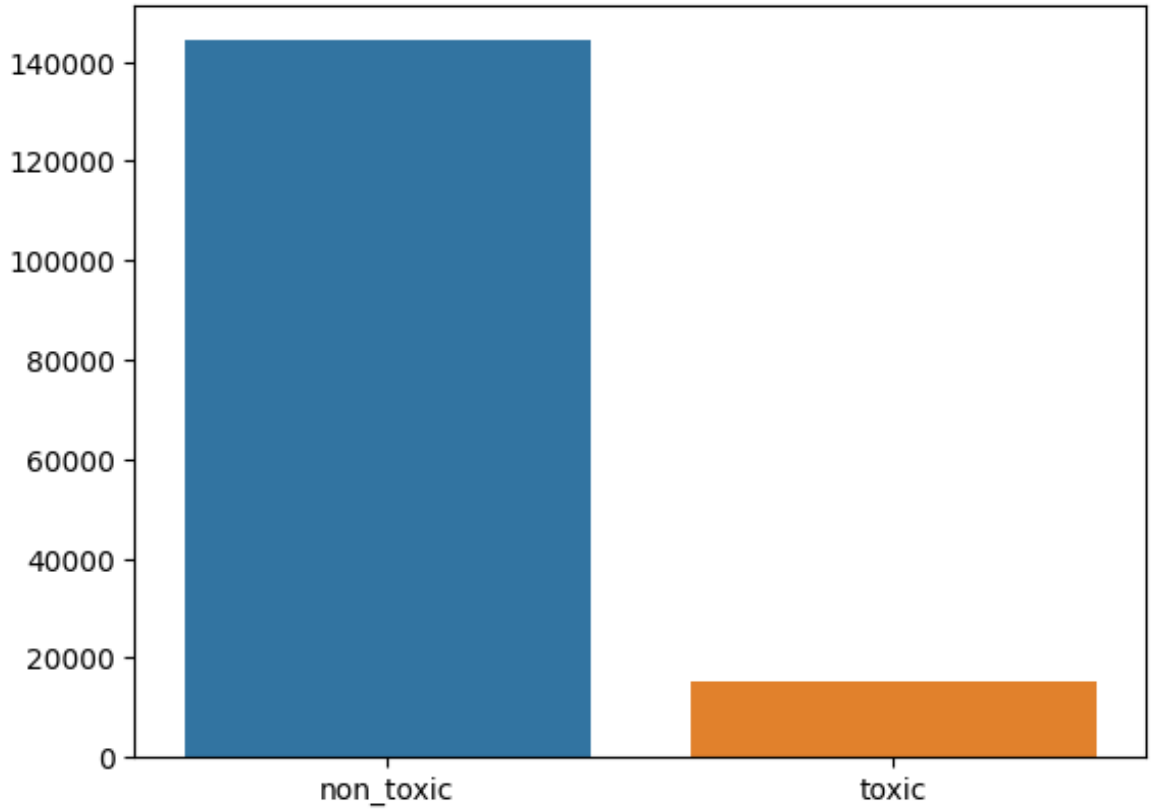
- Data collection and processing

As training data for our classification model, we get data from This kaggle competition. Given data contains a large number of Wikipedia comments which have been labelled by human raters for toxic behaviour. The dataset contains about 160 thousand comments. The types of toxicity are:

- Toxic
- Severe_toxic
- Obscene
- Threat
- Insult
- Identity_hate

We decided to simplify the task a bit and move to binary classification of the toxic comment to show you how it works.

First we left only the toxic and non-toxic columns, then we looked at the class distribution.



So the initial data is also unbalanced. This can affect the result of the training because we can divide the data into training and validation sets in an unfortunate way, so we got 5000 toxic and non-toxic comments for the training set and 500 for the validation set. Taking less data from the dataset will also help us in the future. For example, while researching the topic of adversarial attacks, we found that generating adversarial samples and making the initial model more robust is a computationally expensive task, so we simply wouldn't have enough resources for this task on the full dataset of toxic comments.

Since we used BERT, we need to Convert our comments into a format suitable for BERT. This includes splitting the text into tokens, adding special tokens [CLS] and [SEP], aligning the sequence lengths (we take 120). Also, we clear our comments using some constraints.

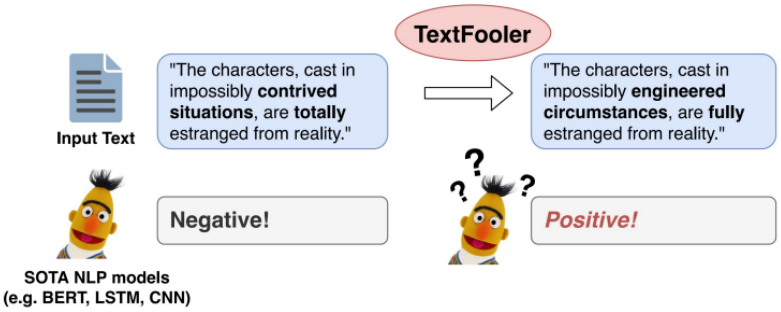
```
    string = re.sub(r"[^A-Za-z0-9(),!?\'\`\.]", " ", string)
    string = re.sub(r"\'s", " 's", string)
    string = re.sub(r"\'ve", " 've", string)
    string = re.sub(r"n\'t", " n't", string)
    string = re.sub(r"\'d", " 'd", string)
    string = re.sub(r",", " , ", string)
    string = re.sub(r"\.", " . ", string)
    string = re.sub(r"!", " ! ", string)
    string = re.sub(r"\(", " ( ", string)
    string = re.sub(r"\)", " ) ", string)
    string = re.sub(r"\?", " ? ", string)
    string = re.sub(r"\s{2,}", " ", string)
```

- Review of methods and models

BERT is a machine learning model based on transformers that helps machines understand and interpret the meaning of text. BERT uses bidirectional processing of text, taking into account the context both to the left and to the right of each token. BERT models are usually pre-trained on a large corpus of texts, and then fine-tuned for specific tasks. One of these tasks is text classification, which is our task, so we decided to use it. So we used a bert-base-cased pre-trained model from Hugging Face.

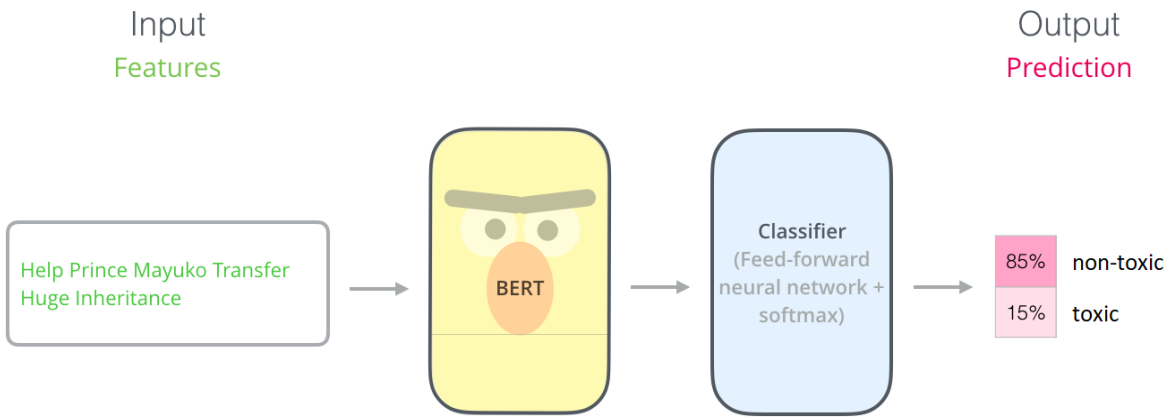
For adversarial attack we take ideas from this article. This attack method is called TextFooler. TextFooler is a system for generating natural adversarial text that can fool state-of-the-art natural language processing (NLP) models. TextFooler works by modifying a given text with minimal changes that preserve its semantic content and grammaticality, but make it misclassified or misinterpreted by the target NLP model. TextFooler is a simple but strong baseline for evaluating and improving the robustness of NLP models against adversarial attacks.

Goal	Constraints	Transformation	Search Method
Untargeted Classification	1. Word embedding distance 2. USE sentence similarity 3. POS consistency	Word substitution by counter-fitted GloVe embedding space	Greedy search with word importance ranking



- Architecture and implementation

We build your own model by combining BERT with a classifier. So, our model architecture will look like this.

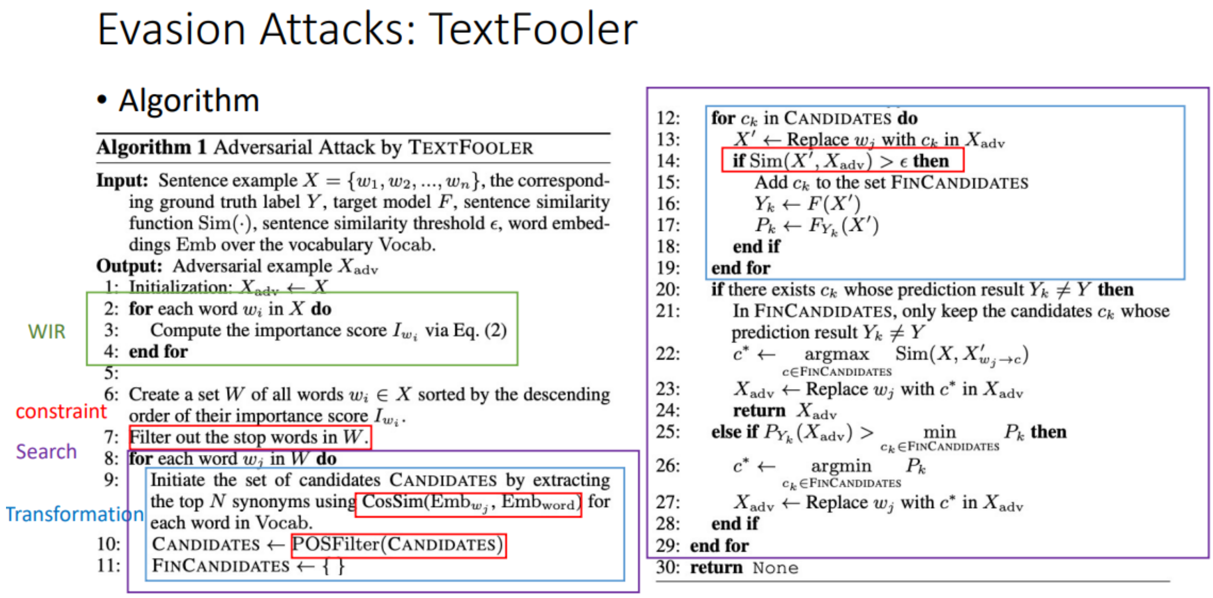


Here is implementation:

```
class BertClassifier(nn.Module):
    def __init__(self, bert: BertModel, num_classes: int):
        super().__init__()
        self.bert = bert
        self.classifier = nn.Linear(bert.config.hidden_size,
num_classes)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, input_ids, attention_mask=None):
        x = self.bert(input_ids, attention_mask=attention_mask)
        cls_x = x[1] # sentence embedding.
        cls_x = self.classifier(cls_x)
        out = self.softmax(cls_x)
        return out
```

TextFooler algorithm is:



Word Importance Ranking: we implemented it from scratch. We iteratively zero out each word and see what probability our model gives us. The lower the probability, the higher importance score of the word
Stop Words: Taken from nltk library
Synonym Extraction: To represent the words, we use word embeddings from this. These word vectors are specially curated for finding synonyms.
POS Checking: We used nltk pos tagger
Semantic Similarity Checking: We used the Universal Sentence Encoder to encode sentences.

- Evaluation and conclusion

To understand if our attack works or not we try to attack 5% of train and valid datasets.

Then make the pandas dataframe to visualise the results. Here is an example:

	Original Label	Attacked Label	Original	Attacked	Original target probability	Attacked target probability	Similarity of Sentences	Is Attack succeed?
5159	1	0	congratulations ! congratulations , jeraphine gryphon , you have won the asshole of the year award ! you have won this award by saying mean things about and to other people ! in order to claim your award , you must call 1 800 asshole (1 800 277 4653) ! once again , congratulations for your win ! 142 . 163 . 76 . 125 you won this for being rude to other contributers by the way . preceding unsigned comment added by 142 . 163 . 76 . 125	congratulations ! congratulations , jeraphine gryphon , you have won the dunce of the leto award ! you have won this award by saying mean things about and to other people ! in order to claim your award , you must call 1 800 asswipe (1 800 277 4653) ! once again , congratulations for your win ! 142 . 163 . 76 . 125 you won this for being rude to other contributers by the way . preceding unsigned comment added by 142 . 163 . 76 . 125	0.943993	0.341135	0.536970	Yes
100776	1	0	this is so stupid there are 9 planets damn it	this is so asinine there are 9 planets damn it	0.975057	0.016763	0.813821	Yes
87798	0	0	little late on the draw there , fag . 136 . 160 . 138 . 51	little late on the draw there , fag . 136 . 160 . 138 . 51	0.967012	0.967012	1.000000	Skipped
122400	0	0	i suggest you get some mental help .	i suggest you get some mental help .	0.956190	0.956190	1.000000	Skipped
114079	1	0	you again ? you 've acquired a reputation for being a right arsehole , aint'cha ? listen wanker , keep deleting my entries without explanation and you and i will have issues . go back to your hollow earth page and rustle a few feathers there if you feel the need to be a prick . do n't do it on my time . savvy ?	you again ? you 've obtains a reputation for being a right mongrel , aint'cha ? hear prick , maintaining deleting my entries without explanation and you and i will receive problem . went back to your hollow earth pages and rustled a small birds there if you feels the need to remain a bunghole . fait n't fait it on my time . shrewd ?	0.964479	0.488326	0.505571	Yes

Columns:

- Original Label - Label, which model predict
- Attacked Label - Label, after attacking
- Original - Original text
- Attacked - Attacked text
- Original target probability - Probability of Label, which model predict

- Attacked target probability - Probability of original Label, after attacking Similarity of Sentences - Cosine similarity, which show us semantic similarity of sentences
- Is Attack succeed? - Attack succeeded or not

Some statistic: 5% of Train dataset and 5% of Val dataset:

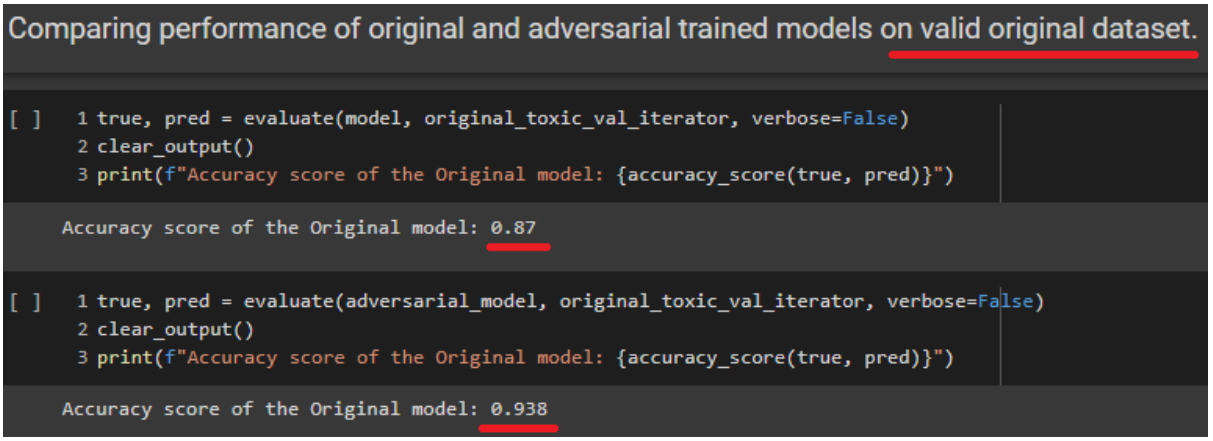
```
Yes        279
No         170
Skipped     51
Name: Is Attack succeed?, dtype: int64
```

```
No         22
Yes        21
Skipped     7
Name: Is Attack succeed?, dtype: int64
```

Where:

- Yes - Attack is success
- No - Attack is failed
- Skipped - Model predict toxic label as non-toxic

To train a robust model we add success attack samples to our original dataset and then we retrain the model. So, here are some statistic of non-robust (original) and robust (adversarial trained) model:



Conclusion: original model on valid dataset shows worse result in comparison with adversarial trained model. But we should keep in mind that the situation can be vice versa, when trained adversarial model shows worse results (its called trade-off between robustness and accuracy).

Comparing performance of original and adversarial trained models on valid dataset with adversarial examples.

```
1 from IPython.display import clear_output
2
3 true, pred = evaluate(model, prepared_toxic_val_iterator, verbose=False)
4 clear_output()
5 print(f"Accuracy score of the Original model: {accuracy_score(true, pred)}")
```

```
Accuracy score of the Original model: 0.8349328214971209
```

```
[ ]  1 true, pred = evaluate(adversarial_model, prepared_toxic_val_iterator, verbose=False)
     2 clear_output()
     3 print(f"Accuracy score of the Adversarial trained model: {accuracy_score(true, pred)}")
```

```
Accuracy score of the Adversarial trained model: 0.9385796545105566
```

Conclusion: model, which is not trained on adversarial examples give bad performance comparing with model trained on adversarial attack.

To evaluate robustness of two models we use this formula:

$$\text{attack success rate} = \frac{\text{\# of successful attacks}}{\text{\# of total attacks}}$$

Results:

Original model performance

```
[ ]  1 result_train["Is Attack succeed?"].value_counts()
```

```
Yes        279
No         170
Skipped     51
Name: Is Attack succeed?, dtype: int64
```

Adversarial trained model performance

```
[ ]  1 new_result_train["Is Attack succeed?"].value_counts()
```

```
No         244
Yes        231
Skipped     25
Name: Is Attack succeed?, dtype: int64
```

```
Original model attack success rate: 0.621380846325167
Adverasarial trained model attack success rate: 0.4863157894736842
```

Conclusion: As we can see, not only the Robustness has increased (attack success rate decreased), but also Accuracy (number of skipped values decreased 51 vs 25).

In the last block of the notebook we show you how to attack nlp models step by step without implementing algorithms from scratch. We used the textattack library to attack our model.

- Each team member contribution

Gia Trong Nguyen - Researching about adversarial attacks on nlp models, Implementing attack on the current model, Logging attacks, Evaluating the models, Report writing, and bringing beauty to our notebook.

Nikita Sergeev - Preparing data for training and evaluating processes, Training model for binary classification, Researching about adversarial attacks on nlp models, Report writing.

- Link to github

https://github.com/naryst/toxic_comments_classification