Dependable A.I.

Minor 1 Report

SHUBHAM KUMAR

(B20AI039)

ATTACKING NEURAL TEXT DETECTORS

`Paper link –` https://arxiv.org/pdf/2002.11768v4.pdf

Abstract of the Paper

Machine learning based language models have recently made significant progress, which introduces a danger to spread misinformation. To combat this potential danger, several methods have been proposed for detecting text written by these language models. This paper presents black-box attacks on these detectors, by randomly replacing characters with homoglyphs.These types of attacks are very successful for text generated by GPT-2. Results also indicate that the attacks are transferable to other neural text detectors.

RESOURCES USED

- `Paper link –` https://arxiv.org/pdf/2002.11768v4.pdf
- GPT 2 output dataset - https://github.com/openai/gpt-2-output-dataset.

  From a 5000 text dataset, we have chosen around 300 texts randomly. (different from paper).

- Hugging face API for roBERTa neural text detector - https://api-inference.huggingface.co/models/roberta-base-openai-detector

HOMOGLYPH ATTACK

Homoglyph attack is a non-human-like attack, which imperceptibly (according to humans) changes neural text in a way that humans normally would not. This class of attack shifts the modified text's distribution away from its original one. In this work, the non-human-like attacks are realized by swapping selected characters with Unicode homoglyphs (e.g. changing English "a"s to Cyrillic "a"s throughout a neural text sample). Homoglyphs are chosen because they appear visually similar to their counterparts, but get tokenized differently by neural text detectors.

Following are the homoglyphs used in our code for attacking the detector:

```
homoglyphs = {
    'a': 'а',
    'e': 'е',
    'i': 'і',
    'o': 'о',
    'u': 'υ',
    'b': 'Ь',
    'c': 'с',
    'h': 'һ',
    'k': 'к',
    'm': 'м'
}
```

Also following function changes the text to similar looking homoglyph containing text:

```python
def replace_with_homoglyph(text, percentage, alphabets_to_replace):
    # Calculate the maximum number of homoglyph replacements for each alphabet
    max_replacements = {}
    for char in alphabets_to_replace:
        char_count = text.count(char)
        max_replacements[char] = int(char_count * percentage)

    replacements = {char: 0 for char in alphabets_to_replace}
    new_text = ""

    for char in text:
        if char in alphabets_to_replace and replacements[char] < max_replacements[char] and random.random() < 0.5:
            new_char = homoglyphs.get(char, char)
            replacements[char] += 1
        else:
            new_char = char
        new_text += new_char

    return new_text
```

RoBERTa Detector

We have used roBERTa detector API to get the probability of text generated by the machine or not. Note that this detector is only good for text generated by GPT-2. It is not trained for CHAT-GPT.

```python
# Define the API endpoint and the model name
API_URL = "https://api-inference.huggingface.co/models/roberta-base-openai-detector"
HEADERS = {"Authorization": "Bearer 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"}

def probability_human_text(text):
    payload = {"inputs": text}
    response = requests.post(API_URL, headers=HEADERS, data=json.dumps(payload))


    real_score = 0

    try:
      output = json.loads(response.content.decode("utf-8"))
      if output[0][0]['label'] == 'Real':
        real_score = output[0][0]['score']
      else:
        real_score = output[0][1]['score']
    except:
      # default score when api returns null obect
      real_score = .5

    return round(real_score*100,2)
```

I have done 5 experiments which is discussed below:

EXPERIMENT 1

Objective :

In this experiment, we will try to find the most effective homoglyph pair for attacking a Roberta neural detector.

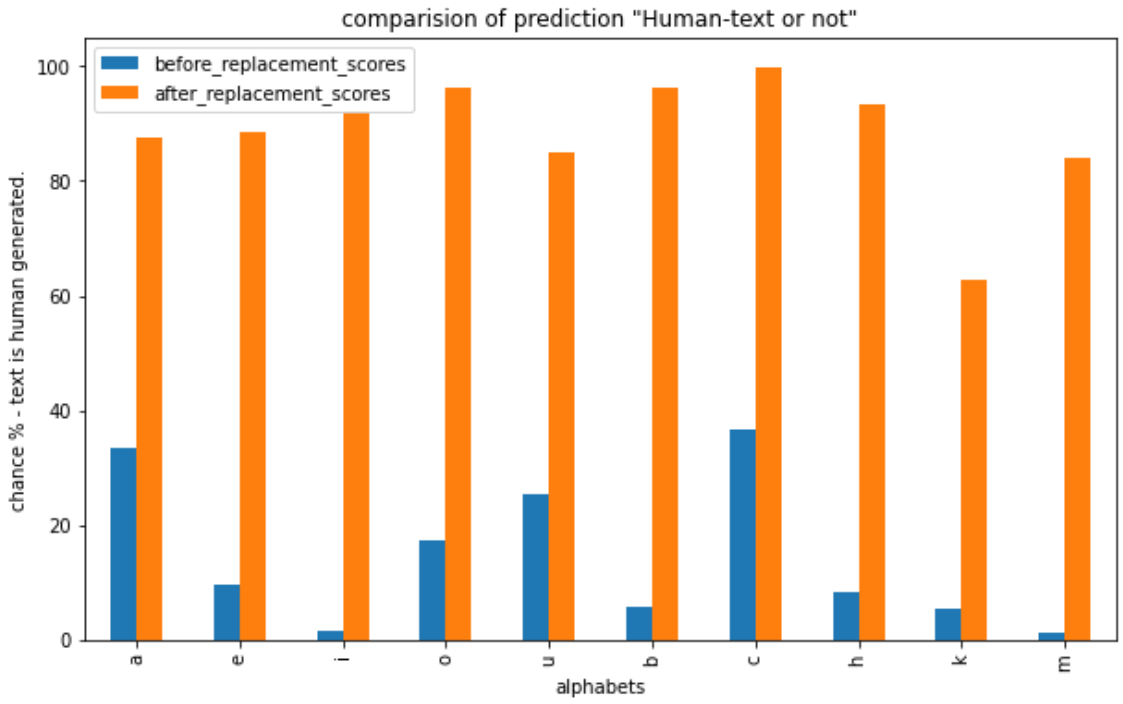
Methodology :

For each alphabet, we replace 1.5% of its occurrence with its homoglyph and then feed the text to the detector.

Following function does the same:

```
percentage = 0.015
alphabets_to_replace = ['a','e','i','o','u','b','c','h','k','m']

n_lines = 10
text_data = '/content/processed_data.jsonl'

before_replacement_scores = []
after_replacement_scores = []

for alphabet in alphabets_to_replace:
  real_score_before_replacement, real_score_after_replacement = experiment(percentage, alphabet, n_lines, text_data)

  before_replacement_scores.append(real_score_before_replacement)
  after_replacement_scores.append(real_score_after_replacement)
```

Observation :



We can see from the above graph that attack after replacement is highly successful in all cases. Interestingly, replacing vowels with homoglyphs was a much more effective attack, even when the frequency of replacement was the same as that of consonants.

Experiment 2

Objective:

The second homoglyph experiment took the most effective homoglyph pair **"a"** found in the first experiment and tested the effectiveness of the homoglyph attack when it was allowed to replace every occurrence of the target character(s).

Methodology:

We replaced 100% of the ‘a’ with its homoglyph and recorded the result.

Observation:

```
Real score before replacement:  17.454
Real score after replacement:  99.97500000000001
```

Experiment 3

Objective:

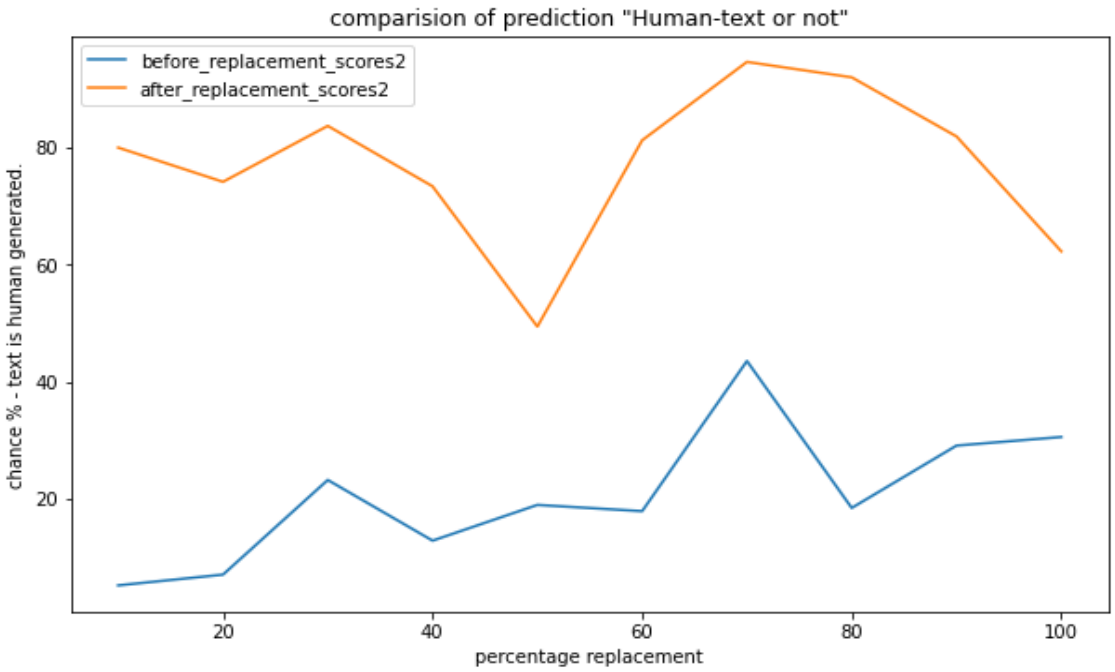
The third homoglyph experiment is designed to take the most effective homoglyph pair and test how varying frequencies of replacement may affect detector recall on neural text.

Methodology:

We increased the percentage of replacement from 10 % to 100% and noticed the change in accuracy. Letter chosen for the same is ‘a’.

Observation:

	percentage	before_replacement_scores2	after_replacement_scores2
0	10.0	5.211	79.950
1	20.0	7.047	74.109
2	30.0	23.207	83.660
3	40.0	12.847	73.340
4	50.0	18.950	49.389
5	60.0	17.883	81.198
6	70.0	43.519	94.582
7	80.0	18.432	91.948
8	90.0	29.078	81.858
9	100.0	30.545	62.237



Conclusion

Neural text detector accuracy on neural text was inversely proportional to the amount of characters a homoglyph attack was allowed to replace.

Experiment 4

Objective:

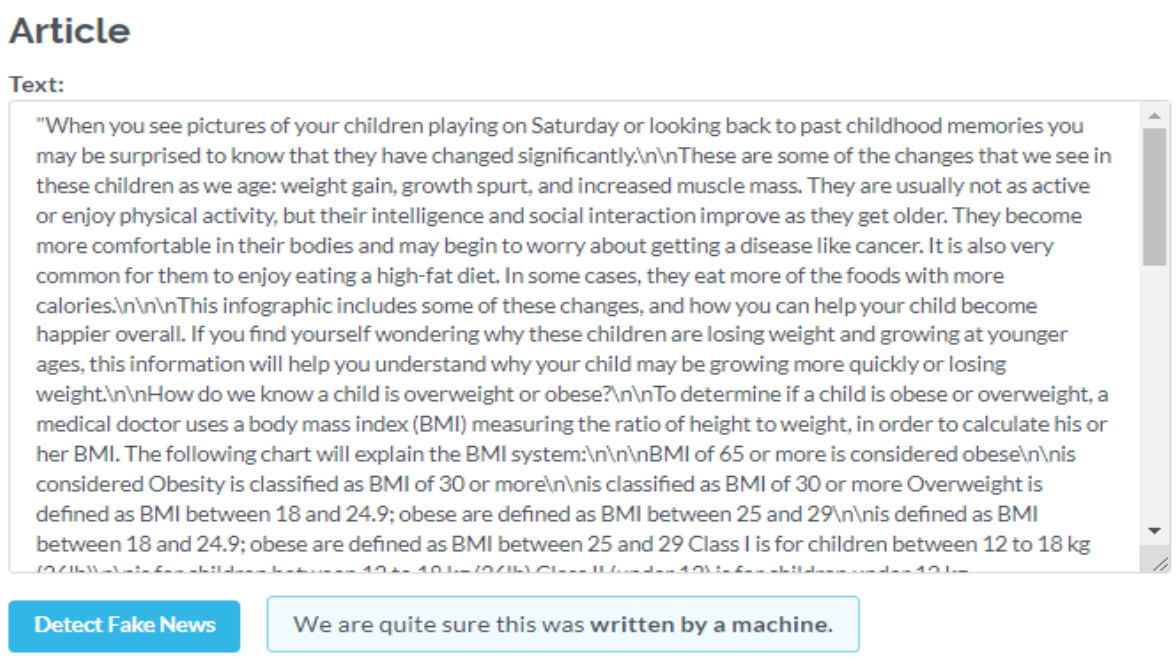
The fourth homoglyph experiment is designed to test the transferability of the homoglyph attacks to the **GROVER online demo**. website link : https://grover.allenai.org/detect

Methodology:

We took a text and checked for its machine generated probability. Then i replaced ‘a’ with its homoglyph 100% and again checked for its machine generated probability.

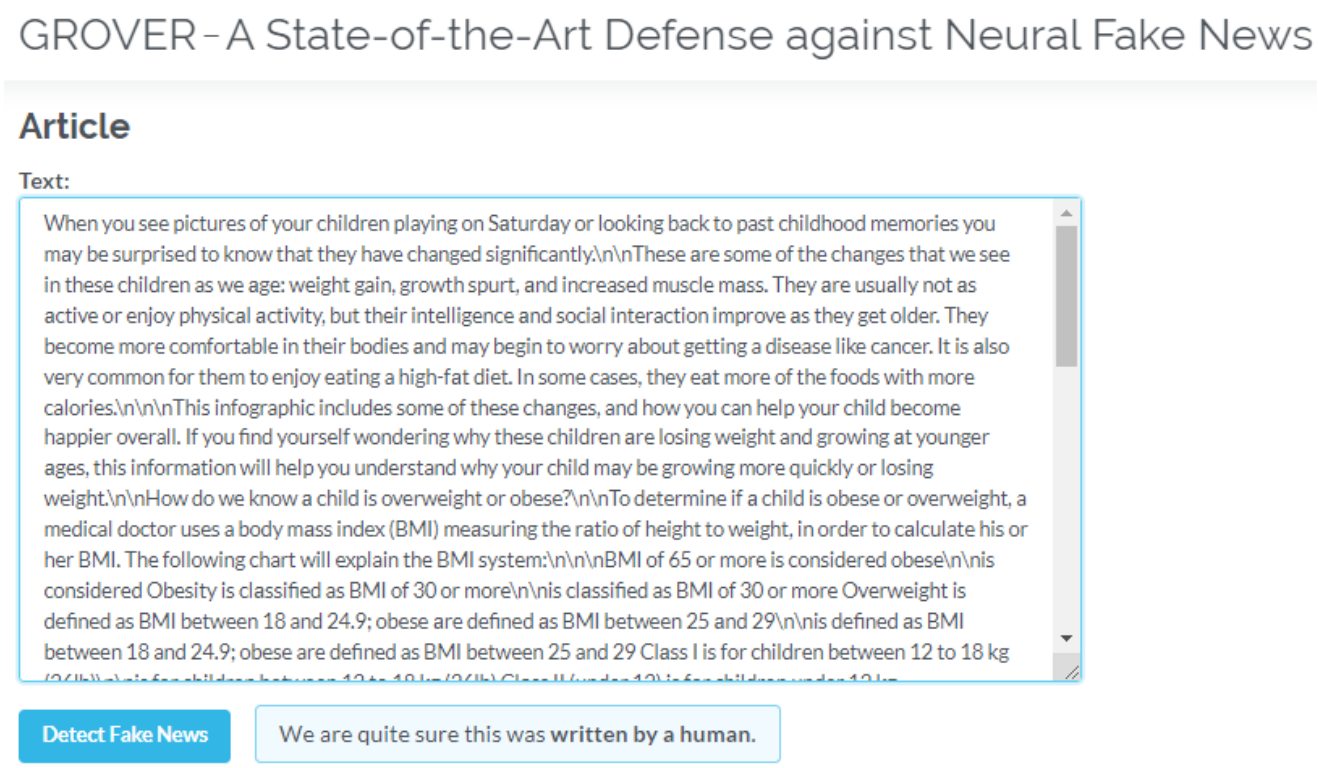
```
modified_text = replace_with_homoglyph(text, 1.0, 'a')
modified_text
```

Observation:



Without homoglyph attack, Grover online demo was quite sure that it was written by machine.

After the homoglyph attack, the following output was given which is quite opposite.



Conclusion:

The results of the fourth homoglyph experiment indicate that the homoglyph attacks are transferable to other neural text detectors. Before the English “a” to Cyrillic “a” attack was implemented, GROVER predicted Machine for 19 of the 20 samples, and predicted Human++ for 1 of the 20 samples. After the homoglyph attack, GROVER predicted Machine for 3 of the 20 samples, Machine+ for 1 of the 20 samples, Human+ for 1 of the 20 samples, and Human++ for the remaining 15 samples.

Experiment 5

Objective:

The fifth homoglyph experiment is designed to test the transferability of the homoglyph attacks to the GLTR online demo.

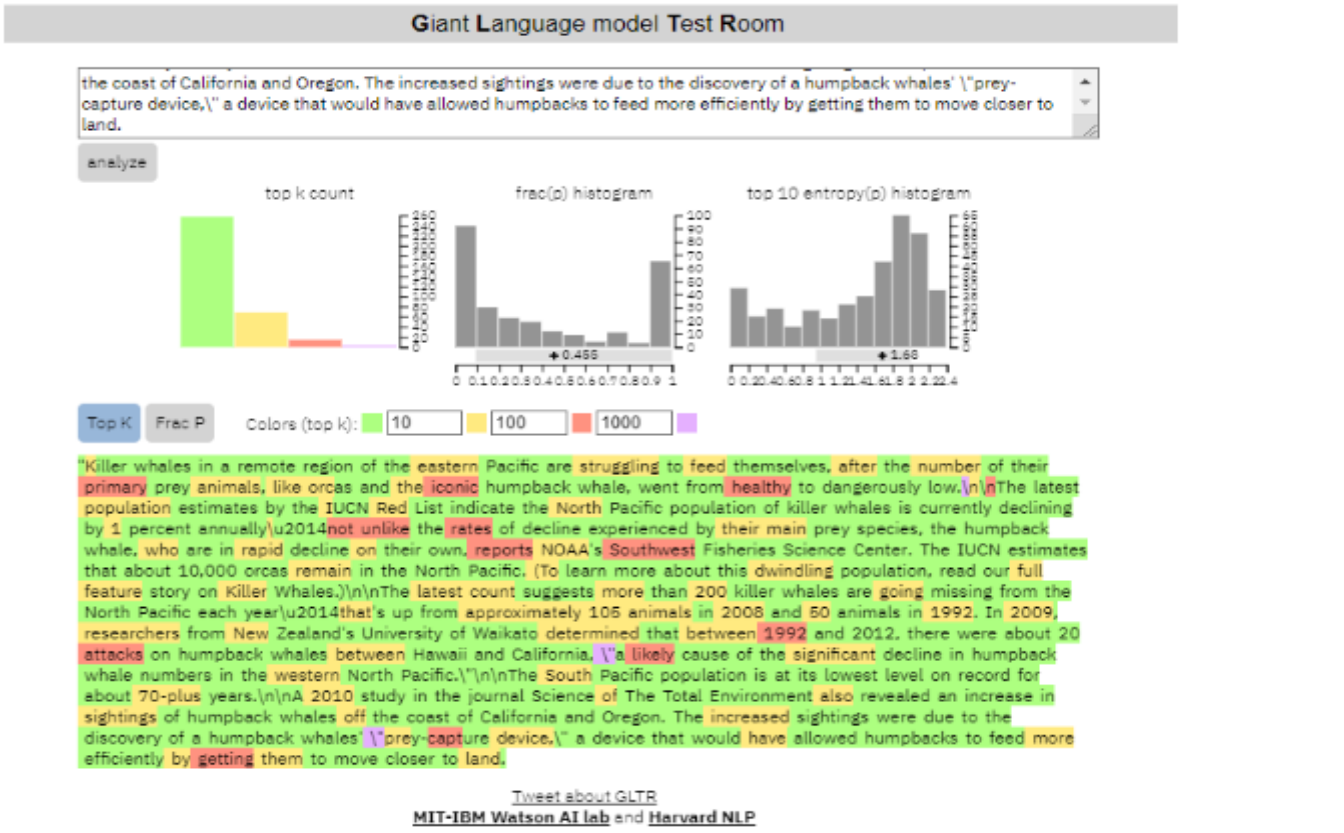
website link : http://gltr.io/dist/index.html

Methodology:

```
modified_text = replace_with_homoglyph(text, 1.0, 'a')
modified_text
```

Observation:

Before homoglyphic attack, increase in red and blue highlights denoting difficulty of recoginising the word written by machine.



After homoglyphic attack, increase in red and blue highlights denoting difficulty of recoginising the word written by machine.

Conclusion :

GLTR output before(fig 1) and after(fig 2) homoglyph attack. The presence of red and purple highlighted words indicates that GPT-2 117M had a difficult time predicting the word being highlighted, which helps human readers decide whether text was written by a language model or human.

Final Analysis:

It is interesting to note that the non-human like attacks were effective because they are not characteristic of human-written nor neural text, yet the neural text detectors predicted the text was human written–just because the modified neural text wasn't characteristic of neural text. Clearly, automatic neural text detectors are trained not to discriminate between neural text and human-written text, but rather decide what is characteristic and uncharacteristic of neural text. As seen by the success of the homoglyph attacks presented in this paper, this creates a vulnerability for neural text detectors in which an adversary can change neural text to be characteristic of neither language models nor humans (e.g. mixing English and Cyrillic alphabets), yet have the modified neural text be classified as human-written text.

REFERENCES USED:

- `Paper link –` https://arxiv.org/pdf/2002.11768v4.pdf
- Alternative python implementation with local detector (not API) - https://github.com/mwolff31/attacking_neural_text_detectors
- https://towardsdatascience.com/homoglyph-attack-prevention-with-ocr-a6741ee7c9cd