Natural Language Processing (CS60075)

Assignment 3: News Topic Classification

Deadline: 10 April, 2024 23:59 hrs

Guidelines:

- ➔ This assignment is to be done individually by each student. You should not copy any code from one another, or from any web source. We will use standard plagiarism detection tools on the submissions. Plagiarized codes will be penalized heavily.
- ➔ This assignment is meant to use Python only, specifically using the Pytorch library. You are also allowed to use other libraries like numpy, transformers, scikit-learn which are needed for efficient implementation of models.
- ➔ For training and testing your codes, you may use Google Colab **[Click here]**.
- ➔ Solutions should be uploaded via the CSE Moodle website (see course website for details). Submit one .zip file containing a folder with your codes and the pdf report. Name the compressed file the same as your roll number. Example: "*24CS60R00.zip*"
- ➔ Try writing code that is organized into different functions and add explanatory comments. Make sure your code can be easily read. Codes that cannot be understood will be penalized.

Problem:

You are given a collection of news articles. Your task is to classify the news texts into 4 topics/classes – World, Sports, Business, and Science/Technology. You will first use the Word2Vec and RNN model for ***Task1*** and then improve it by using BERT model in ***Task2***.

Dataset Description:

- The data folder '*NLP3*' contains two *.csv* files. You can download it from here -> NLP3.zip
- This folder contains a train and a test dataset, each consisting of multiple sentences (*Column1*) and their respective labels (*Column2*). The labels are -
  - World (0)
  - Sports (1)
  - Business (2)
  - Sci/Tech (3)
- The train dataset contains 2000 sentences, while the test dataset contains 500 sentences. The train dataset is equally distributed, i.e., it has 25% samples from each of the four classes.
- Consider 10% randomly selected samples from the training set as a **validation set**.

Task1: *[30 + 40 = 70 marks]*

In this task, (A) use the Word2Vec model to generate your input feature vector. Next, (B) do classification using two types of RNN models to catch the ordering and dependencies among the words in a sentence and classify them.

Steps to be followed:

A) Word2Vec Model:

Data Preprocess:

- Preprocess the data – lowercasing, removing unnecessary symbols, stop words, etc. Replace all '\\' with <space>. Mention the steps you have taken in the report.
- Create the vocabulary out of the sentences for your model. Mention your choices such as Min and Max document frequency in the report.

Word2Vec Model:

- Download the Word2Vec model from Python library *gensim.models* **[Link to Docs]**.
- Use it for input feature generation from your actual raw text – get the embeddings for each token in the text.

NN Layers:

- 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.
- Pass the average of the word embeddings of a sentence to the NN layer.

Training:

- Train the model with the training set for a fixed number of epochs and save the best model according to the validation set loss. Describe your choice of number of epochs in the report.

Testing and Evaluating:

- Find performance of the (best) trained model on the test set.
- Report the test accuracy and macro-f1 score **[click here]**.
- Plot the confusion matrix and classification score table **[click here]**.
- **Store the predictions on the test set in a standard CSV file called "w2v_test.csv".** This should be the same as test.csv, but with an additional *"preds"* column.

B) Recurrent Neural Networks (RNN):

Data Preprocess:

- Use the already preprocessed data from *Task1(A).*

Data Batching:

- Define a maximum sequence length for your dataset.

- Accordingly, tokenize and pad/truncate the sentences in your dataset.
- You can't use *batch_size* = 1 for his task. Describe your choice of Batch size and Sequence Length in the report.
- Define Dataloaders for batch processing while training and testing.

RNN Model:

- Define your model class. It should include the definition of the embedding layer, RNN layer, classification layer, Dropout, and activation functions.
- Initialize the embedding layer with Word2Vec embeddings as described above.
- Use two types of RNN from Pytorch - Vanilla RNN **[click here]** and LSTM **[click here]** (thus you will be defining two classes). Use bidirectional variants in both case.
- Pass the average of outputs of RNN/LSTM for all the tokens to the classification layer to get the final labels.

Training:

- Design the train loop, optimizers, and loss functions.
- Train the model on the training data given for a fixed number of epochs, and save the model with the best validation loss. Describe your choice of number of epochs in the report.

Testing and Evaluating:

- Find the performance of each of the two trained models on the test set.
- Report the test accuracy and macro-f1 score **[click here]**.
- Plot the confusion matrix **[click here]**.
- **Store the predictions on the test set in a standard CSV file called "rnn_test.csv" and "lstm_test.csv".** This should be the same as test.csv, but with an additional *"preds"* column.

Deliverables:

- A python file/ipython notebook "rnn.py"/"rnn.ipynb"
- Create a report "rnn_report.pdf" to record all metrics and output scores and tables.
  - In the report, mention if you faced any challenges faced during the implementation, and how you resolved it.
  - Report the hyperparameters you have used, e.g. batch size, learning rate, num epochs, max sequence length, vocab size…
  - Report if you used any assumptions in the code. Also explain if you have done anything extra to improve the performance of your model.
- The set of predictions on the test data "w2v_test.csv", "rnn_test.csv", "lstm_test.csv"

Task2: *[30 marks]*

In this task, you will use the transformer-based model BERT for the text classification.

Steps to be followed:

Data Preprocess:

- Use the already preprocessed data from *Task1,* but do **NOT** remove stop words.

Data Batching:

- Define the maximum sequence length of your dataset (cannot be more than 512 RoBERTa tokens).
- Accordingly, tokenize and pad/truncate the sentences in your dataset.
- You can't use *batch_size* = 1 for his task. Describe your choice of Batch size and sequence length in the report.
- Define Dataloaders for batch processing while training and testing.

BERT Model:

- Create a model class – you can use the sequence classification model (tranformer model with classification head) from the huggingface library; or you may load just the transformers model and add your own classification layer above it.
- For input feature generation from your actual raw text, load the Model & Tokenizer for the "RoBERTa-base" checkpoint from the huggingface library **[link to docs]**.

Training:

- Design the train loop, optimizers, and loss functions.
- Train the model on the training data given for a fixed number of epochs, and save the model with best validation loss. Describe your choice of number of epochs in the report.

Testing and Evaluating:

- Find performance of the trained, saved model on the test set.
- Report the test accuracy and macro-f1 score **[click here]**.
- Plot the confusion matrix **[click here]**.
- **Store the predictions on the test set in a CSV file called "bert_test.csv".** This should be the same as test.csv, but with an additional *"preds"* column.

Deliverables:

- A python file/ipython notebook "bert.py"/"bert.ipynb"
- Create a report "bert_report.pdf" to record all metrics and output scores and tables.
  - In the report, mention if you faced any challenges faced during the implementation, and how you resolved it.
  - Report the hyperparameters you have used, e.g. batch size, learning rate, num epochs, max sequence length, vocab size…
  - Report if you used any assumptions in the code. Also explain if you have done anything extra to improve the performance of your model.
- The set of predictions on test data "bert_test.csv"