset once would take over 6 hours). We ended up with one epoch and evaluation after 2000 steps and early stopping with (5 evaluations step patience). However, the additional pretraining on this dataset did not improve the results, so we didn't repeat this process again on larger models.

The last architecture we trained was GPT2. We trained this model once with a setup that proved best in previous examples.

5 Results

To evaluate the results of the models described above, we used several metrics, i.e., accuracy, balanced accuracy, precision, recall and f1. The evaluation was done on the testing dataset ($\sim$ 10% of the whole dataset that was not used during the training/validation process).

Table 1: Table with 10 best approaches with respect to balanced accuracy

ba	f1	precision	recall	model	dataset
0.8835	**0.8821**	**0.8934**	0.8711	XLM2	raw
0.8757	0.8763	0.8729	0.8797	SVC	bert
0.8713	0.8786	0.8328	**0.9297**	XLM1	raw
0.8698	0.8686	0.8773	0.8602	DISTIL_BERT0	raw
0.8671	0.8686	0.8593	0.8781	GPT2	raw
0.8561	0.8590	0.8429	0.8758	LGBM	bert
0.8554	0.8529	0.8681	0.8383	DISTIL_BERT1	raw
0.8518	0.8546	0.8395	0.8703	XGB	bert
0.8408	0.8518	0.7975	0.9141	CharCNN+GRU	lemmatized
0.8393	0.8416	0.8304	0.8531	LR	bert

Figure 1 presents the results of different models with respect to the type of data used to train them. We can see that raw data achieved the best results. This data was only used by transformers. Bert embedding has also achieved high results with respect to the accuracy score. The other 3 datasets, i.e., lemmatized, simple and stemmed, used either TF-IDF or DL embedding created only on the data from TweepFake. We can see that those embeddings proved to perform significantly worse.

Table 1 illustrates the balanced accuracy of the best 10 models. The transformers models (XLM, Distil-Bert, GPT2) achieved the most favorable outcomes when trained on the original dataset. Notably, the second-best performing model was the SVC model utilizing embeddings generated by Bert. In a comparison with (Fagni et al., 2021), our findings align closely, with the transformer models' results proving comparable to those reported in the referenced article. Notably, we achieved superior results with the SVC model, possibly attributable to variations in data preprocessing methods.

Table 2 provides insights into the overall accuracy and specific accuracies categorized by the method used to create the deepfakes. Notably, detecting deepfakes generated by the GPT2 model was the most challenging task. Tweets generated by humans ranked as the second most challenging category, although their accuracies were significantly higher than those for GPT2-generated tweets. On the other hand, tweets generated using RNNs and alternative methods are generally easy to detect, having notably high accuracies.

The XML model trained with a lower learning rate (XLM2) achieved the highest global accuracy. Interestingly, the XML model without a lower learning rate (XML1) yielded the best results for identifying GPT2-generated deepfakes, while maintaining relatively high scores for other methods.

The detailed accuracies for each class type and model are presented in Figure 2. Notably, RNN and other classes emerge as the easiest to detect, with accuracies around 95% for RNNs and 92% for other class types. The heatmap further highlights the challenge in detecting deepfakes generated by GPT2, as they exhibit lower detection rates. Transformers, on the other hand, consistently showcase the highest accuracies within this class type.

Compared to the findings presented in the paper (Fagni et al., 2021), our study achieved superior results in detecting deepfakes generated by GPT2, all while maintaining a high overall accuracy. Given that detecting GPT2-generated deepfakes was identified as the most challenging aspect of the task, this accomplishment is considered a success in our research.

6 Conclusions

We have stated research questions which defined the experiments. We managed to investigate them and generally, the answers are positive. We improved the results obtained on GPT-2 generated text by a considerable margin without a drop in global accuracy. We can say we built a well-performing deepfake detection algorithm. We also found out that effective embeddings are created by high-level deep learning models like BERT. Finally, we recognized the basic patterns of artificially generated texts, i.e. lack of emoticons, mentions, misspells and URLs.

Future works

The interesting thing would be to extend the current TweepFake dataset with more examples and use more state-of-the-art models to generate fakes

Table 2: Table with 10 best approaches with respect to global accuracy

bot_type model_name	all	gpt2	human	others	rnn
XLM2_raw	**0.8835**	0.6953	**0.8959**	0.9153	0.9830
SVC_bert_embeddings	0.8757	0.6927	0.8717	0.9442	0.9782
XLM1_raw	0.8714	**0.8307**	0.8130	0.9607	0.9854
DisitlBERT0_raw	0.8698	0.6589	0.8795	0.9112	**0.9879**
GPT2_raw	0.8671	0.6693	0.8560	0.9587	0.9782
LGBM_bert_embeddings	0.8561	0.6745	0.8365	0.9483	0.9782
DistilBERT1_raw	0.8554	0.6849	0.8725	0.8471	0.9709
XGB_bert_embeddings	0.8518	0.6562	0.8333	0.9525	0.9733
CharCNN_GRU_lemmatized	0.8409	0.7760	0.7676	**0.9628**	0.9854
LR_bert_embeddings	0.8393	0.6380	0.8255	0.9236	0.9709

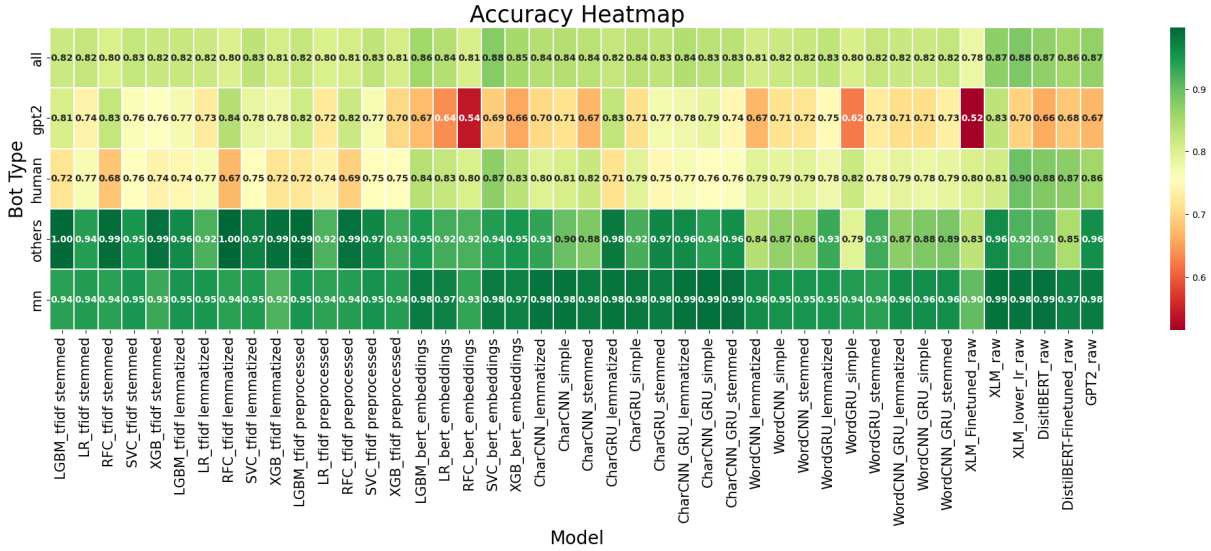

Figure 2: Accuracy for class type and every model

like GPT-3.5 (Brown and et al., 2020), GPT-4 (OpenAI et al., 2023), or Gemini (lastly announced Google solution). Unfortunately, with the change in X API policy (formerly known as Twitter), it is much harder because there is no free plan to read tweets, and paid plans are much more restricted than earlier ($3,000$ reads per month on basic plan). Before that, you could read $300,000$ tweets per month on a free plan and even $10,000,000$ after applying for a scientific plan. That said, the existing examples with proper prompt engineering can be used with state-of-the-art models to generate even more challenging deepfakes.

Another thing which could be checked with TweepFake is trying to fine-tune more state-of-the-art and bigger LLM like LLaMA (Touvron et al., 2023) or any of the earlier mentioned solutions.

Table 3: Contribution table (code in this table adheres to the number within the name of notebook)

person	task	t[h]
Adam Frej	Introduction	3
	Related works (research)	9
	Concept and work plan	4
	Rebuttal I	1
	review I Team 4	2
	Approach	5
	Conclusions	1
	code: eda preprocessing	4
Adrian Kamiński	Related works	6
	Datasets (research)	1.5
	Concept and work plan	4
	Rebuttal I	1
	review I Team 4	2
	Approach	3

person	task	t[h]
	Results	3
	Conclusions	1
	code 01: GPT-2 download, eda	2
	code 41-43,49: transformers	10
	code 91: results	3
Piotr Marciniak	Introduction	4
	Datasets (writing & research)	3
	Concept and work plan	3
	Approach	3
	Rebuttal I	2
	review I Team 10	2.5
	code 01-12	10
	code 21-23 (modifications)	1.5
	code 31-33	5
	code 90	3
Szymon Szmajdziński	Related works (research)	4
	Datasets (research)	3
	Concept	3
	Abstract	1
	review I Team 10	2.5
	code 21-23: Deep Learning Models	10
	Results	5
	Conclusions	1

References

David Ifeoluwa Adelani, Haotian Mai, Fuming Fang, Huy H. Nguyen, Junichi Yamagishi, and Isao Echizen. 2019. Generating sentiment-preserving fake online reviews using neural language models and their human- and machine-based detection. *CoRR*, abs/1907.09177.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Anton Bakhtin, Sam Gross, Myle Ott, Yuntian Deng, Marc'Aurelio Ranzato, and Arthur Szlam. 2019. Real or fake? learning to discriminate machine from human generated text. *CoRR*, abs/1906.03351.

Tyler Barrus. 2019. pyspellchecker. `https://pypi.org/project/pyspellchecker/`. accessed: 09.12.2023. [Online].

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition.

Tom B. Brown and Benjamin Mann et al. 2020. Language models are few-shot learners.

Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Unsupervised cross-lingual representation learning at scale. *CoRR*, abs/1911.02116.

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning*, 20(3):273–297.

David R Cox. 1958. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Tiziano Fagni, Fabrizio Falchi, Margherita Gambini, Antonio Martella, and Maurizio Tesconi. 2021. Tweepfake: About detecting deepfake tweets. *PLOS ONE*, 16(5):1–16, 05.

Sebastian Gehrmann, Hendrik Strobelt, and Alexander M. Rush. 2019. Gltr: Statistical detection and visualization of generated text.

Biyang Guo, Xin Zhang, Ziyuan Wang, Minqi Jiang, Jinran Nie, Yuxuan Ding, Jianwei Yue, and Yupeng Wu. 2023. How close is chatgpt to human experts? comparison corpus, evaluation, and detection.

Ruining He and Julian J. McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. *CoRR*, abs/1602.01585.

Tin Kam Ho. 1995. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spaCy: Industrial-strength Natural Language Processing in Python.

Daphne Ippolito, Daniel Duckworth, Chris Callison-Burch, and Douglas Eck. 2020. Automatic detection of generated text is easiest when humans are fooled. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings*

of the 58th Annual Meeting of the Association for Computational Linguistics, pages 1808–1822, Online, July. Association for Computational Linguistics.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

OpenAI, :, Josh Achiam, Steven Adler, and et al. 2023. Gpt-4 technical report.

Alec Radford, Jeff Wu, Rewon Child, D. Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Fabrizio Sebastiani. 2002. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, mar.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units.

Irene Solaiman, Miles Brundage, Jack Clark, Amanda Askell, Ariel Herbert-Voss, Jeff Wu, Alec Radford, Gretchen Krueger, Jong Wook Kim, Sarah Kreps, et al. 2019. Release strategies and the social impacts of language models. *arXiv preprint arXiv:1908.09203*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models.

James Vincent. 2018. Why we need a better definition of 'deepfake' / let's not make deepfakes the next fake news. `https://www.theverge.com/2018/5/22/17380306/deepfake-definition-ai-manipulation-fake-news`, May.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2020. Xlnet: Generalized autoregressive pretraining for language understanding.

Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. 2019a. Defending against neural fake news. *CoRR*, abs/1905.12616.

Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. 2019b. Defending against neural fake news. *CoRR*, abs/1905.12616.

Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626.

Appendix

Table 4: The table presenting model and encoder hyperparameters which were optimized by the optuna package (Integer - means that values are integers from the given range, Float - floats from the given range, Categorical - category from the given list, log=True - means that the values are drawn from log-uniform distribution between given range)

model / encoder	hyperparameter	value space
LGBM	Boosting Type	Categorical([gbdt, dart])
	max_depth	Integer(1, 15)
	n_estimators	Integer(10, 500, log=True)
	subsample	Float(0.6, 1)
XGB	booster	Categorical([gbtree, dart])
	max_depth	Integer(1, 15)
	n_estimators	Integer(10, 500, log=True)
	subsample	Float(0.6, 1)
RF	max_depth	Integer(1, 15)
	n_estimators	Integer(10, 500, log=True)
	criterion	Categorical([gini, entropy, log_loss])
	min_samples_split	Float(0.01, 0.1)
SVC	kernel	Categorical([linear, poly, rbf, sigmoid])
	C	Float(1e-2, 1e2, log=True)
Logistic regression	penalty	Categorical([l1, l2])
	C	Float(1e-2, 1e2, log=True)
TF-IDF	ngram_range	Categorical([(1, 1), (1, 2), (1, 3)])
	max_features	Integer(1000, 10000, log=True)
	max_df	Float(0.8, 1)
	min_df	Float(0.0, 0.2)

Table 5: Table with results of all experiments with with more specific metrics

balanced_accuracy	f1_score	precision	recall	model	dataset
0.8835	0.8821	0.8934	0.8711	XLM2	raw
0.8757	0.8763	0.8729	0.8797	SVC	bert_embeddings
0.8713	0.8786	0.8328	0.9297	XLM1	raw
0.8698	0.8686	0.8773	0.8602	DISTIL_BERT0	raw
0.8671	0.8686	0.8593	0.8781	GPT2	raw
0.8561	0.8590	0.8429	0.8758	LGBM	bert_embeddings
0.8554	0.8529	0.8681	0.8383	DISTIL_BERT1	raw
0.8518	0.8546	0.8395	0.8703	XGB	bert_embeddings
0.8408	0.8518	0.7975	0.9141	CharCNN+GRU	lemmatized
0.8393	0.8416	0.8304	0.8531	LR	bert_embeddings
0.8385	0.8451	0.8125	0.8805	CharCNN	lemmatized
0.8381	0.8432	0.8184	0.8695	CharCNN	simple
0.8354	0.8420	0.8101	0.8766	CharGRU	simple

Continued on next page

balanced_accuracy	f1_score	precision	recall	model	dataset
0.8350	0.8374	0.8260	0.8492	CharCNN	stemmed
0.8330	0.8442	0.7919	0.9039	CharCNN+GRU	stemmed
0.8322	0.8444	0.7881	0.9094	CharCNN+GRU	simple
0.8311	0.8433	0.7873	0.9078	SVC	lemmatized
0.8291	0.8423	0.7827	0.9117	CharGRU	stemmed
0.8287	0.8373	0.7982	0.8805	WordGRU	lemmatized
0.8287	0.8407	0.7864	0.9031	SVC	simple
0.8260	0.8373	0.7869	0.8945	SVC	stemmed
0.8248	0.8315	0.8019	0.8633	WordCNN+GRU	stemmed
0.8244	0.8323	0.7974	0.8703	WordGRU	stemmed
0.8244	0.8340	0.7916	0.8812	LR	stemmed
0.8244	0.8421	0.7658	0.9352	CharGRU	lemmatized
0.8236	0.8402	0.7686	0.9266	LGBM	simple
0.8225	0.8356	0.7787	0.9016	LGBM	lemmatized
0.8209	0.8257	0.8049	0.8477	WordCNN	stemmed
0.8201	0.8256	0.8019	0.8508	WordCNN+GRU	lemmatized
0.8189	0.8325	0.7751	0.8992	XGB	stemmed
0.8186	0.8241	0.8004	0.8492	WordCNN	simple
0.8186	0.8276	0.7890	0.8703	LR	lemmatized
0.8185	0.8356	0.7646	0.9211	LGBM	stemmed
0.8178	0.8244	0.7962	0.8547	WordCNN+GRU	simple
0.8100	0.8134	0.7998	0.8273	WordCNN	lemmatized
0.8092	0.8256	0.7609	0.9023	XGB	lemmatized
0.8084	0.8185	0.7782	0.8633	XGB	simple
0.8080	0.8278	0.7511	0.9219	RF	simple
0.8061	0.8069	0.8043	0.8094	RF	bert_embeddings
0.8045	0.8019	0.8135	0.7906	WordGRU	simple
0.8037	0.8251	0.7447	0.9250	RF	stemmed
0.8021	0.8140	0.7688	0.8648	LR	simple
0.8013	0.8243	0.7395	0.9313	RF	lemmatized
0.7811	0.7764	0.7941	0.7594	XLM0	raw

Table 6: Table with results of all experiments with split distinguishing the bot type responsible for the creation of twitter posts.

bot_type model_name	all	gpt2	human	others	rnn
XLM2_raw	0.8835	0.6953	0.8959	0.9153	0.9830
SVC_bert_embeddings	0.8757	0.6927	0.8717	0.9442	0.9782
XLM1_raw	0.8714	0.8307	0.8130	0.9607	0.9854
DisitlBERT0_raw	0.8698	0.6589	0.8795	0.9112	0.9879
GPT2_raw	0.8671	0.6693	0.8560	0.9587	0.9782
LGBM_bert_embeddings	0.8561	0.6745	0.8365	0.9483	0.9782
DistilBERT1_raw	0.8554	0.6849	0.8725	0.8471	0.9709
XGB_bert_embeddings	0.8518	0.6562	0.8333	0.9525	0.9733
CharCNN_GRU_lemmatized	0.8409	0.7760	0.7676	0.9628	0.9854

Continued on next page

bot_type model_name	all	gpt2	human	others	rnn
LR_bert_embeddings	0.8393	0.6380	0.8255	0.9236	0.9709
CharCNN_lemmatized	0.8385	0.7031	0.7966	0.9339	0.9830
CharCNN_simple	0.8382	0.7135	0.8067	0.8967	0.9830
CharGRU_simple	0.8354	0.7109	0.7942	0.9194	0.9806
CharCNN_stemmed	0.8350	0.6693	0.8208	0.8822	0.9782
CharCNN_GRU_stemmed	0.8331	0.7448	0.7621	0.9587	0.9879
CharCNN_GRU_simple	0.8323	0.7865	0.7551	0.9380	0.9903
SVC_tfidf lemmatized	0.8311	0.7839	0.7543	0.9731	0.9466
CharGRU_stemmed	0.8292	0.7734	0.7465	0.9669	0.9757
WordGRU_lemmatized	0.8288	0.7474	0.7770	0.9277	0.9490
SVC_tfidf preprocessed	0.8288	0.7682	0.7543	0.9669	0.9539
SVC_tfidf stemmed	0.8260	0.7630	0.7574	0.9525	0.9490
WordCNN_GRU_stemmed	0.8249	0.7318	0.7864	0.8864	0.9587
WordGRU_stemmed	0.8245	0.7266	0.7786	0.9256	0.9393
LR_tfidf stemmed	0.8245	0.7370	0.7676	0.9421	0.9442
CharGRU_lemmatized	0.8245	0.8281	0.7136	0.9793	0.9830
LGBM_tfidf preprocessed	0.8237	0.8151	0.7207	0.9938	0.9515
LGBM_tfidf lemmatized	0.8225	0.7708	0.7433	0.9628	0.9515
WordCNN_stemmed	0.8210	0.7161	0.7942	0.8636	0.9515
WordCNN_GRU_lemmatized	0.8202	0.7083	0.7895	0.8719	0.9587
XGB_tfidf stemmed	0.8190	0.7578	0.7387	0.9876	0.9272
LGBM_tfidf stemmed	0.8186	0.8073	0.7160	0.9959	0.9393
LR_tfidf lemmatized	0.8186	0.7266	0.7668	0.9174	0.9490
WordCNN_simple	0.8186	0.7135	0.7879	0.8678	0.9539
WordCNN_GRU_simple	0.8178	0.7083	0.7809	0.8822	0.9587
WordCNN_lemmatized	0.8100	0.6693	0.7926	0.8388	0.9612
XGB_tfidf lemmatized	0.8092	0.7786	0.7160	0.9897	0.9150
XGB_tfidf preprocessed	0.8084	0.7005	0.7535	0.9256	0.9417
RFC_tfidf preprocessed	0.8081	0.8177	0.6941	0.9876	0.9417
RFC_bert_embeddings	0.8061	0.5417	0.8028	0.9174	0.9320
WordGRU_simple	0.8045	0.6224	0.8185	0.7934	0.9442
RFC_tfidf stemmed	0.8038	0.8255	0.6823	0.9917	0.9393
LR_tfidf preprocessed	0.8022	0.7240	0.7394	0.9153	0.9369
RFC_tfidf lemmatized	0.8014	0.8385	0.6714	0.9979	0.9393
XLM0_raw	0.7811	0.5156	0.8028	0.8306	0.9029