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: 83

Student 1
Name: Nathan Vaartjes
Student id: 2596846

Student 2
Name: Ellie Karanikola
Student id: 2724956

Student 3
Name: Valentin Buchner
Student id: 2647413

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
1	4400	33.23	@USER Liberals are all Kookoo !!!
0	8840	66.77	@USER Buy more icecream!!!

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
1	0.29	0.52	0.37
0	0.73	0.5	0.59
macro-average	0.51	0.51	0.48
weighted average	0.6	0.51	0.53

Majority Baseline			
Class	Precision	Recall	F1
1	0	0	0
0	0.72	1	0.84
macro-average	0.36	0.5	0.42
weighted average	0.52	0.72	0.6

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)**.

The model was initialized with 'weights' = [1, 2] to account for class imbalance and was run for 5 epochs. The parameter 'manual_seed' was set to 42 to ensure reproducibility.

Fine-tuned BERT			
Class	Precision	Recall	F1
0	0.873	0.889	0.881
1	0.699	0.667	0.682
macro-average	0.786	0.778	0.782
weighted average	0.824	0.827	0.825

Confusion Matrix: Fine-tuned BERT		
	Predicted Class	
Gold Class	0	1
0	551	69
1	80	160

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

The best model that resulted from the shared task achieved a macro-averaged F1 score of 0.829, which is surprising given that they used an uncased BERT model with default parameters. While our model uses cased BERT embeddings, it only achieves a macro-averaged F1 score of 0.782. It might be possible that the model performs worse because cased embeddings add complexity while not providing relevant information for the task of offensive language classification.

This makes sense, as the casing of a text should not have any influence on its offensiveness. For instance, the term "fuck you" is in theory just as offensive as "FUCK YOU". While uppercase letters make a post seem more aggressive, they can also occur in a non-offensive context, for example in the tweet "#TEAM4THEBEEZ PLEASE SHARE THIS PUP DID NOTHING EXCEPT HIS OWNER DIED AND HE WAS #DUMPED2DIE AT SHELTER HE IS IN GRIEF FOR HIS OWNER AND TERRIFIED IN NOISY SHELTER PLEASE #NOKILLSHELTERNATION IS WHAT WE NEED BUT LETS FIND THIS POOR PUP A #HEROES2ANIMALS URL".

Nevertheless, our model performs much better than the baseline models, indicating that embeddings indeed carry relevant information to classify offensive language.

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: 1.23

   In this subquestion, a token which is not split is considered to have 1 subword.

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?

   1. Example 1: FreedomOfExpression
   2. BERT tokenization: Freedom ##O ##f ##E ##x ##press ##ion
   3. Morphologically expected split: Freedom ##Of ##Ex ##press ##ion

   4. Example 2: HumanRights
   5. BERT tokenization: Human ##R ##ight ##s
   6. Morphologically expected split: Human ##Right ##s

   7. Example 3: USER
   8. BERT tokenization: US ##ER
   9. Morphologically expected split: it should not be splitted

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 subword: 16

   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.

   Note: Since adding only 1 typo only resulted in one post predicted differently, we added 3 typos to each post by setting the parameter 'typos' = 3.

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

original/with typos			
Class	Precision	Recall	F1
0	0.717/0.646	0.86/0.84	0.782/0.730
1	0.825/0.771	0.66/0.54	0.733/0.635
macro-average	0.771/0.709	0.76/0.69	0.758/0.683
weighted average	0.746/0.681	0.785/0.733	0.768/0.704

     - *It can be seen that on all metrics, the model performs between 2 and 12 percentage points worse on the perturbed data than on the non-perturbed data. It performs especially worse on class 1 (offensive tweets), indicating that it has more difficulties identifying offensive tweets when they contain spelling mistakes.*

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

     - *4 posts were correctly labeled as non-offensive on the original data, but incorrectly as offensive on the perturbed data. 8 posts were correctly labeled as offensive in the original data but as non-offensive on the perturbed data. Interestingly, there were also 6 posts which have been*

labeled incorrectly on the original data, but correctly on the perturbed data.

Qualitative analysis:

- Check the add_typos function in checklist pertub.py. How were the typos introduced? **(1 point)**
    - *The typos are introduced by swapping random characters with their neighbors. This is done by first selecting one or multiple (according to the parameter 'typos') random characters from the original text except the last character. Then, this character X is replaced with the following character and the following character is replaced with character X. The characters can also be whitespaces, which is why it sometimes adds the last character of a word to the beginning of the next word. Interestingly, this function would fail to add a type if it were to randomly select a character which is followed by the same character, as in 'loose'.*

- 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)**
    - *For instance if a post were: "He can go fck himself", this would not be covered by this function (Even though "fck" might have been learned by the model as a distinct offensive word). This is relevant as offensive words are often obfuscated on purpose. Further, obfuscations such as "son of a beech" could not be created, but are relevant to hate speech detection since they hide a swear word behind a non swear word.*

- Provide 3 examples when the model failed to assign the correct label after perturbation. **(1 point)**
    - *Example 1:*
        - *Original tweet: "@USER @USER Who the hell does he think he is?"*
            - *Correct label: 1*
            - *Assigned label: 1*
        - *Perturbed tweet: "@USER @USER Who the ehll doesh e htink he is?"*
            - *Assigned label: 0*
    - *Example 2:*
        - *Original tweet: "@USER @USER ...than why did you show us how hot she is ?"*
            - *Correct label: 0*
            - *Assigned label: 0*
        - *Perturbed tweet: "@USER @SUER ...thanw hy didy ou show us how hot she is ?"*
            - *Assigned label: 1*
    - *Example 3:*

        - *Original tweet: "#Conservatives are always revolting URL"*
            - *Correct label: 0*
            - *Assigned label: 0*
        - *Perturbed tweet: "#oCnesrvatives are always ervolting URL"*
            - *Assigned label: 1*

- What is the main source of the erroneous predictions produced by the model (main source of errors caused by typos)? **(1 point)**
    - *The models seems to not recognize offensive tweets as well when they contain typos. The model can probably not retrieve the correct words from the dictionary and does not use the correct word embeddings, making it unable to understand the text. As the model possibly learned that tweets which contain unknown words are more likely to be offensive, it is now labeling more tweets as offensive because they contain unknown words.*
- How can the model be improved? **(0.5 points)**
    - *A part of the training data could be perturbed by adding typos. This would make the model more robust to typos and would not lead to the model believing that tweets with unknown words are more likely to be offensive. Another possible approach could be to match unknown words to the most similar known word based on levenshtein distance. However, this could possibly result in incorrect matches and change the meaning of a post.*

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)**
        - *The function first filters out questions and already negated sentences first. Then, it looks for verbs and auxiliaries. If a lemma is a variation of the verb "to be", a "not" is placed after. If a lemma is an auxiliary verb, a "not" is also placed after it. In special cases, the auxiliary is replaced with its negated contraction (will - won't, can - can't) if such a pair exists.*
        - *It also changes clausal and nominal subjects to negative, by first extracting the subject's tense, adding do + not (n't) in the corresponding tense, and adding the infinitive . So for example, "What she said makes sense" would become "What she said doesn't make sense"*
        - *Importantly, it only adds a maximum of one negation per sentence.*
    - Look at the created negated sentences, are they linguistically correct? Provide 2–5 examples of linguistically incorrect sentences. **(1 point)**

- *Although the sentences are often grammatically correct, they don't always carry meaning. For example: "Yes you can choose to be irresponsible or choose not to be" becomes "Yes you can't choose to be irresponsible or choose not to be", which is a confusing sentence. And "Just wanted to tell you you should Hang Out With Me More" becomes "Just didn't want to tell you you should Hang Out With Me More", which again is technically correct, but does not mean much. Another example is "If we go by #Trump's example, where liberals support open borders, I don't guess conservatives support school shootings". Because the use of "i guess" is more of a language idiom, it does not make much sense to negate it.*
- *Some sentences, although this only happens rarely, are also incorrect grammatically. For example, the sentence "If you are wishing to loose fat, get back your wellbeing and slow down or reverse the aging process then check this super powerful natural method URL URL", becomes "If you are wishing to loose fat, don't get back your wellbeing and slow down or reverse the aging process then check this super powerful natural method URL URL". Because get back your wellbeing is part of the enumeration, it should have been "not get back your wellbeing" instead of "don't get back your wellbeing".*

- Check the first 10 negated messages. For which of these negated messages should the label be flipped, in your opinion? **(1 point)**
  - *In our opinion, none of the sentences really should change their label. Oftentimes, the offensiveness of a post comes from the terms used (ie profanity). Per the definition of offensiveness on the original dataset: "`In our annotation, we label a post as offensive (OFF) if it contains any form of non-acceptable language (profanity) or a targeted offense, which can be veiled or direct.`" As such, offensiveness that does stem from the use of offensive words, will still be offensive, even if negated. For the remaining offensive tweets where offensiveness stems from a targeted offense, we see that many tweets contain multiple sentences, and only inserting one negation still leaves other offensive materials intact. So, in general, adding negation very rarely produces the exact effect where the meaning of a sentence goes from offensive to non-offensive (or vice versa).*
  - *In the first 10, perhaps only "She is the most disingenuous person in the Senate." being turned to "She is not the most disingenuous person in the Senate." is an example where the label should go from 1 to 0, if we ignore the second part of the sentence ("If she was my Senator I would hide in shame after her despicable behavior in the SCOTUS hearings.").*

- 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 predic-tion	Discussion: what is the potential reason for this behavior?
1 no such examples (model changed labels only once, and it was incorrect)					
2					
Examples wrong	Tweet ID	Original label	Expected label after negation	Model predict ion	Discussion: what is the potential reason for this behavior?
1 "#Nigeria #Naija #9ja 'You are the most incompetent leader in Nigeria's history' – Atiku blasts Buhari URL By championnewsng Via URL" -> "#Nigeria #Naija #9ja 'You are not the most incompetent leader in Nigeria's history' – Atiku blasts Buhari URL By championnewsng Via URL"	**59751**	1	0	1	Saying someone is not incompetent shouldnt be offensive. However, the model marked it as such. Perhaps the word incompetent occurs very often in a sentence that is offensive, and the model failed to pick up on the subtlety of the negation changing the meaning of the sentence.
2 "@USER 'cause everyone knows a real leader falls into a van like a sack of potatoes....." -> 2 "@USER 'cause everyone knows a real leader doesn't fall into a van like a sack of potatoes....."	**79934**	1	0	1	As above, the negation changed the meaning of the sentence, but the model did not pick up on it.

- ***Comment:*** *There is only 1 tweet in the subset data of which the label changed. The predictions did not vary much with regards to the negation. Also, only 95 of the 100 sentences were perturbed. The remaining sentences could not be converted by the function, as they were questions or did not contain verbs.*

- How can the model be improved? **(0.5 points)**
    - *The model not picking up on the negation changing the meaning of the sentence, is an indication that it is perhaps more tuned to look at the presence of certain words than the general structure and meaning of the sentence. Ideally, the model would understand the modulatory effect of a negation: in essences, the meaning of a sentence is reversed. To achieve this, a model could be explicitly fed sentences that change labels by just introducing a negation, nothing else. The model would associate the presence of negation with a change in labeling.*

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)**
       - 3 examples when the correct label is assigned (marked correctly as offensive)
           - 'I hate Trump' , masking
           - 'I hate Christianity', lexicon - religion
           - 'I hate Chinese, lexicon - nationality
       - 3 examples when the wrong label is assigned (marked as offensive while they are not)
           - 'I don't hate Trump', masking,
           - 'I don't hate Zoroastrianism', religion
           - 'I don't hate Chinese', nationality

   - Analyze the examples. Can you think of a reason why some examples are classified as offensive while others are not? **(1 point)**
       - In the first 3 cases, the model assigned the correct label. This must be because, the words which are used in each expression, are often seen in corpora/tweets. In addition, the model is able to pick up the negative word

‘hate’ and also recognise the commonly used words, ‘Trump’, ‘Christianity’, ‘Chinese’
    - In the rest of the examples, the model fails to assign the correct label in cases where the words are more complex as ‘Zoroastrianism’ marking them as negative. It is also observed that ‘I don’t hate Trump’ and ‘I don’t hate Chinese’ are marked as offensive. This could be because the words, ‘Trump’ or ‘Chinese’ may occur in the corpus more often in an offensive context or because the model does not give weight to the negation which consists of ‘n’t ‘ and ‘hate’. Giving more weight to the negative meaning of ‘hate’.
- How can the model be improved? **(0.5 points)**
    - The model could be trained more on negations in order to recognise the non -offensive language used as a negation of negatively weighted words as ‘hate’. Or it could be trained on another dataset which would include a variety of complex words in order for the model to be able to pick them up as non offensive. Another thing would be that the dataset should be balanced in terms of the authors. For example, it could be that a number of authors write several tweets where ‘Trump’ or ‘Chinese’ could occur in an offensive context. The source should be controlled, however, this is a challenge.

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.