Team members:

1. Nikita Sergeev
2. Gia Trong Nguyen

Current progress:

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 labeled by human raters for toxic behavior. The types of toxicity are:

- `toxic`
- `severe_toxic`
- `obscene`
- `threat`
- `insult`
- `identity_hate`

  Dataset presented as `csv` file which looks this way:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0
5	00025465d4725e87	"\n\nCongratulations from me as well, use the ...	0	0	0	0	0	0
6	0002bcb3da6cb337	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0
7	00031b1e95af7921	Your vandalism to the Matt Shirvington article...	0	0	0	0	0	0
8	00037261f536c51d	Sorry if the word 'nonsense' was offensive to ...	0	0	0	0	0	0
9	00040093b2687caa	alignment on this subject and which are contra...	0	0	0	0	0	0

  We then divided this dataset into training and validation parts at a ratio of 1 to 20.

The next step we perform is to create a `pytorch Dataset` to continue using the `pytorch Dataloader` while training the model. To do this, we write a custom class that inherits from the base `pytorch Dataset` class.
In this class we define the next functions:

- `row_to_tensor` and
- and redefine the standard Python functions `__len__` and `__getitem__`.
  `row_to_tensor` function used to tokenise initial comment, get labels of comment and return tensor with all needed information. As a tokenizer we used standart `BertTokenizer`.

And after these steps, our data seems ready to train a model on it. As a classifier model we chose `bert-base-cased`. Firstly, we modify bert model for the given task with the following code:

```
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)

        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 = torch.sigmoid(cls_x)

        return out
```

And from this step we started to train model with the following hyperparameters:

- 1 epoch (should be enought for the current baseline solution)
- loss function - `Binary Cross Entropy`
- learning rate = 2×10^{-5}
- optimizer - `AdamW optimizer`
- scheduler - `linear scheduler with warmup`
  With the following parameters we got following results on the validation set in the end:

```
EVALUATING


ROC_AUC for labels:
 * toxic - 0.9859797022067768
 * severe_toxic - 0.9932098326934896
 * obscene - 0.9914925504716566
 * threat - 0.9823533107174268
 * insult - 0.9840761252835163
 * identity_hate - 0.9712344222780656

EVALUATE LOSS -  0.04146576672792435
```

  We think this is a pretty good result for the baseline model. And for now, this model is fully ready for further experiments. Source code for the training process can be found in our github repository at the following link.

Each team member contribution:

- Gia Trong - colab notebook with the implementation of the steps, described above
- Nikita - writing a report and organizing github repository for the project

What's next:

Over the next 3 weeks, we plan to integrate 'adversarial machine learning' into this project. We want to do this because we want to improve the robustness of the trained model against attacks, making it more reliable.
For now, we have the following plan of how to implement this idea:

1. Generate adversarial examples.

2. Train the model on adversarial examples.
3. Evaluate the model's robustness.
4. Fine-tune the model, using generated adversarial examples.
   However, this is not the final plan for the development of this project. During the research we may find other experiments that we find interesting and would like to implement and add to the model. They may not be that useful in terms of model performance, but they will be a good chance to practice our machine learning skills.

GitHub repository link