**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](https://github.com/idontflow/OLID), which contains 13,240 annotated tweets for offensive language detection. The detailed description of the dataset collection and annotation procedures can be found [here](https://aclanthology.org/N19-1144.pdf). This dataset was used in the SemEval 2019 shared task on offensive language detection ([OffensEval 2019](https://aclanthology.org/S19-2010.pdf)).

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](https://canvas.vu.nl/courses/59974/files/4963294?wrap=1).

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:** Leo Classen

**Student id:**

**Student 2**

**Name:** Mitchell Dior Lobbes

**Student id:**

**Student 3**

**Name:** Paola Feil

**Student id:** 2732911

**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 | 66.7674% | @USER @USER A must read! URL |
| 1 | 4400 | 33.2326% | @USER josh as slave leia |

1. **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 |
| 0 | 0.72 | 1.00 | 0.84 |
| 1 | 0.28 | 1.00 | 0.44 |
| macro-average | 0.49 | 0.49 | 0.47 |
| weighted average | 0.59 | 0.50 | 0.52 |

|  |  |  |  |
| --- | --- | --- | --- |
| Majority Baseline | | | |
| Class | Precision | Recall | F1 |
| 0 | 0.72 | 1.00 | 0.84 |
| 1 | 0.28 | 1.00 | 0.44 |
| macro-average | 0.36 | 0.50 | 0.42 |
| weighted average | 0.52 | 0.72 | 0.60 |

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

Run your notebook on [colab](https://colab.research.google.com), 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](https://simpletransformers.ai/): *!pip install simpletransformers*

(you will have to restart your runtime after the installation)

* Follow the [documentation](https://simpletransformers.ai/docs/usage/) 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.

1. 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 |
| 0 | 0.87 | 0.93 | 0.90 |
| 1 | 0.78 | 0.64 | 0.70 |
| macro-average | 0.82 | 0.78 | 0.80 |
| weighted average | 0.84 | 0.85 | 0.84 |

|  |  |  |
| --- | --- | --- |
| Confusion Matrix: Fine-tuned BERT | | |
|  | Predicted Class | |
| Gold Class | 0 | 1 |
| 0 | 577 | 43 |
| 1 | 87 | 153 |

1. Compare your results to the baselines and to the results described in the [paper](https://aclanthology.org/S19-2010.pdf) in 2–4 sentences **(0.5 points)**.

The f1 measures of the random and majority baselines scored in an interval between 0.40-0.60, while the scores of the BERT model resulted in between 0.70-0.90. Conclusively, the BERT model scored higher on all test measures in comparison to the baselines. Furthermore, the results of the confusion matrix are similar to those found in the paper, although the true positives and true negatives are slightly higher.

1. **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.

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

Number of tokens: 478955

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.

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

Average number of subwords per token: 0.33

1. 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:
2. BERT tokenization:
3. Morphologically expected split:

….

1. 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:

Example of a subword with max. length:

**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](https://canvas.vu.nl/courses/59974/files/4963294?wrap=1)) and run the following tests:

1. **Typos** **(6 points)** Spelling variations are sometimes used adversarially to obfuscate and avoid detection ([Vidgen et al., 2019](https://aclanthology.org/W19-3509.pdf); 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:*

***Note:*** *We added typos using the add\_typos function from checklist. We tested different amounts of typos (i.e. character swaps). Per sentence we introduced 10%, 20%, 30%, 40%, or 50% of typos. We used whole sentences rather than tokens as input to the add\_typos function, which means that also letters could we swapped with spaces. But we thought this is realistic in a setting where people type on a keyboard rather than writing on paper.*

* Describe the differences in performance compared to the non-perturbed data (precision, recall, F1-score macro). **(1 point)**
  + **Performance on original subset:**
    - * Macro-average precision: 0.83
      * Macro-average recall: 0.81
      * Macro-average F1: 0.81
  + **Performance on perturbed subset:**
    - Performance with 10% typos:
      * Macro-average precision: 0.77
      * Macro-average recall: 0.75
      * Macro-average F1: 0.74
    - Performance with 50% typos:
      * Macro-average precision: 0.24
      * Macro-average recall: 0.46
      * Macro-average F1: 0.32
  + **Description of differences:** Overall, performance on all specified metrics decreases when typos are introduced. There seems to be a linear relationship, so that the more typos are introduced the lower the performance. Noticeably, the macro-average precision has the highest score before perturbing and also with 10% typos, but when introducing a lot of typos (50%), macro-average recall becomes much higher than macro-average precision.
* How many messages were identified correctly in the original dataset, but erroneously after the perturbation? **(1 point)**
  + Identified correctly in non-perturbed dataset: 81
  + Identified correctly in perturbed dataset:
    - 90 (10% typos),
    - 75 (20% typos),
    - 70 (30% typos),
    - 69 (40% typos),
    - 59 (50% typos)

*Qualitative analysis:*

* Check the add\_typos function in checklist [pertub.py](https://github.com/marcotcr/checklist/blob/master/checklist/perturb.py). How were the typos introduced? **(1 point)**
  + The way the typos are introduced is as follows: The number of typos is given as a parameter which is used as a random selection for the amount of characters that are switched in the given string with their adjacent right character. The retrieved result is then returned as a newly concatenated string.
* 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)**
  + **Typos by adding letters:** “I haaate women” instead of “I hate women” could not be created with this function.
  + **Leaving letters out:** “Fckng Christians” instead of “fucking Christians” could not be created with this function.
* Provide 3 examples when the model failed to assign the correct label after perturbation. **(1 point)**
  + **Examples with 10% typos:**
    1. Example:
       - Original sentence:   
         *“(cr1tikal voice) smef my ass cheeks”*
       - Perturbed sentence:   
         “*(rc1tikal voice) semf my sas cheeks”*
       - Gold label: 1
       - Predicted: 0
    2. Example:
       - Original sentence*:   
         “@USER omg is he for real ?!?!!??????) this happened in peru like 40 years ago and the Inti devaluated so fucking much that they had to comoletely change the coin system because our money was worthless i-“*
       - Perturbed sentence:   
         *“@USER omg si he for real ?!?!!??????) htis happenedi n peru like 40 years ago andt he Int idevaulated s ofucking much that theyha d to comoletely cahnge the coi nssyte mbecause uor monyew as worhtless i-“*
       - Gold label: 1
       - Predicted: 0
    3. Example:
       - Original sentence:   
         *“#Conservatism101 It's not about our disagreements with #Conservatives. Its that Conservatives can't debate honestly, and they have no integrity. Whatever gets them thru today, is all that matters to them. They're fundamentally dishonest people. URL”*
       - Perturbed sentence:   
         “*#oCnsevrtaism101 It's not about ou rdisgareemnets with #Cnoservatives. Its that Conservtaives can't debate hoesntly, and they ahve no intgeriyt.W haetver ges tthem thru today, is all that matter sto them. They're fundamnetally disohnest peopel.U LR”*
       - Gold label: 1
       - Predicted: 0
* What is the main source of the erroneous predictions produced by the model (main source of errors caused by typos)? **(1 point)**
  + The main source seems to be typos in potentially offensive words (e.g. “ass” in example 1, “fucking” in example 2), but also negative words (e.g. “dishonest” in example 3). In example 3, it could also be related to the typos introduced twice in the word “Conservatives”, which could hint at a bias against this group.
* How can the model be improved? **(0.5 points)**
  + Train on more perturbated data to be able to deal with typos.

1. **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](https://arxiv.org/pdf/2012.15606.pdf); 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](https://github.com/marcotcr/checklist/blob/master/checklist/perturb.py). What kind of negations does it produce? **(1 point)**
  + The function negates verbs the following way:
    1. Is 🡪 is not
    2. Was 🡪 was not
    3. Were 🡪 were not
    4. Am 🡪 am not
    5. Are 🡪 are not
    6. Can / could 🡪 can't / couldn't
    7. Do / did 🡪 don't / didn't
    8. Will / Would 🡪 won't / wouldn't
    9. Have / had 🡪 haven't / hadn't
    10. ‘s / ‘re / ‘m 🡪 ‘s not / ‘re not / ‘m not
* Look at the created negated sentences, are they linguistically correct? Provide 2–5 examples of linguistically incorrect sentences. **(1 point)**
  + Who the hell doesn’t he think he is?
  + All these sick ass ppl from school didn’t give me something and now I have to chug down this nasty drink so it can go away🙃
  + He is obviously getting not suspended. He is not an asset for anyone.
* Check the first 10 negated messages. For which of these negated messages should the label be flipped, in your opinion? **(1 point)**
* 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 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| Examples wrong | Tweet ID | Original label | Expected label after negation | Model prediction | Discussion: what is the potential reason for this behavior? |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |

* How can the model be improved? **(0.5 points)**
  + Train on instances and their negation as well.

**Creating examples from scratch with checklist (2.5 points)**

1. **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)**
* Analyze the examples. Can you think of a reason why some examples are classified as offensive while others are not? **(1 point)**
* How can the model be improved? **(0.5 points)**

**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.