Assignment 2 & 3:

NLP Autumn 2023

Hateful/Abusive text detection.

Opening: Oct 17, 2023
Assignment 2(deadline): Nov 5, 2023
Assignment 3(deadline): Nov 15, 2023

Assignment 2: Hate speech classification using Word2Vec+NN, RNN, and Transformer(Bert)

In the first part of this assignment, you will learn how to classify comments as abusive or non-abusive using the Word2Vec & NN classifier.
In part 2 of the assignment, you will classify the same thing using RNN.
In part 3, you will build a text classifier using the Transformer-based language model BERT.

Dataset:
Link:
https://drive.google.com/drive/folders/1fHzeQamwZdbGhfBoVc7GTGiuQCW55bL7?usp=sharing

The dataset contains a collection of online social media comments in English. Based on their content, the comments have been labeled as either "hatespeech", "normal", or "offensive". Download the dataset(train, validation, test files) and train your models.

Data Fields:

- Comment Text: The text content of the comment.
- Comment Label: Label of the comment as either "hatespeech", "normal", or "offensive".

Task 1[50 Marks]: In this task, you will use the pre-trained Word2Vec model to generate your input feature vector, then do classification using the Neural Network.

Sample steps of the expected algorithm:

1. Preprocess the data, remove unnecessary symbols, stop words, etc.
2. Download the pre-trained Word2Vec model(link).
3. Use it for input feature generation from your actual raw text.
4. Design a Neural Network where the input layer dimension will equal your input feature dimension and the output layer will equal the number of target classes in the dataset.
5. Train the model with the training set and save the best model according to the validation set loss/performance, whichever suits you.
6. Report the test set performance on the final saved model.

Note: The choice of the number of hidden layers and dimensions is up to you. Play with them to get the best validation set performance.

Task 2[50 Marks]:

As you may have guessed, the above approach is fast but may not be optimal for text input. One of the main reasons is that in the above technique, you are not considering the word ordering in a sentence, which can be crucial for text classification.

So, you will use RNN to catch the ordering and dependencies among the words in a sentence and classify them.

Sample steps of the expected algorithm:

1. Preprocess the data, remove unnecessary symbols, stop words, etc.

2. Create the vocabulary of your model. (Fun exercise: You can change the size and check the performance vs efficiency trade-off.)
3. Define the maximum sequence length of your dataset.
4. Accordingly, tokenise and pad/truncate the sentences in your dataset.
5. Load train, validation & test set.
6. Define Dataloaders for batch processing while training.
7. Define your model class. It should include the definition of the embedding layer, RNN layer, classification layer, Dropout, and activation functions.
8. Train the model on the training data given.
7. Find validation set performance at the end of each epoch.
8. If validation set performance doesn't improve over k iterations (called patience), stop training(early stopping).
9. Load best model weights based on the validation set performance.
10. Find performance on the test set.

Task 3[50 Marks]:

You will use the transformer-based model BERT for text classification in this part.

The procedures are more or less the same as above.

Sample steps of the expected algorithm:

1. Preprocess the data, remove unnecessary symbols, stop words, etc.
2. Load the Model & Tokenizer from the huggingface library.
3. Design the dataloader class for tokenising and batch-processing the input data.
4. Create a model class; here, you can use the classification model from the huggingface library or the normal model without a classification head and add your own classification layer above it.

Also, the dropouts and activation functions should be defined here only.

5. Design the train loop and the optimizers, schedulers, and loss functions.
6. Load train, validation & test set.
7. Train the model on the training data given.
11. Find validation set performance at the end of each epoch.
12. If validation set performance doesn't improve over k iterations (called patience), stop training(early stopping).
13. Load best model weights based on the validation set performance.
14. Find performance on the test set.

Assignment 3: Hate speech classification using Flan-T5(Zero/Few-shot prompting & Finetuning)

In the first part of this assignment, you will learn how to classify comments as abusive or non-abusive using the Zero/Few-shot prompting with the Flan-T5(small & base) model.
In part 2 of the assignment, you will finetune the Flan-T5-base in Google Colab notebook using Peft & LoRA with limited resources.

Task 1[50+50 Marks]: In this task, you will make zero/few-shot inferences with the Flan-T5(small & base) model.

Sample steps of the expected algorithm:

1. Download the model from the hugging face.
2. For Zero-shot: Write a suitable prompt, and append the query test sentence and pass it to the model for inference. Process the output to get the target value.
3. For Few-shot: Similar to the Zero-shot setup, here you can provide the model with a few examples(input, output pair) for a better understanding of the task. Then, append the query test sentence and pass it to the model for inference. Process the output to get the target value.

Note: Play with different task descriptions for zero-shot settings for better test set performance.
Also, for the Few-shot case, play with the task description and the examples you are providing the model as a prompt.

Bonus (optional):

Task 2[50 Marks]: In the previous step, you have only exploited the model capacity in zero & few-shot settings in a pure inference-only setup, where you just download and use the off-the-shelf model. But we could do better by finetuning the model with our task-specific data, as in Assignment 2. But These Large language models(LLMs) are

notoriously large to load and finetune in a free Google Colab setup. But don't need to worry much as there are smart people around us who are constantly working to make our lives easier. As a result, we have Peft and LoRA/QLoRA, which will help us to load and finetune such LLMs in resource-constraint setup.
Don't worry; it may seem a bit difficult, but there are awesome tutorials available over the internet to learn and implement them easily.

Sample tutorial:
https://youtube.com/playlist?list=PLgy71-0-2-F30F5jSZnfFbqXVcclFf5zH&si=4e9H_PV1gRzy_itS
QLoRA: https://huggingface.co/blog/4bit-transformers-bitsandbytes

Sample steps of the expected algorithm:

1. Need to install a few packages to use Peft, LoRA, etc.
2. Format the dataset as query(Prompt+actual text input) and answer pairs.
3. Configure the Lora like the rank and keys.
4. The rest of the things are similar to Task 3 of Assignment 2.
5. Run the model. Save the best model till now after each epoch on the basis of validation set performance.
6. Test on the saved model.

Submission guidelines:

1. Name the Colab files as <roll_no><assignment_1/2>.ipynb

2. In CSE Moodle, you don't need to submit the whole code; just share the two Google Colab links with us through the Moodle submission. Make the link public so that it can be accessible. Private path links will lead to zero marks.

3. There should be a separate cell for each part, where you need to run and report the test accuracy and macro-f1 score clearly.

4. Also, add a function which will calculate the intersection(number of common sentences) between the test set with the train and the validation set. Run the function at the end and print the results. **Without this cell, we will not grade your submission.**

5. Marks will be relative as per the accuracy and macro-f1 score reported by the whole class. So, try to play with different hyper-parameters to get the most out of it.