Assignment 6 - Exercise 1 - Text Classification with Transformers

Contributor: Dimitris Vougioukos - f3352411

The exercise 1 focuses on building a BERT classifier for sentiment analysis on an existing dataset and comparing its performance with baseline classifiers.

The following chapters will outline the steps taken and the design decisions made to achieve the desired results.

Exercise 1

At the beginning of the exercise, we chose to download and use the `movie_reviews` dataset from the `NLTK` library, which contains various movie reviews. We examined its structure and identified two file types, `neg` (negative review) and `pos` (positive review), representing the distinct classes under which the reviews are categorized. Specifically, we observed that the dataset contains 1.000 reviews categorized under the `neg` class and another 1.000 reviews under the `pos` class. This defines a binary classification problem.

After that, we extracted the raw text of each review along with its corresponding class/label and stored them into two separate lists for further processing. We used the `train_test_split` function from the `sklearn` library to divide the dataset into three subsets: training, validation, and test sets, with the following sizes.

Subset	Size - Number of Reviews
Training	1120
Validation	480
Test	400

Next, we transformed the datasets into the Hugging Face dataset format to ensure compatibility with the training process of the BERT models.

To build and train BERT models, we implemented some functions for each step of training, evaluation, and testing, as described below.

Function - tokenize_reviews

This function is used to tokenize a dataset of reviews using a BERT tokenizer.

The function follows these steps:

- Takes a dataset of reviews as input.
- Uses the provided BERT tokenizer to tokenize the text of the reviews.
- Applies padding to ensure all tokenized sequences have a uniform length.
- Truncates reviews that exceed the specified maximum length.
- Returns the tokenized reviews.

Function - configure_lora

This function is used to set a `LoRA` configuration for fine-tuning BERT models.

The function follows these steps:

- Accepts parameters for `LoRA` rank, scaling factor and dropout rate.
- Defines a `LoraConfig` object with the specified parameters.
- Targets the query and value modules for adaptation.
- Specifies the task type as sequence classification.
- Returns the configured `LoraConfig` object.

Function - train_bert_model

This function is used to train a BERT model for binary classification.

The function follows these steps:

- Defines training arguments, including learning rate, batch size, weight decay etc.
- Configures automatic model selection by tracking the validation loss and loading the best model at the end of training.
- Initializes a `Trainer` object with the specified BERT model, training dataset, validation dataset, and early stopping.
- Trains the model using the defined `Trainer`.
- Returns the trained `Trainer` object.

Function - collect_losses

This function is used to collect the training and validation losses from a `Trainer` object.

The function follows these steps:

- Initializes two lists to store training and validation losses.
- Iterates over the training logs.
- Extracts and appends the training loss.
- Extracts and appends the validation loss.
- Returns the lists of training and validation losses.

Function - compute_classification_scores

This function is used to compute the classification metrics of a BERT model, including precision, recall, f1-score, and precision-recall AUC.

The function follows these steps:

- Computes precision, recall, and f1-score for `neg` (negative) and `pos` (positive) classes and the macro-averaged values of them.
- Computes precision-recall AUC for `neg` (negative) and `pos` (positive) classes.
- Computes macro-averaged precision-recall AUC.
- Returns the classification scores.

After defining all necessary functions, we trained and fine-tuned three different pre-trained BERT models.

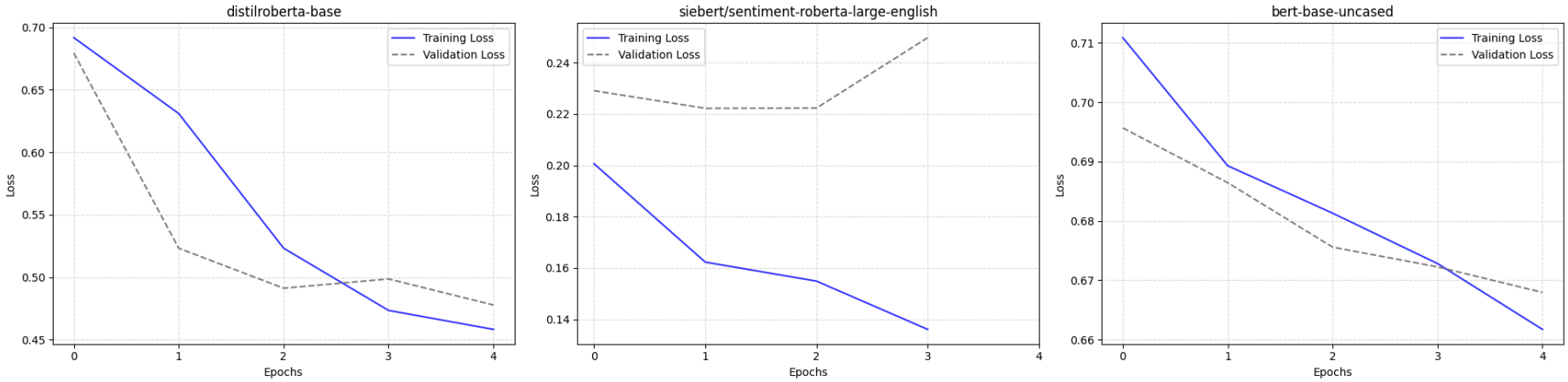
The pre-trained models:

1. `distilroberta-base`
2. `siebert/sentiment-roberta-large-english`
3. `bert-base-uncased`

The training process follows these steps:

- Defines a list of BERT models.
- Iterates over each model, loading the corresponding tokenizer and tokenizing the training and validation datasets.
- Converts the tokenized datasets into `PyTorch` tensors for efficient processing.
- Loads the pre-trained BERT model.
- Applies `LoRA` to the model to optimize training efficiency.
- Trains the adapted BERT model using the tokenized datasets.
- Collects training and validation losses.
- Stores relevant information, including losses, the trained model, and its tokenizer.

After collecting all the loss values, we visualized the corresponding curves, as shown below.



As shown in the plots above, the best-performing BERT model is the `siebert/sentiment-roberta-large-english` achieving the lowest validation loss, approximately 0.222176 at epoch 2. Therefore, we selected it for comparison with the baseline classifiers.

Using the best BERT model, we tokenized the reviews in the training, validation, and test datasets and converted them into PyTorch tensors for efficient processing. We then generated predictions, applied the softmax function to obtain probabilities, extract predicted labels, and compute classification scores to evaluate the model's performance.

Classification Scores across all Classifiers

Training Dataset

Class	Metric	Best BERT Model	CNN Classifier	RNN Classifier	MLP Classifier	k-NN Classifier	Majority Classifier
neg	Precision	0.9615	1.0	1.0	0.9251	0.8237	0.0
	Recall	0.9375	1.0	1.0	0.8821	0.8429	0.0
	F1-Score	0.9494	1.0	1.0	0.9031	0.8332	0.0
	Precision-Recall AUC Score	0.9904	1.0	1.0	0.9658	0.9091	0.75
pos	Precision	0.939	1.0	1.0	0.8874	0.8391	0.5
	Recall	0.9625	1.0	1.0	0.9286	0.8196	1.0
	F1-Score	0.9506	1.0	1.0	0.9075	0.8293	0.6667
	Precision-Recall AUC Score	0.99	1.0	1.0	0.9698	0.9108	0.75
macro avg	Precision	0.9503	1.0	1.0	0.9062	0.8314	0.25
	Recall	0.95	1.0	1.0	0.9054	0.8313	0.5
	F1-Score	0.95	1.0	1.0	0.9053	0.8312	0.3333
	Precision-Recall AUC Score	0.9902	1.0	1.0	0.9678	0.91	0.75

Validation Dataset

Class	Metric	Best BERT Model	CNN Classifier	RNN Classifier	MLP Classifier	k-NN Classifier	Majority Classifier
neg	Precision	0.9469	0.7897	0.8692	0.8435	0.7094	0.0
	Recall	0.8917	0.8292	0.8583	0.8083	0.7833	0.0
	F1-Score	0.9185	0.8089	0.8637	0.8255	0.7446	0.0
	Precision-Recall AUC Score	0.9805	0.8826	0.9304	0.9209	0.8141	0.75
pos	Precision	0.8976	0.8202	0.8601	0.816	0.7581	0.5
	Recall	0.95	0.7792	0.8708	0.85	0.6792	1.0
	F1-Score	0.9231	0.7991	0.8654	0.8327	0.7165	0.6667
	Precision-Recall AUC Score	0.9681	0.8906	0.9093	0.9041	0.7891	0.75
macro avg	Precision	0.9223	0.8049	0.8646	0.8297	0.7338	0.25
	Recall	0.9208	0.8042	0.8646	0.8292	0.7312	0.5
	F1-Score	0.9208	0.804	0.8646	0.8291	0.7305	0.3333
	Precision-Recall AUC Score	0.9743	0.8866	0.9198	0.9125	0.8016	0.75

Test Dataset

Class	Metric	Best BERT Model	CNN Classifier	RNN Classifier	MLP Classifier	k-NN Classifier	Majority Classifier
neg	Precision	0.9485	0.8041	0.8511	0.8514	0.7062	0.0
	Recall	0.92	0.78	0.8	0.745	0.745	0.0
	F1-Score	0.934	0.7919	0.8247	0.7947	0.7251	0.0
	Precision-Recall AUC Score	0.9803	0.875	0.9143	0.9011	0.7804	0.75
pos	Precision	0.9223	0.7864	0.8113	0.7733	0.7302	0.5
	Recall	0.95	0.81	0.86	0.87	0.69	1.0
	F1-Score	0.936	0.798	0.835	0.8188	0.7095	0.6667
	Precision-Recall AUC Score	0.9775	0.8668	0.9135	0.8849	0.7638	0.75
macro avg	Precision	0.9354	0.7953	0.8312	0.8124	0.7182	0.25
	Recall	0.935	0.795	0.83	0.8075	0.7175	0.5
	F1-Score	0.935	0.795	0.8298	0.8067	0.7173	0.3333
	Precision-Recall AUC Score	0.9789	0.8709	0.9139	0.893	0.7721	0.75

From the classification scores across all datasets, we reached the following conclusions:

- The best BERT model (`siebert/sentiment-roberta-large-english`) is the top-performing classifier, consistently achieving the highest Precision, Recall, F1-Score, and Precision-Recall AUC across all datasets. Its performance indicates that fine-tuning a pre-trained transformer model is highly effective for the given text classification task.
- The RNN classifier performs well, especially in the training dataset, achieving perfect scores in some metrics. However, in the validation and test datasets, its performance drops slightly, suggesting a risk of overfitting.
- The CNN classifier shows strong performance in the training dataset but underperforms compared to the RNN and MLP classifiers in the validation and test datasets. This suggests that it may be overfitting, making it less generalizable to unseen data.
- The MLP classifier provides competitive results, performing better than the CNN but worse than the RNN classifier on the validation and test datasets in some metrics.
- The k-NN classifier performs poorly across all datasets, with significantly lower scores in Precision, Recall, F1-Score, and AUC. This indicates that it is not well-suited for this text classification problem, likely due to its inability to effectively capture contextual relationships.
- The Majority classifier is the weakest, as expected, since it simply predicts the majority class, leading to poor overall metrics.