Intro to NLP: Assignment 2. Offensive Language Detection

Content warning: this assignment contains an analysis of offensive language examples.

In this assignment, we will work with the OLIDv1 dataset, which contains 13,240 annotated tweets for offensive language detection. The detailed description of the dataset collection and annotation procedures can be found here. This dataset was used in the SemEval 2019 shared task on offensive language detection (OffensEval 2019).

We will focus on **Subtask A** (identify whether a tweet is offensive or not). We preprocessed the dataset so that label '1' corresponds to offensive messages ('OFF' in the dataset description paper) and '0' to non-offensive messages ('NOT' in the dataset description paper).

The training and test partitions of the OLIDv1 dataset (olid-train.csv and olid-test.csv, respectively) can be found here.

You submit a **pdf** of this document, the format should not be changed.
Your analyses should be conducted using **python 3.8**.
You submit a **zip**-file containing all your code.

Each team member needs to be able to explain the details of the submission. By default, all team members will receive the same grade. If this seems unjust to you, provide an extra statement indicating the workload of each team member.

Total points: 20
Structure:

- Part A: Fine-tune BERT for offensive language detection (7 points)
- Part B: Error analysis with checklist (13 points)
- Bonus tasks: options for obtaining a grade > 8

Fill in your details below:
Group number:

Student 1
Name: Márton Fejér
Student id: 2694002

Student 2
Name: Gergő Pandurcsek
Student id: 2730356

Student 3
Name: Carol Rameder
Student id: 2747982

Part A: Fine-tune BERT for offensive language detection (7 points)

1. **Class distributions (1 point)**

Load the training set (olid-train.csv) and analyze the number of instances for each of the two classification labels.

Class label	Number of instances	Relative label frequency (%)	Example tweet with this label
0	8840	66.77%	'@USER @USER @USER She is a role model Adam you are not!'
1	4400	33.23%	'@USER She is fucking delusional'

2. **Baselines (1 point)**

Calculate two baselines and evaluate their performance on the test set (olid-test.csv):

- The first baseline is a random baseline that randomly assigns one of the 2 classification labels.
- The second baseline is a majority baseline that always assigns the majority class.

Calculate the results on the test set and fill them into the two tables below. Round the results to two decimals.

Random Baseline			
Class	Precision	Recall	F1
Offensive	0.26	0.46	0.33
Not offensive	0.7	0.49	0.58
macro-average	0.48	0.48	0.46
weighted average	0.58	0.48	0.51

Majority Baseline			
Class	Precision	Recall	F1
Offensive	0.00	0.00	0.00
Not offensive	0.72	1.00	0.84
macro-average	0.36	0.50	0.42
weighted average	0.52	0.72	0.60

3. **Classification by fine-tuning BERT (2.5 points)**

Run your notebook on colab, which has (limited) free access to GPUs.

You need to enable GPUs for the notebook:

- navigate to Edit → Notebook Settings
- select GPU from the Hardware Accelerator drop-down

- Install the simpletransformers library: *!pip install simpletransformers*
  (you will have to restart your runtime after the installation)
- Follow the documentation to load a pre-trained BERT model: ClassificationModel('bert', 'bert-base-cased')
- Fine-tune the model on the OLIDv1 training set and make predictions on the OLIDv1 test set (you can use the default hyperparameters). Do not forget to save your model, so that you do not need to fine-tune the model each time you make predictions.
  If you cannot fine-tune your own model, contact us to receive a checkpoint.

a. Provide the results in terms of precision, recall and F1-score on the test set and provide a confusion matrix **(2 points)**.

Fine-tuned BERT			
Class	Precision	Recall	F1
Offensive	0.732	0.6375	0.6815
Not offensive	0.8664	0.9097	0.8875
macro-average	0.7992	0.7736	0.7845
weighted average	0.8289	0.8337	0.83

Confusion Matrix: Fine-tuned BERT		
	Predicted Class	
Gold Class	OFF	NOT
OFF	153	87
NOT	56	564

b. Compare your results to the baselines and to the results described in the paper in 2–4 sentences **(0.5 points)**.

The predictions of the model are considerably more accurate than those of the random or majority baselines, but the model still struggles to identify offensive tweets more than not offensive tweets. Nonetheless, the results obtained here (macro F1-score: 0.7845) are in line with some of the better-performing participants in the paper (our score would have ranked between places 12-23, and perhaps with a bit of hyperparameter tuning, maybe even higher). Some of the top performers, including the winner of Sub-task A, also used the BERT. Indeed, the winner of the competition used default parameters and trained for just 2 epochs, but performed minimal feature engineering on the dataset, suggesting the usefulness of this approach.

4. **Inspect the tokenization of the OLIDv1 training set using the BERT's tokenizer (2.5 points)**

The tokenizer works with subwords. If a token is split into multiple subwords, this is indicated with a special symbol.

a. Calculate how many times a token is split into subwords (hint: use model.tokenizer.tokenize()). **(0.5 points)**

Number of tokens: 387931

Number of tokens that have been split into subwords: 67045

Example: if 'URL' is tokenized by BERT as 'U', '##RL', consider it as one token split into two subwords.

b. What is the average number of subwords per token? **(0.5 points)**

Average number of subwords per token (when all tokens are counted):

1.23 subwords/token

Average number of subwords per token (when only split tokens are counted):

2.36 subwords/token

c. Provide 3 examples of a subword split that is not meaningful from a linguistic perspective. **(1 point)**

Which split would you expect based on a morphological analysis?

Example 1: “@USER I'M SO FUCKING READY“

- **BERT tokenization:** ['@', 'US', '##ER', 'I', "'", 'M', 'S', '##O', 'F', '##UC', '##K', '##ING', 'R', '##EA', '##D', '##Y']
- **Morphologically expected split**: ['@', 'USER', 'I', ‘'M’, 'SO', 'FUCKING', 'READY']
- Because we are using the cased (instead of uncased) BERT model, all-capital-letter words are mistakenly split into what might be considered abbreviations or initials.

Example 2: “@USER Blow hard”

- **BERT tokenization:** ['@', 'US', '##ER', 'B', '##low', 'hard']
- **Morphologically expected split:** ['@', 'USER', 'Blow', 'hard']
- the word “Blow” should obviously not be split. This might also be a word play, in which case “blowhard” should be the expected split (although it also clearly consists of meaningful subwords)

Example 3: “@USER Principled conservatives are #Hypocrickets”

- **BERT tokenization**: ['@', 'US', '##ER', 'P', '##rin', '##ci', '##pled', 'conservative', '##s', 'are', '#', 'H', '##y', '##po', '##c', '##rick', '##ets']
- **Morphologically expected split:**

  ['@', 'USER', 'Principled', 'conservatives', 'are', '#', 'Hypocrickets']

  or

  ['@', 'USER', 'Principled', 'conservatives', 'are', '#', 'Hypo’, ‘##crickets']

- “hypocrickets” is a wordplay, so I would argue it could be a single named entity, but it could be argued that it’s composed of two subwords.

d. BERT's tokenizer uses a fixed vocabulary for tokenizing any input (model.tokenizer.vocab). How long (in characters) is the longest subword in the BERT's vocabulary? **(0.5 points)**

   Length of the longest token: 18
   Length of the longest subword: 14

   Example of a token with max. length: "telecommunications"
   Example of a subword with max. length: "sunderstanding"

Part B: Error analysis with checklist (13 points)

Often accuracy or other evaluation metrics on held-out test data do not reflect the actual model behavior. To get more insights into the model performance, we will employ three different diagnostic tests, as described in https://github.com/marcotcr/checklist.

Relevant literature:

- https://homes.cs.washington.edu/~marcotcr/acl20_checklist.pdf
- https://arxiv.org/pdf/2012.15606.pdf

Creating examples from existing datasets via perturbations (10.5 points)

Use a subset of the OLIDv1 test set, which contains 100 instances: (olid-subset-diagnostic-tests.csv, can be found in the same directory) and run the following tests:

5. **Typos (6 points)** Spelling variations are sometimes used adversarially to obfuscate and avoid detection (Vidgen et al., 2019; subsection 2.2), that is, users introduce typos to avoid their messages being detected by automated offensive language/hate speech detection systems. Let us examine how it influences our offensive language detection model.
   Use checklist to add spelling variations (typos) to the subset (olid-subset-diagnostic-tests.csv) and evaluate the model's performance on the perturbed data. Use a fixed random seed (`np.random.seed(42)`) to facilitate comparison.

Quantitative analysis:

- Describe the differences in performance compared to the non-perturbed data (precision, recall, F1-score macro). **(1 point)**

	Non-perturbed data	Perturbed - 2 typos	Perturbed - 4 typos
Precision	0.8293	0.84 ?!	0.8286
Recall	0.68	0.64	0.58
F1-Score	0.75	0.73	0.67
TP+TN	77	76	73

The metrics are reported only for the Offensive class (not the macro). The precision is largely unaffected, but the recall and F1-score decreases with increasing number of typos.

- How many messages were identified correctly in the original dataset, but erroneously after the perturbation? **(1 point)**

  The number of correctly classified tweets dropped from 77 to 73, after the perturbation with 4 typos. However, it is found that 6 previously correct predictions were now wrong, whereas 2 previously false predictions were now true.

Qualitative analysis:

- Check the add_typos function in checklist pertub.py. How were the typos introduced? **(1 point)**

  The typos were produced by swapping the position of a randomly chosen character from the initial string with its neighbour. There was a predefined number of typos added.

- Provide an example of a typo that cannot be produced by this function, but would play a role in offensive language detection. **(0.5 points)**

  Good and god - completely changes the meaning of the word Ugly and “U gli” - misspelling the word, even splitting it into two parts, so that the model does not recognize it as offensive and the user can easily understand the meaning

- Provide 3 examples when the model failed to assign the correct label after perturbation. **(1 point)**

Tweet - 5	Label = 1
Initial form	0-1 lost my acca on the first fucking fight cba
After perturbation	0-1 lost my acac on the firs tfucking ifgh tcba
Tweet - 13	Label = 1
Initial form	*gets all the bitches*
After perturbation	*gets lal hte ibtchse*
Tweet - 17	Label = 1
Initial form	@USER oh fuck off 😂😂
After perturbation	@USER h ofcuk fof 😂😂

- What is the main source of the erroneous predictions produced by the model (main source of errors caused by typos)? **(1 point)**

  The model leaned the correlation between some key offensive words found in the tweets and their labelling as offensive. After the noise was added with the perturbation function, that specific words were modified and, therefore, not recognized anymore as offensive.

- How can the model be improved? **(0.5 points)**

  Adding a new layer in the model or a completely new model that is trained to recognize typos could probably increase the efficiency of the model in our case. Also training on a larger dataset, which also includes tweets with typos, would probably make the model learn and workaround this issue.

6. **Negation (4.5 points)** Offensive language detection models have been shown to struggle with correctly classifying negated phrases such as “I don’t hate trans people” (Rottger et al., 2021; subsection 2.2).
   Add negations to the subset and evaluate the model's performance on the perturbed data. *Qualitative analysis:*

- Check the add_negation function in checklist pertub.py. What kind of negations does it produce? **(1 point)**

  It first checks the root token and what kind of word it is. If it's be then it adds not to the appropriate place. Afterwards it checks for auxiliary words and changes their form to the negated one. If none of them are present it looks at the word's tense and conjugates it into the proper negated form.

- Look at the created negated sentences, are they linguistically correct? Provide 2–5 examples of linguistically incorrect sentences. **(1 point)**

  - **89200** - @USER @USER Who the hell doesn't he think he is?
  - **51610** - #Canada - EXCLUSIVE: #Trudeau #Liberals leave $372M didn't mean to help veterans unspent since taking office URL
  - **16856** - @USER @USER I'll use that one the next time im in a gun control debate or in a debate about free speech or taxes. Yes you can't choose to be irresponsible or choose not to be. I argue responsible. Whats wrong with that? Don't justify murder by saying it was never alive or its my right.

- Check the first 10 negated messages. For which of these negated messages should the label be flipped, in your opinion? **(1 point)**

  Tweet ID: 60466, 16856, 71294

- Provide 2 examples when the model correctly assigned the opposite label after perturbation and 2 examples when the model failed to identify negation. Fill in the table below **(1 point)**

Examples correct	Tweet ID	Original label	Expected label after negation	Model prediction	Discussion: what is the potential reason for this behavior?
1	71294	1	0	0	
2	16856	1	0	0	
Examples wrong	Tweet ID	Original label	Expected label after negation	Model prediction	Discussion: what is the potential reason for this behavior?
1	22067	1	0	1	The hate speech part did not get negated
2	80947	1	1	1	The tweet still contains curse words

- How can the model be improved? **(0.5 points)**

In our opinion in this case to achieve better results it is not the model that should be improved but the negation function (sometimes it only partially negates or doesn't negate at all), and maybe preprocess some of the training data so it becomes less vague and more comprehensible. Apart from this, the general improvement of the model by including more (negated) examples always holds.

Creating examples from scratch with checklist (2.5 points)

7. **Creating negated examples**
   Let us further explore the impact of negations on our offensive language detection model.
   Consider the following templates: '*I hate …*' and '*I don't hate…*', and fill in the templates below:
   - Use masked language model suggestions: 'I hate {mask}' and 'I don't hate {mask}' .
   - Offensive language is often directed towards minority groups. Use the built-in lexicon and explore: 'I hate {nationality}', 'I don't hate {nationality}', 'I hate {religion}', 'I don't hate {religion}'

   Run the model on the created examples.
   - Provide 3 examples when the model assigns the correct label (correct label according to you) and 3 examples when the model fails to assign the correct label (choose both from masking and lexicon suggestions) **(1 point)**

     - I hate Trump 1
     - I don't hate you 0
     - I hate Christianity 1

   - Analyze the examples. Can you think of a reason why some examples are classified as offensive while others are not? **(1 point)**

     The model may assign a non offensive word as offensive because in some examples of the training data it was used as offensive

   - How can the model be improved? **(0.5 points)**

     Include more clear and simple examples in the training data.

BONUS:
Develop 2 new diagnostic tests (you can use checklist): describe what they test, explain why they are relevant and implement them. Run the tests and describe your observations. Provide examples of difficult cases, that is, when the model fails to assign the correct label. Discuss potential sources of errors and propose improvements to the model.