# Assignment 4 in Artificial Intelligence II 2022-2023

**Dimitris Orinos ic121006-7115132100006**

This assignment refers to the creation of a sentiment classifier for the dataset imdb-reviews.csv, which is the same dataset that is used in the three previous assignment of this course, by fine tuning the pretrained BERT-model which could be found on the Hugging Face[1].

## 1st part :BERT Implemenation on movie sentiment analysis

BERT stands for Bidirectional Encoder Representations from Transformers. It is state of the art NLP technique for a variety of applications such as Name Entity Recognition, Text classification, Question and Answering and many more. BERT was developed by Google Research team and made it public in October 2018 [1,2]. The Bert model has twelve encoder and twelve decoder layers where each of BERT's 12 encoder layers, takes in the enhanced embeddings from the previous layer, process them by passing these vectors into a 'self-attention' layer i.e a layer that helps the encoder look at other words in the input content (i.e sentence) as it encodes a specific word [1,3], then into a feed-forward neural network, and further enhances them by sends out the output upwards to the next encoder. The decoders have the self attention and an feed forward layer and additionally they have between these two layers is an attention layer that helps the decoder focus on relevant parts of the input [1,2,3,4].

The two basic steps in BERT framework is the pre-training on a large amount of unlabeled data set, which is done based on Masked Language Model (MLM) and Next Sentence Prediction (NSP) [1,4] mode and fine-tuning, where all of the parameters, firstly initialized with with the pre-trained parameters, are fine-tuned using labeled data [1].

The BERT has two different sizes: BERT-base and BERT-large. Performance wise BERT-large is more accurate as it has more bert-layer(24) and embedding size = 1024 but it is very hard and time taking to fine tune on other hand BERT-base comes with 12 bert-layer and 768 embedding size which is much more easy to train as compare to BERT-large and performance wise also BERT-base gives reasonably good performance [2].

The Hugging Face library is downloaded through !pip install transformers. In terms of quickness and needs for a bigger and stronger RAM, colab Pro is downloaded and for

all the experiments the GPU with the 15 gb CPU RAM is used and through !nvdia-smi. Through the function set_seed(), where we set the seed as seed=1234 and the results are the same (or almost similar) every time this program is run.

The preprocessing of our dataset means that the text must be emptied from duplicate words, numbers, urls, whitespaces, single characters from start or from end etc. All these are removed from every text review of the dataset of this exercise by using nltk library or other python libraries such as re, collections etc. About punctuation and stopwords their removal is done through BERT tokenization as it described in the next paragraph.

Tokenization is a very basic part of the Bert and for this reason the BERT tokenizer BertTokenizerFast.frompretrained() is loaded[5,6,7,8,9] which is the Bert Tokenizer in a faster version. Later through the dataset of csv file is splitted through train_test_split method from sklearn, and with random_state = 1234, the results are the same (or almost similar) every time this program is run [10].

A basic step is the creation of a custom class BertReviewsDataset about our dataset which is child class of Dataset provide and this class contains blue print for the data which will be feed to our BERT model [2]. About the inputs of the Bert Model, the model expects two inputs, token ids, which is an integer that represents a particular token and attention mask, which is a sequence of ones and zeroes to tell the model which token comes from input sentence (segment id =1) and which are just padding token(segment id =0).

Also padding is a very important factor for our classification. More specifically when BERT model is trained we make sure that every input to the model should have same size that means same length of inputs so that the model can perform back propagation efficiently but all our input which are review text can not be in same size, some can be small review and some can be large. Padding is the technique where we make our entire review in the same size, so it is important to decide a fixed length or maximum length [2]. Hugging face library provides another function called tokenizer.encode_plus() which we will use to perform almost entire preprocessing steps in one go. It:

- converts reviews into tokens
- adds [CLS] token at the beginning of input
- performs padding if sequence length is less than max_len
- performs truncation if sequence length is greater than max_len

- adds [SEP] token at the end of sequence.

The train dataset which is the 80% and the validation dataset which is the 20% of our dataset through pytorch DataLoader, where it combines a dataset and a sampler through shuffle = True, and provides an iterable over the given dataset, will be converted into batches[2,11,12]. Also about the optimization part the AdamW optimizer from pytorch is used implements Adam algorithm with weight decay fix [13].

About the learning rate the hugging face function get_linear_schedule_with_warmup is used, where a schedule is created with a learning rate that decreases linearly from the initial lr set in the optimizer to 0, after a warmup period during which it increases linearly from 0 to the initial learning rate set in the optimizer[14]. In this exercise in this scheduler we have 0 warmup steps.

Also in the training iteration the vector of reviews and reviews ratings a for every batch requires grad. It is important to refer that in the iteration of the training set the optimization consists of the steps below [15]:

- Call optimizer.zero_grad() to reset the gradients of model parameters. Gradients by default add up; to prevent double-counting, we explicitly zero them at each iteration.
- Backpropagate the prediction loss with a call to loss.backward(). PyTorch deposits the gradients of the loss w.r.t. each parameter.
- Once we have our gradients, we call optimizer.step() to adjust the parameters by the gradients collected in the backward pass.

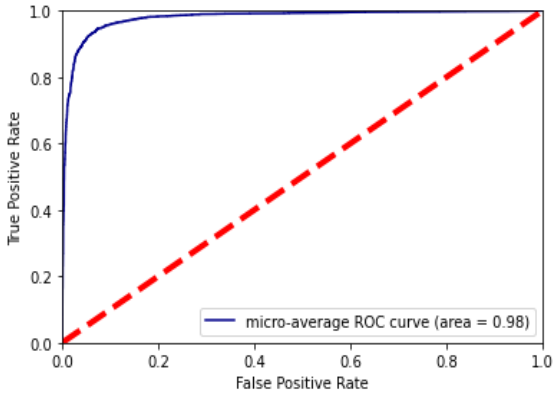
The next step is the creation of the BERT classifier. For this purpose a neural network is created which contains the bert model with name ‘bert-base-uncased’, which is loaded from the BertModel.from_pretrained(‘bert-base-uncased’) [5,6,7,9,16], three linear hidden layers and one dropout layer with probability 0.05 [17,18].

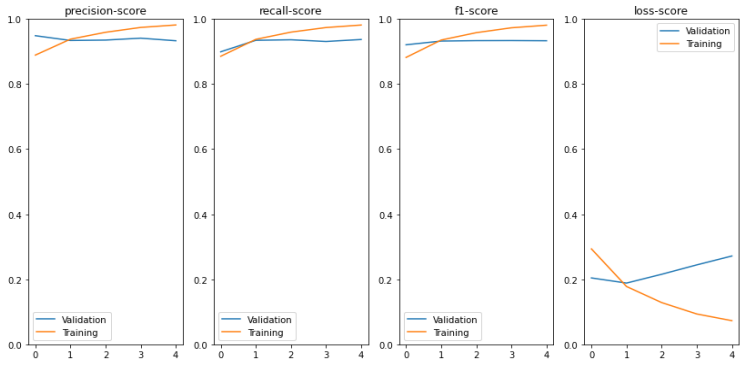
Also in this exercise gradient clipping is applied, which is a way to solve the problem of the exploding gradients, which refers to a large increase in the norm during training [19,20]. For the evaluations of the model sigmoid function is used after all Bert ,linear and dropout layers in our classifier in order to make the prediction numbers probabilities and also calculate metrics from sklearn such as precision, F1-score, recall, accuracy and roc curve [21,22,23].

Below some experiments of the Bert Classifier are represented:

- Optimizer = AdamW, dropout probability = 0.05 ,batch size = 32, initial learning_rate = 1e-5, padding maximum length for training dataset = 250, number of epochs = 5, with three hidden layers with hidden sizes 768,16,8, padding maximum length for validation dataset = 450, max_norm of gradient clipping by norm = 1.0, total time running (with calcuations, and ROC curves): 2 hours and 19 minutes and 56 seconds. The ROC curve of the last epoch is represented.

| Performance metric | Last epoch |
|---|---|
| **Loss function (train)** | 0.07270 |
| **Precision (train)** | 0.98177 |
| **Recall (train)** | 0.98159 |
| **f1-score (train)** | 0.98111 |
| **Acurracy (train)** | 0.98182 |
| **Loss function (validation)** | 0.27149 |
| **Precision (validation)** | 0.93324 |
| **Recall (validation)** | 0.93714 |
| **f1-score (validation)** | 0.93326 |
| **Acurracy (validation)** | 0.93426 |

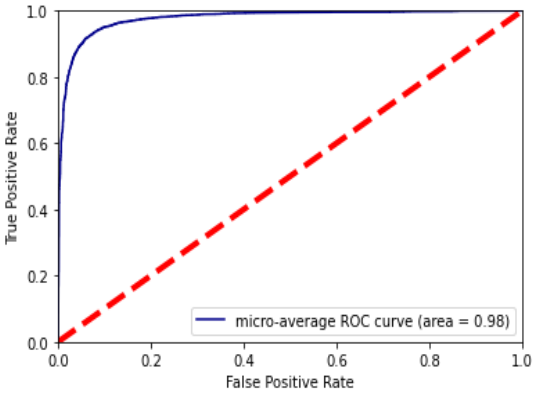


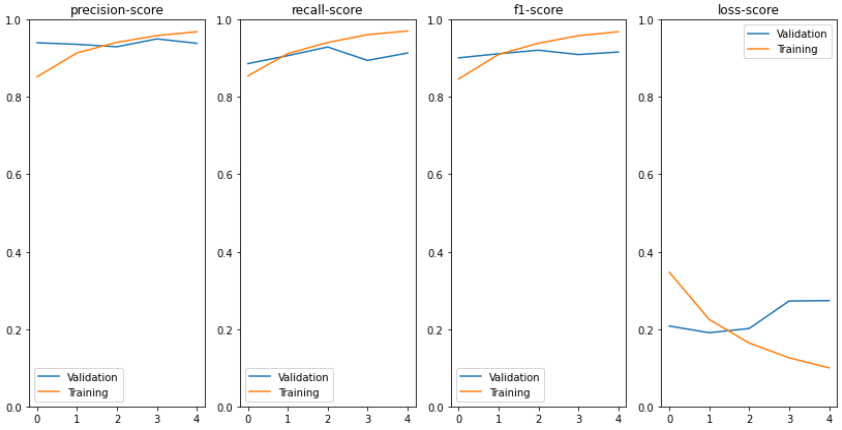




- Optimizer = AdamW, dropout probability = 0.05 ,batch size for training set and validation test correspondingly = 32,8, initial learning_rate = 1e-5, padding maximum length for training dataset = 125, number of epochs = 5, with three hidden layers, padding maximum length for validation dataset = 450, max_norm

of gradient clipping by norm = 1.0, total time running (with calcuations, and ROC curves): 1 hours and 27 minutes. The ROC curve of the last epoch is represented.

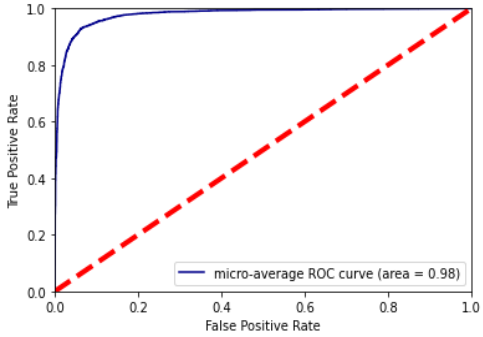
| Performance metric | Last epoch |
|---|---|
| **Loss function (train)** | 0.09979 |
| **Precision (train)** | 0.96873 |
| **Recall (train)** | 0.97069 |
| **f1-score (train)** | 0.96882 |
| **Acurracy (train)** | 0.96972 |
| **Loss function (validation)** | 0.27351 |
| **Precision (validation)** | 0.93884 |
| **Recall (validation)** | 0.91371 |
| **f1-score (validation)** | 0.91623 |
| **Acurracy (validation)** | 0.92773 |

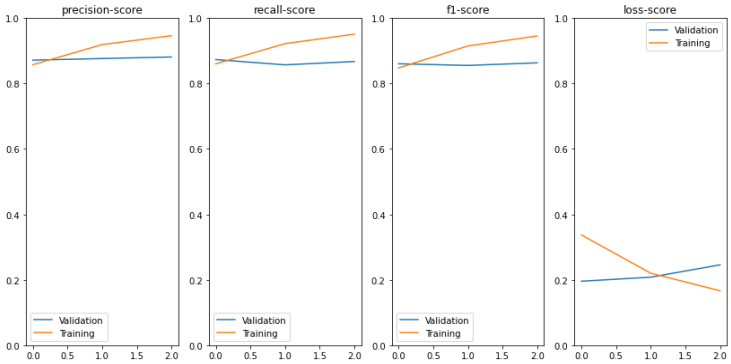



- Optimizer = AdamW, dropout probability = 0.05 ,batch size for training set and validation test correspondingly = 16,4, initial learning_rate = 1e-5, padding maximum length for training dataset = 125, number of epochs = 3, with three hidden layers with hidden sizes 768,16,8, padding maximum length for validation dataset = 450, max_norm of gradient clipping by norm = 1.0, total time running (with calcuations, and ROC curves): 54 minutes. The ROC curve of the last epoch is represented.

| Performance metric | Last epoch |
|---|---|
| **Loss function (train)** | 0.16656 |
| **Precision (train)** | 0.94634 |
| **Recall (train)** | 0.95104 |
| **f1-score (train)** | 0.94540 |
| **Acurracy (train)** | 0.94850 |
| **Loss function (validation)** | 0.24569 |
| **Precision (validation)** | 0.88127 |
| **Recall (validation)** | 0.86743 |
| **f1-score (validation)** | 0.86338 |
| **Acurracy (validation)** | 0.93125 |

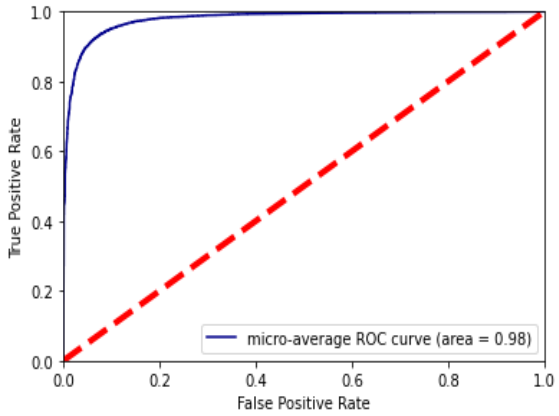


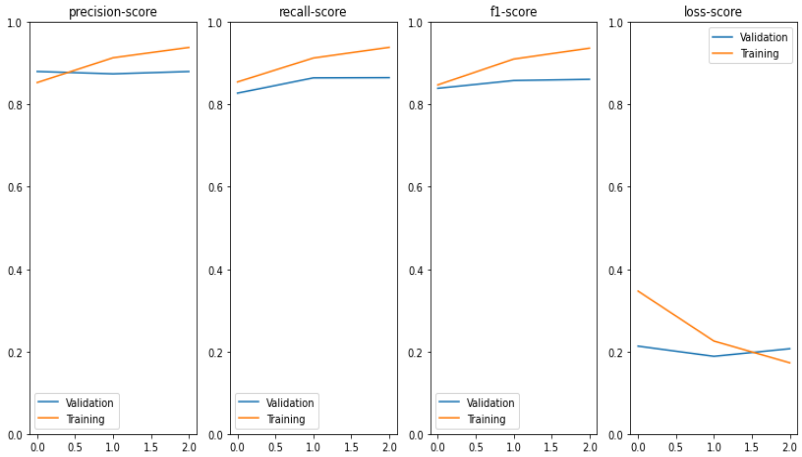




- Optimizer = AdamW ,batch size for training set and validation test correspondingly = 32,4, initial learning_rate = 1e-5, padding maximum length for training dataset = 125, number of epochs = 3, with three hidden layers with hidden sizes 768,16,8, padding maximum length for validation dataset = 450, max_norm of gradient clipping by norm = 1.0, total time running (with calcuations, and ROC curves): 53 minutes The ROC curve of the last epoch is represented.

| Performance metric | Last epoch |
|---|---|
| **Loss function (train)** | 0.17244 |
| **Precision (train)** | 0.93847 |
| **Recall (train)** | 0.93872 |
| **f1-score (train)** | 0.93667 |
| **Acurracy (train)** | 0.93828 |
| **Loss function (validation)** | 0.20666 |
| **Precision (validation)** | 0.87994 |

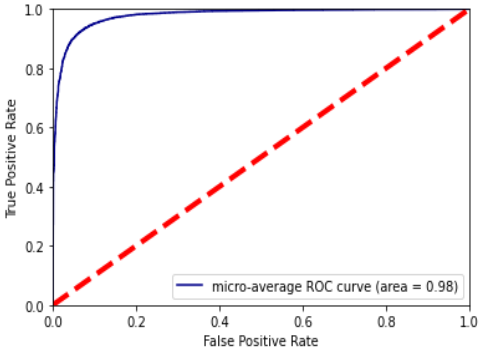
| | |
|---|---|
| **Recall (validation)** | 0.86510 |
| **f1-score (validation)** | 0.86108 |
| **Acurracy (validation)** | 0.92914 |

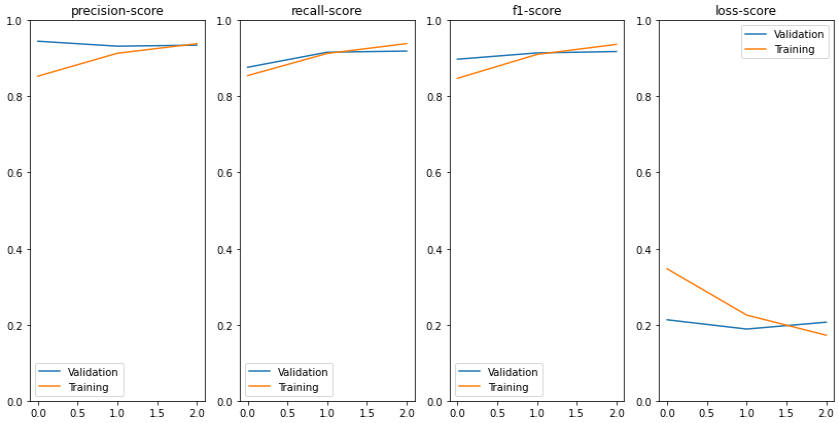



- Optimizer = AdamW ,batch size for training set and validation test correspondingly = 32,8, initial learning_rate = 1e-5, padding maximum length for training dataset = 125, number of epochs = 3, with three hidden layers with hidden sizes 768,16,8, padding maximum length for validation dataset = 450, max_norm of gradient clipping by norm = 1.0, total time running (with calcuations, and ROC curves): 53 minutes. The ROC curve of the last epoch is represented.

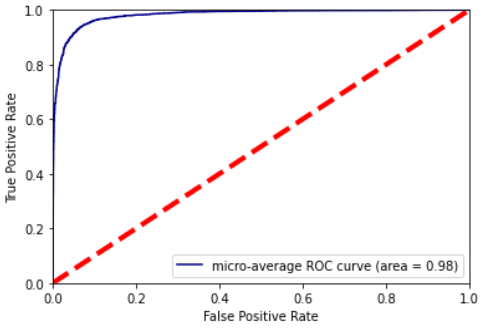
| **Performance metric** | **Last epoch** |
|---|---|
| **Loss function (train)** | 0.17244 |
| **Precision (train)** | 0.93847 |
| **Recall (train)** | 0.93872 |
| **f1-score (train)** | 0.93667 |
| **Acurracy (train)** | 0.93828 |
| **Loss function (validation)** | 0.20657 |
| **Precision (validation)** | 0.93460 |
| **Recall (validation)** | 0.91911 |
| **f1-score (validation)** | 0.91788 |
| **Acurracy (validation)** | 0.92917 |

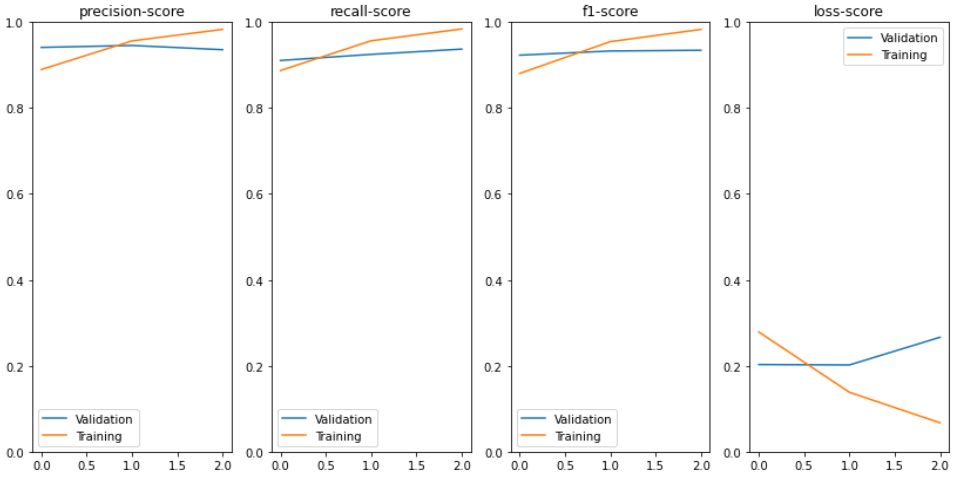



- Optimizer = AdamW, dropout probability = 0.05, batch size = 32, initial learning_rate = 5e-5, padding maximum length for training dataset = 250, number of epochs = 3, with three hidden layers with hidden sizes 768,16,8, padding maximum length for validation dataset = 450, max_norm of gradient clipping by norm = 1.0, total time running (with calcuations, and ROC curves): 1 hour and 37 minutes and 43 seconds. The ROC curve of the last epoch is represented.

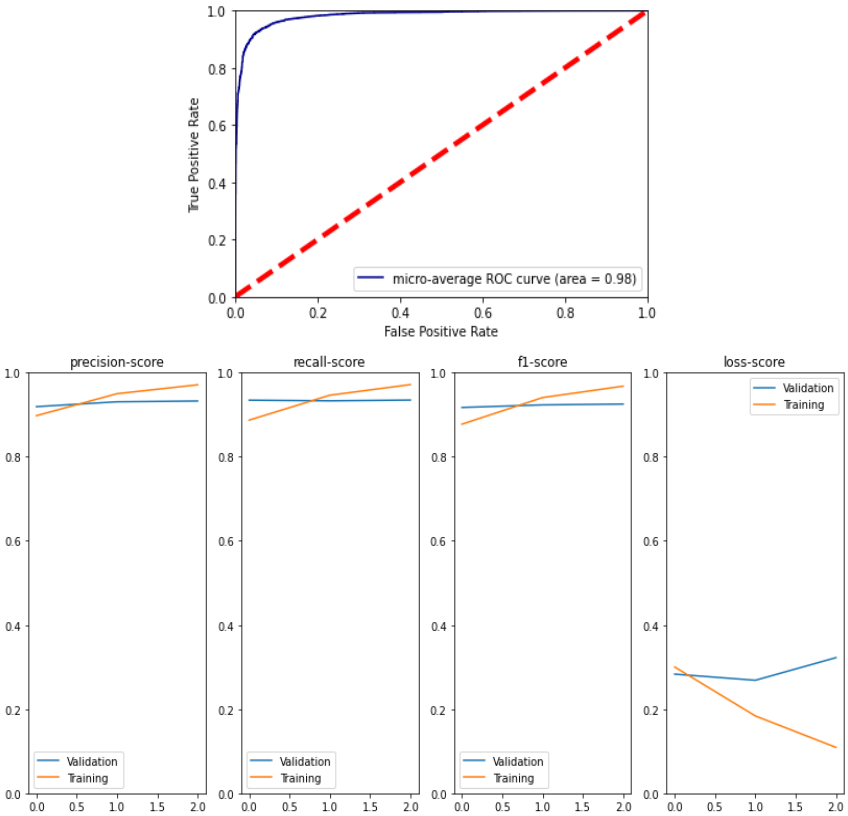
| Performance metric | Last epoch |
|---|---|
| **Loss function (train)** | 0.06717 |
| **Precision (train)** | 0.98318 |
| **Recall (train)** | 0.98411 |
| **f1-score (train)** | 0.98313 |
| **Acurracy (train)** | 0.98354 |
| **Loss function (validation)** | 0.26638 |
| **Precision (validation)** | 0.93584 |
| **Recall (validation)** | 0.93725 |
| **f1-score (validation)** | 0.93457 |
| **Acurracy (validation)** | 0.93570 |



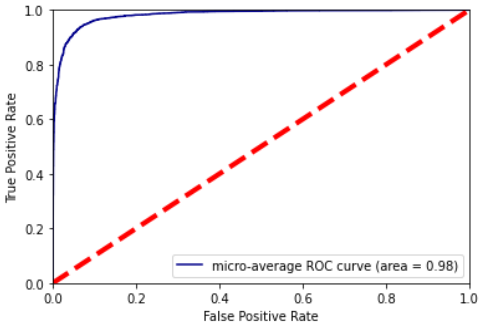

- Optimizer = AdamW, dropout probability = 0.05, batch size = 8, initial learning_rate = 1e-5, number of epochs = 3, with three hidden layers with hidden sizes 768,512,256, padding maximum length for training and validation dataset = 512, max_norm of gradient clipping by norm = 1.0, total time running (with calcuations, and ROC curves): 2 hours 59 minutes and 57 seconds.The ROC curve of the last epoch is represented.

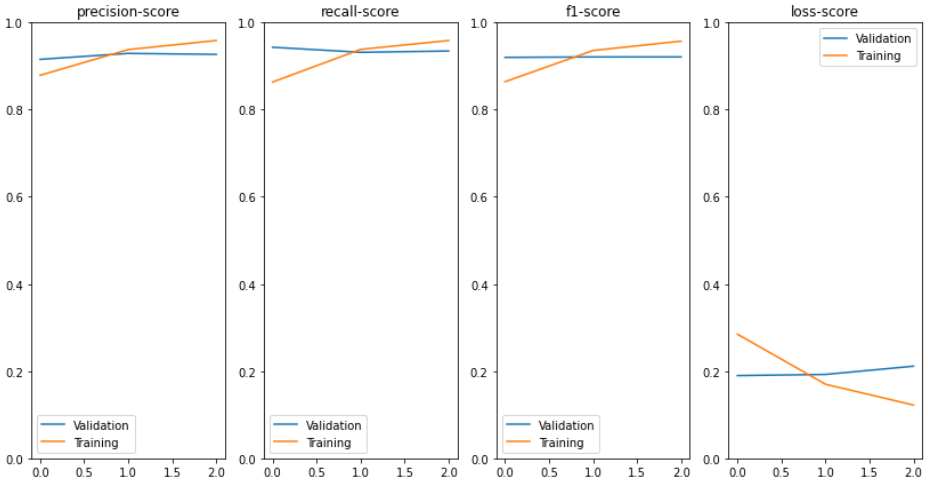
| Performance metric | Last epoch |
|---|---|
| **Loss function (train)** | 0.10907 |
| **Precision (train)** | 0.97105 |
| **Recall (train)** | 0.97153 |
| **f1-score (train)** | 0.96782 |
| **Acurracy (train)** | 0.97570 |
| **Loss function (validation)** | 0.32232 |
| **Precision (validation)** | 0.93251 |
| **Recall (validation)** | 0.93482 |
| **f1-score (validation)** | 0.92527 |
| **Acurracy (validation)** | 0.93583 |




- Optimizer = AdamW, dropout probability = 0.05, batch size for training and validation dataset correspondingly = 32,8, initial learning_rate = 1e-5, padding maximum length for training dataset = 250, number of epochs = 3, with three hidden layers with hidden sizes 768,512,256, padding maximum length for validation dataset = 450, max_norm of gradient clipping by norm = 1.0, total time running (with calcuations, and ROC curves): 1 hour and 20 minutes. The ROC curve of the last epoch is represented.

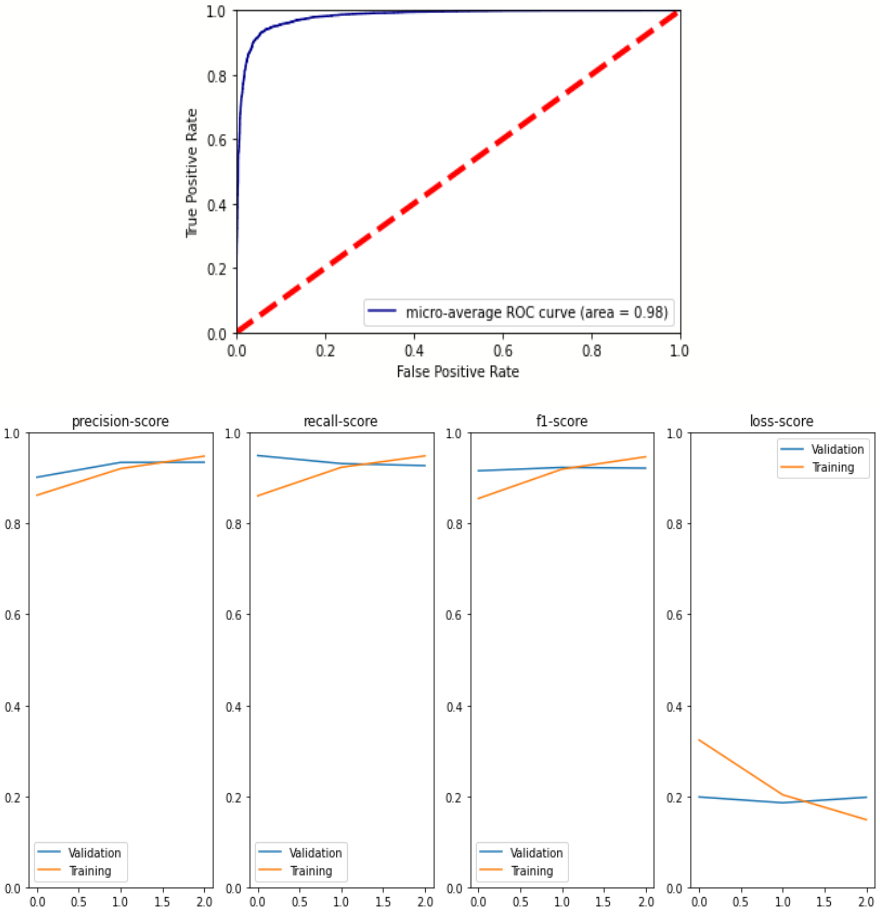
| Performance metric | Last epoch |
|---|---|
| **Loss function (train)** | 0.12218 |
| **Precision (train)** | 0.95867 |
| **Recall (train)** | 0.95865 |
| **f1-score (train)** | 0.95721 |
| **Acurracy (train)** | 0.95906 |
| **Loss function (validation)** | 0.21135 |
| **Precision (validation)** | 0.92686 |
| **Recall (validation)** | 0.93456 |
| **f1-score (validation)** | 0.92102 |
| **Acurracy (validation)** | 0.93162 |




- Optimizer = AdamW, dropout probability = 0.05, batch size for training and validation dataset correspondingly = 32,8, initial learning_rate = 1e-5, padding maximum length for training dataset = 125, number of epochs = 3, with three hidden layers with hidden sizes 768,512,256, padding maximum length for validation dataset = 512, max_norm of gradient clipping by norm = 1.0, total time running (with calcuations, and ROC curves): 1 hour and 3 minutes and 6 seconds. The ROC curve of the last epoch is represented.

| Performance metric | Last epoch |
|---|---|
| **Loss function (train)** | 0.14827 |
| **Precision (train)** | 0.94815 |
| **Recall (train)** | 0.94876 |
| **f1-score (train)** | 0.94683 |
| **Acurracy (train)** | 0.94843 |
| **Loss function (validation)** | 0.19762 |
| **Precision (validation)** | 0.93480 |
| **Recall (validation)** | 0.92736 |
| **f1-score (validation)** | 0.92176 |
| **Acurracy (validation)** | 0.93735 |




In all the above experiments the main purpose is the performance of the train and validation set to be good and also very near each other. Padding as it mentioned before is a very important parameter about the performance [2,11], because it is very helpful to have all the text reviews in the same size for each set.

For all these reasons for the validation there are experiments with padding maximum length values of 450 and 512 and for the training set, there are experiments with values of 125,250 and 512. From tokenizer.encode_plus it is important to refer that any input with token length less than max length is padded up to max length and inputs with token length greater than max length are truncated from its ends[2,11].

For smaller values of the padding maximum length the model training will be fast but a lot of information might be lost and finally the performance of the model is not so good, on the other hand for bigger values of the padding maximum length, the training of the model is more slow, but there is no lost or very small loss of information and as a result the performance of the model is much more better[2].

Additionally in the first two experiments the model is running for 5 epochs and the rest of the experiments are running for 3 epochs. It is observed that 3 epochs is a very good number in order to understand how the neural network of the BERT classifier is being learned. This is proven from the learning curves, especially for the loss, where the convergence of the training and validation curves are not so good for 5 epochs compared to the cases of running the model for 3 epochs.

Generally the metrics referring to its numerical values over the epochs in each experiment is very satisfactory. The last two experiments it is observed that they have the best performance about the convergence of the curves and on the numerical values of the metrics compared to all previous experiments, due to the bigger number of hidden units(neurons) in each hidden layer in the model of Bert classifier after the Bert model. The best maximum value for padding is 512 and it is proven from the distribution graph for the number of words in each review with the help of with the loading of seaborn library in python. The value of maximum padding length consists the values for which most of the reviews is covered without truncation. This value is decided to be 512, because the value that occurs from the distribution graph is above 512 and 512 is the maximum acceptable value for padding in Bert [2].

That is also a reason that the two last experiments have the best performance of all other experiments. In all the experiments the combination of all the hyperparameters(learning rate, number of hidden layers, hidden units per hidden layer, dropout probability, batch sizes etc), and maximum padding length is the best suitable combination in order to have the best possible results without having any memory issues in google colab[24].

The main reason that pooled output or else [CLS] token, that must be added to the start of each sentence, so BERT knows we're doing classification [2,6,11], assumed as the BERT output in this exercise is that must be always used as a starting token while performing pretraining using a masked language model and next sentence prediction that's why a pooler output which itself doesn't have any interpretability but it captures all the information for a particular input [6].

Finally there is an experimentation with learning rate:1e-5,5e-5 and batch size: 4,8,16,32,64. It is occurred that increasing the batch size reduces the training time significantly, but gives you lower accuracy, About learning rate, the higher value it has, the model is more quicker learned, something that is shown from the steep variation in the train and validation learning curves for all the metrics(recall, f1-score, precision, loss).

About the ROC curves [23] the results are very good because for all of the above experiments the value is 0.98, which means that true positive rate and false negative rate illustrates the same result. In fact the predicting power of the classifier is very big because the curve is getting closer to the top left corner i.e point (0.0, 1.0) and also because it is farer from the random classifier (dotted line in each roc curve). In this exercise the classifier is getting closer to be perfect.

After training the model and preprocessing of the test set as described above, the next step is to predict the test set. Test set is set as none through the variable test_data, in order to evaluate each test set, only by passing the path of whichever test set, for example test_data = pd.read_csv(path_file) (or for example if we say test_data = into the variable test_data. Also the path for the test set can be occurred through the splitting of the validation set as the 50% of this set through train_test_split [10]. The preprocessing, the tokenization through BertTokenizerFast and the corresponding evaluations of the model of the BERT classifier are done as described in previous paragraphs.

About the saving of our model it is implemented through saving by interference, where it is only necessary to save the trained model's learned parameters. Saving the model's state_dict with the `torch.save()` function will give you the most flexibility for restoring the model later, which is why it is the recommended method for saving models [25].

## References


1. The BERT Model (uoa.gr)
2. Fine Tuning BERT-base Using PyTorch for Sentiment Analysis | by Niteshkumardew | Medium
3. The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)
4. The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning) – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)
5. A Visual Guide to Using BERT for the First Time – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)
6. Implementing BERT for Question and Answer | by Niteshkumardew | Medium
7. Models - Hugging Face

8. BERT (huggingface.co)
9. Utilities for Tokenizers (huggingface.co)
10. sklearn.model_selection.train_test_split — scikit-learn 1.2.0 documentation
11. Sentiment Analysis with BERT and Transformers by Hugging Face using PyTorch and Python | Curiously - Hacker's Guide to Machine Learning
12. torch.utils.data — PyTorch 1.13 documentation
13. AdamW — PyTorch 1.13 documentation
14. Optimization (huggingface.co)
15. Automatic Differentiation with torch.autograd — PyTorch Tutorials 1.13.0+cu117 documentation
16. BERT (huggingface.co)
17. Linear — PyTorch 1.13 documentation
18. Dropout — PyTorch 1.13 documentation
19. Vanishing Gradients and Fancy RNNs (uoa.gr)
20. Understanding Gradient Clipping (and How It Can Fix Exploding Gradients Problem) - neptune.ai
21. API Reference — scikit-learn 1.2.0 documentation
22. Regression (uoa.gr)
23. Receiver operating characteristic - Wikipedia
24. Artificial Neural Networks - Backpropagation (uoa.gr)
25. Saving and Loading Models — PyTorch Tutorials 1.12.1+cu102 documentation