First report

Baseline:

My baseline consist of 3 steps:

1) Decide whether the sentence (or word) are toxic (e.g. build a toxicity classifier)
2) Generating substitutes of toxic sentences (or words) using model (probably pretrained)
3) Check if the substitution is toxic and if it fitting for the context

Evaluation metric:
The metric I want to use from the start is J metric. J metric firstly was proposed by Kalpesh Krishna, John Wieting and Mohit Iyyer in their paper. J metric is multiplication of three other metrics: **transfer accuracy** (how toxic a sentence is), **semantic similarity** (how sentences save the initial meaning) and **Fluency** (how sentences remain grammatically correct). You can find my realization of the J metric here. Big thanks to David Dale and his paper, I take a portion of the code from their realization, which you can find here.

Hypothesis 1:

My first hypothesis is: Build a toxicity classifier for words and then substitute the bad words with a star symbol (' * '). This approach known as "Profanity Filter" is used, for example, in video games or social platforms.

Toxicity classifiers:

1) Bag-of-words:

For the Toxicity classifier I firstly chose the most simple architecture: simple bag-of-words logistic regression classifier. I trained the model on a provided ParaNMT-detox corpus.

Here the results:
Accuracy = 0.5295718093391949 (A little better than random guess)
Mean_AUC = 0.551235928017817

Here the top 10 word importance:

	word	feature_importance
1779	Shit	7.267882
477	Dana	7.178648
5135	glued	7.144665
737	Friday	7.037260
8277	shithead	6.958261
2550	assassins	6.718610
7050	pattern	6.579329
3095	bum	6.534000
2835	bitching	6.522133
4983	frozen	6.514779

As you can see this classifier is not what I initially wanted to build. The 'most toxic' words are full of random words and accuracy is just bad. So I started to find another approach to build a logistic toxicity classifier. As told here the pretrained Word2Vec model will not work well, so I have not even tried to implement it. The next thing I use was NN with an embedding layer
The code (.ipynb) for the classifier you can find here

2) Neural network with embedding layer:

My model "ToxicWordClassifier" is a PyTorch-based model designed to classify toxic words in text. It utilizes an embedding layer to transform input words into continuous vectors, followed by a linear layer with a sigmoid activation for binary classification. Dropout is applied to improve model generalization and prevent overfitting.

```
self.embedding = nn.Embedding(vocab_size, embedding_dim)
self.fc = nn.Linear(embedding_dim, output_dim)
self.sigmoid = nn.Sigmoid()
self.dropout = nn.Dropout(dropout_rate)
```

For this NN I choose to use toxic and positive words inside the github from the paper. After training I get around 90% validation accuracy, that is definitely better than the previous classfier. I change every toxic word with a <Mask> and test some results on the main ParaNMT dataset. Here some examples:

```
Original Sentence:
I'm not gonna have a child... ...with the same genetic disorder as me who's gonna die. L...
Reconstructed Sentence:
i'[MASK] not gonna have a child..... with the same genetic [MASK] as me who'[MASK] gonna [MASK]. [MASK]...
```

```
Original Sentence:
They're all laughing at us, so we'll kick your ass.
Reconstructed Sentence:
they're all laughing at us, [MASK] we'[MASK] kick your [MASK].
```

Of course sometimes it is not the best and my classifier deciding that usual words are toxic:

```
Original Sentence:
Now you're getting nasty.
Reconstructed Sentence:
now [MASK]'re getting nasty.
```

```
Original Sentence:
I've got orders to put her down.
Reconstructed Sentence:
i've got [MASK] to put her [MASK].
```

My handling of toxic words is replacing all of them by * symbol. After that I take the J score and we can see that it is **0.5111,** and it is very high.

```
0.9271976281367242
| ACC | SIM | FL | J |

|0.7788|0.7077|0.9272|0.5111|
```

We can see that Similarity metric is the lowest point. It is mean that detoxified sentences are not very similar to initial sentences, but it is pretty good metrics for first Hypothesis. Next thing I want to test is synonyms generator from nltk library.

Hypothesis 2:

My second hypothesis is: Build a toxicity classifier for words and then use nltk.synonyms to find non-toxic synonyms. It should be definitely be better than replacing toxic words with star symbols, but we will se in the end.

NLTK synonyms:

For my first Hypothesis I use NLTK synonyms.lemmas() as my non-toxic word generator. I use the given ParaMT dataset. For preprocessing I delete all entries with similarity less than 0.8 and entries that do not lower the toxicity of reference inside the translation. I use values between 0.7 and 0.8 as my toxic threshold and get some results. Some of them are bad, some of them are good:

```
Original sentence: Ah! Monkey, you've got to snap out of it.
Non-toxic sentence: ah! imp, you've got to snap out of it.
```

```
Original sentence: Another one simply had no clue what to do, so whenever he met my brother he'd beat the crap out of him,
and then say:
Non-toxic sentence: another one simply had no clue what to bash, so whenever helium met my brother helium'd pulse the
bullshit out of him, and then state :
```

After evaluation with J metric I got **0.2611**, that is pretty bad, twice as bad as my first Hypothesis:

```
| ACC | SIM | FL | J |

|0.3444|0.8177|0.9272|0.2611|
```

As you can see, the lowest point in the metric is Transfer accuracy, which means that my detoxified sentences are still pretty toxic. We see that similarity and fluency are higher than before, so my sentences is more grammatically correct and similar to the initial ones, but they still very toxic.
The reason behind it actually the poor synonyms lists in nltk library. All synonyms to toxic words are usually also toxic. So results are even worse than before. So now I need new tool for detoxification and paraphrasing.

Hypothesis 3:

So if the main problem was poor nltk synonyms list, we could use LLM to find synonyms and substitute them

Results: