ASSIGNMENT 1.2: EMAIL CLASSIFICATION

Motivation: Th motivation of this assignment is to get you some practice with text categorization using neural models. This assignment will proceed in two (maybe three) parts. Each part is worth equal points. This is part 2.

Problem Statement (same as A1.1): The goal of the assignment is to build a text categorization system for academic emails from annotated data. The input of the code will be a set of annotated emails sent to the Dean (Faculty) office at IIT Delhi from over five years ago.

Train and Dev Data splits (same as A1.1): There are 3 columns in the csv file - *Subject*, *Content* and *Class*. There must be 1622 rows in *train.csv* and 226 rows in *val.csv* excluding the header rows. There are a total of seven classes, namely - *Meeting and appointment*, *For circulation*, *Selection committee issues*, *Policy clarification/setting*, *Recruitment related*, *Assessment related* and *Other*. You may call these labels 1, 2, 3, 4, 5, 6 and 7 in the respective order they are mentioned in the last sentence. Note that the most frequent class is *Other* i.e. label 7. You are allowed to use the *Subject* field as well for training and inference.

The Task: You need to write a classifier that, given an email, predicts its class. In this Part, you must develop a neural classifier. Ideas include using CNN, RNN, BiLSTM, Transformer, or extensions. **But you cannot use (or cannot use code that uses) pre-trained language models like Bert, ElMo, Roberta, T5, GPT2, etc. Pre-trained word embeddings like Glove, Word2Vec, Fasttext are fine**. That is, all your models should initialize word vectors randomly or using pre-trained non-contextual embeddings, **but do not use pre-trained contextual models for word embeddings**. If you have any specific clarifications let us know on Piazza.

Importantly, you cannot use sparse models from A1.1. Of course, you can use insights and features that you developed, but you must convert the features into a dense space and then use neural prediction models. You will be allowed all choices in A1.3. Note: we expect A1.2 performance to be equal or better than A1.1 performance, but it will take some effort to get there if you are training neural models in NLP for the first time.

Methodology and Experiments:

Please use the same train and dev splits for training and validation respectively, as you did for A1.1. It's available on moodle now. This time, **it is compulsory to report the validation accuracy via a google form that we shall float**. You will need to report 2 scores - Score-1: Validation accuracy of the final model you are submitting, Score-2: Average validation accuracy over 5 training runs of the model (to account for possible randomness during training). We will run your submitted code and models to verify these numbers at our end (instructions mentioned in **Submission Format**).

Most importantly, as you work on improving your baseline system, document its performance. Perform (statistical) error analysis on a subset of data and think about why the model is making these mistakes and what additional knowledge could help the classifier the most. That will guide you in picking the next feature (or model component) to add.

Given that total amount of training data is small (which will often be the case in new applications), pay careful attention to overfitting. It will be easy to overfit and you will have to be creative to achieve good performance on the unseen test set. Some ways to reduce overfitting include: early stopping, dropout, batch normalization, layer normalization, reducing model capacity, adding auxiliary losses like L2 regularization or something more task specific (e.g., posterior regularization or adding constraints -- see (Nandwani et. al. NeurIPS 2019)).

Evaluation Metric (same as A1.1):

We will evaluate your output predictions using micro-accuracy and macro-accuracy scores (over all seven classes), same as done in A1.1. **We are releasing the python script *compute_accuracy.py* on Moodle**, which prints the micro and macro accuracy, when given as arguments the test predictions file and the test gold file paths. For consistency, you must also use this script for reporting validation accuracy scores via the google form.

Command to run evaluation script - *python compute_accuracy.py <path-to-test.csv> <path-to-outputfile.txt>*

Test Format (same as A1.1):

Your final program will take as input a set of emails in the same csv format as training, but without the 3rd column named *Class*. So, you need to write a test script which runs on a csv file having 2 columns only - *Subject & Content.* Your test script will output a file containing predicted labels (1,2,3,4,5,6 or 7) one in each new line – matching one prediction per email.

Submission Format:

The deadline for submission is **15th December 2020, 11:55 PM.**

Submit your code in a .zip file named in the format **<EntryNo>.zip** on Moodle. It must contain all the files required to replicate the training and testing of your final model.

We will be evaluating your submissions using HPC GPU nodes (K40 instances - 11.4 GB RAM). We will upload the submitted zip file to HPC and run the following commands:

```
unzip <EntryNo>.zip
cd <EntryNo>
# Create a new virtual environment
conda create -n <EntryNo> python=3.7
# Install all the required libraries
bash install_requirements.sh
# Install gdown utility to download files from google drive
pip install gdown
# Download your trained model from your google drive
gdown https://drive.google.com/uc?id=xxxxxxxxxxxx
# Test the downloaded model on the provided test file
bash run_model.sh test <test_file_path> outputfile.txt
# Compute accuracy
python compute_accuracy.py <test_file_path> outputfile.txt
# Prints: Micro Accuracy: xx%, Macro Accuracy: xx%

# We will randomly train the model from scratch to check that you achieve the same results
# Train the model
bash run_model.sh train <train_file_path>
# Test the trained model
bash run_model.sh test <test_file_path> outputfile.txt
```

You should upload your model to **Google Drive**, change the access settings to ‘Anyone with the link’ and paste the shareable link in a file google_drive_link.txt. We will use the gdown utility to download the model (https://github.com/wkentaro/gdown) as shown in the above commands. We will manually check the timestamp when the model has been uploaded. The submission time of your assignment will be considered as max(Moodle timestamp, Google Drive timestamp). *Please be careful while uploading your final model. Double-check to ensure that you have uploaded the correct model.*

Regarding re-training the model from scratch, we understand that perfect reproducibility is difficult with deep learning libraries like PyTorch and we will not penalize for small deviations in results that may be expected due to randomness among different runs. However, if the deviations between the re-trained model and the uploaded model are significant, then there may be a significant penalty. You can have a look at https://pytorch.org/docs/stable/notes/randomness.html to understand the steps needed to reduce randomness in your trained models.

All the above commands will be run in an automated scripting fashion. Any error(s) will lead to a loss of 20% of the total credit. So make sure you conform to these requirements.

The *outputfile.txt* should only contain a sequence of numbers (ranging from 1 to 7)**,** one per line, with a total number of lines matching the number of emails in *test.csv*.

The writeup.txt should have a first line that mentions names of all students you discussed/collaborated with (see guidelines on collaboration vs. cheating on the course home page). If you never discussed the assignment with anyone else say None.
After this first line you are welcome to write something about your code, though this is not necessary.

Runtime instructions:

Your inference code must run on a HPC K40 GPU (11.4 GB RAM) instance within one hour.

Your training code must run on a HPC K40 GPU (11.4 GB RAM) instance within six hours.

If your inference or training time exceeds the above limits, you will face a significant penalty.

Evaluation Criteria

(1) This part is worth 100 points.
(2) Bonus given to outstanding performers.

What is allowed? What is not?

1. The assignment is to be done individually.
2. **You must use only python version 3.7 and PyTorch library for this assignment.**
3. You must not discuss this assignment with anyone outside the class. **Make sure you mention the names in your write-up in case you discuss with anyone from within the class outside your team.** Please read academic integrity guidelines on the course home page and follow them carefully.
4. Feel free to search the Web for papers or other websites describing how to build a text categorization system. However, you should not use (or read) other people's NLP classifier codes.
5. You are allowed to use any pre-existing ML softwares for your code (preprocessing, evaluation, k-fold etc.). Popular examples include Python Scikit (http://scikit-learn.org/stable/) and Gensim (https://pypi.org/project/gensim/). You may use pre-trained neural models or off-the-shelf tools for tasks such as POS tagging, NER etc. However, they should not be built on top of pre-trained

language models. if you include the other code, use a different directory and don’t mix your code with pre-existing code. And you must not use existing text-categorization softwares.

6. We will run plagiarism detection software. Any team found guilty will be awarded a suitable penalty as per IIT rules.
7. Your code will be automatically evaluated. You get a significant penalty if it does not conform to given guidelines.