Political Blog Classification using Graph Convolutional Networks

Introduction and description of the problem

The aim of this project is to classify political blogs as 'liberal' or 'conservative'. The blogs are collected in the 'Polblogs' dataset, which contains information on individual blogs and how they mention other blogs in the same dataset, forming a graph structure.

The graph structure of the dataset pushes towards the choice of a model such as Graph Neural Networks (GNNs), proposed in 2005 in the paper by M. Gori, G. Monfardini and F. Scarselli entitled 'A new model for learning in graph domains', due to their ability to capture information from individual nodes and connections between adjacent nodes.

In particular, this project exploits a specific type of GNNs such as Graph Convolutional Networks (GCNs) by starting with a simple architecture inspired by the one proposed by Thomas N. Kipf and Max Welling in their 2017 paper "Semi-Supervised Classification with Graph Convolutional Networks" and then enhancing it with appropriate architectural modifications and training techniques according to the needs dictated by the dataset in question.

All the phases and architectures described here have been implemented within the attached Python project entitled 'TE_GCN.ipynb'.

Dataset description and preprocessing

Loading, analysing and visualising the dataset

The dataset used is the one recorded and analysed in the paper "The political blogosphere and the 2004 US Election" by L. A. Adamic and N. Glance proposed in 2005, which contains a collection of the political blogs that were most active in 2004 in the run-up to the US presidential election.

The dataset was downloaded from the 'https://websites.umich.edu/~mejn/netdata/polblogs.zip' site, which returns a zip folder containing a text document with information about the dataset and a file in GML (Graph Modelling Language) format containing all the necessary elements for describing the graph. The igraph library was mainly used for reading and manipulating the graph.

Analysing the structure of the dataset, it emerges that it contains a single directed graph, within which each node represents a blog and each arc represents a citation made to another blog. Classifying blogs therefore means that the task to be solved is a Node Classification.

The graph contains 1490 nodes and 19090 arcs, with an average of 25.62 links per node.

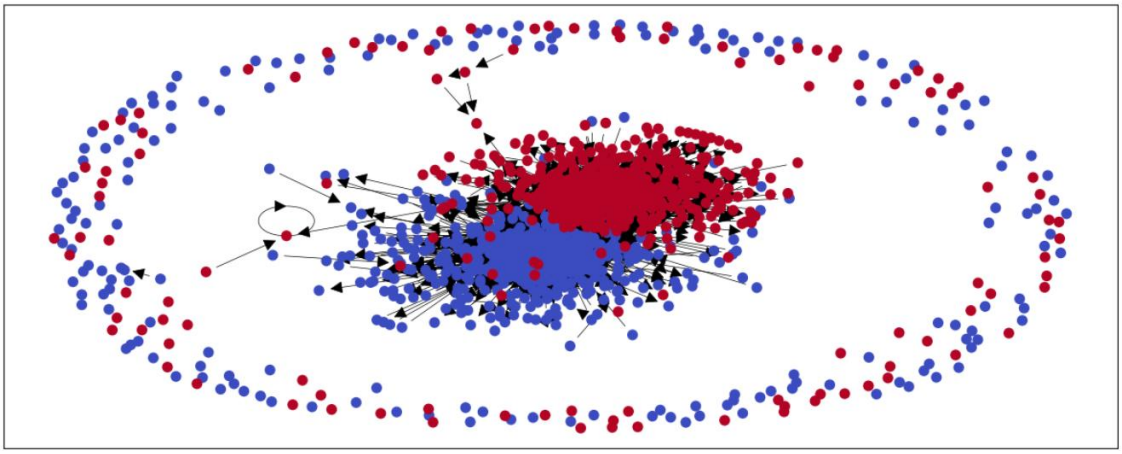
The node attributes are as follows

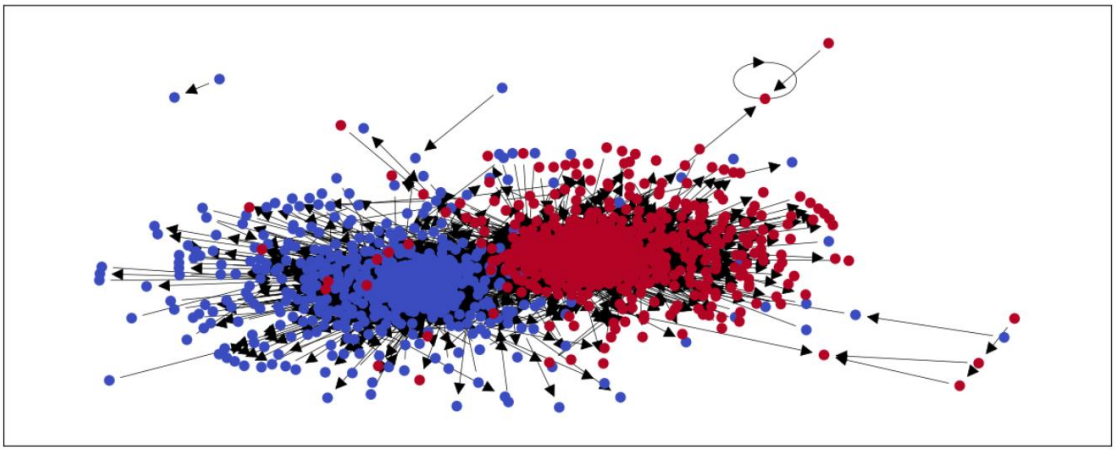
- id: identification value for each node
- label: address at which the blog is available
- value: value indicating the political orientation of the blog
    - 0: Liberal or left-wing blogs. There are 758 blogs belonging to this class
    - 1: Conservative or right-wing blogs. There are 732 blogs belonging to this class
- source: blog directory

However, there are no attributes on the arcs or global attributes.

An initial analysis of the graph showed that it contains 'self-loops', i.e. arcs connecting a node with itself, and contains isolated nodes, i.e. blogs that do not mention any other blogs. Attention will be paid to this latter aspect later on.

Once the graph information had been explored, a visualisation of the graph was provided with the help of the networkx and matplotlib libraries. Of the images below, the first depicts the complete graph, in which the nodes representing liberal blogs are coloured in blue and the nodes representing conservative blogs are represented in red; the second image focuses instead on the non-isolated nodes only, following the same colour code as the first image.





Creating Embeddings

GNNs adopt a 'graph-in, graph-out' architecture in that both input and output are a graph structure. Each node is transformed into an embedding, which is gradually updated during training until the node representation is as aware as possible of its neighbouring elements.

In this respect, it is necessary to choose how embeddings must be initialised. This can be done, for instance, by initialising them randomly, through an identity matrix, or by adopting other information not present in the dataset.

The chosen approach was the last one listed. For this reason, a web scraping operation was carried out to retrieve textual information from each of the blogs, and then exploit the results obtained as starting information for the creation of the embeddings.

In particular, the web scraping part was carried out using the 'BeautifulSoup' library, and was aimed at creating a dictionary whose keys were the ids of the blogs and whose values were the textual information extracted from the corresponding blog.

Given the variety of the structure of the HTML documents describing the various blogs, it was decided to read the textual content within the first block found from among those belonging to the following classes: 'post-body entry-content', 'entry-body', 'post' as it was observed that in most cases the information

content was to be found in blocks belonging to one of the above-mentioned classes. During the process, it emerged that approximately 460 blogs are now unreachable for various reasons; the name of the blog was used as the textual content associated with the blogs that could no longer be found.

Once the dictionary was obtained, the embedding creation phase started. This was done with the BERT (Bidirectional Encoder Representations from Transformers) model, a Transformer Encoder Stack proposed by Google for the creation of contextualised embeddings.

The words contained in the text fragments extracted from the various blogs were first converted into tokens by means of the BertTokenizer, and then proceeded to the embedding production phase with BERT. The policy chosen for the extraction of the embeddings from the output of BERT was to select the information content of the CLS token, in which it contains the meaning of the entire portion of text.

At the end of this process, the embeddings obtained were saved within a csv file to avoid repeating the same steps of scraping and obtaining embeddings each time the entire algorithm was run, as they were time-consuming.

Noting that BERT returns embeddings of 768 dimensions, a dimensionality reduction technique was adopted to assess whether it was actually necessary to provide such large inputs to the model. This step is motivated by the Manifold Hypothesis, according to which information from the real world, such as portions of text extracted from blogs, described with a high number of dimensions are actually representable by a smaller number of dimensions without loss of information.

This made it possible to provide input to the embedding model that was not too large that would have slowed down the training process or since it would have required a larger architecture.

The technique chosen to reduce the dimensionality of the embeddings was PCA (Principal Component Analysis), which was first used to calculate the number of dimensions expressing 90% of the variance and then to extract the identified components. The first operation showed that the first 134 components were sufficient; therefore, the embeddings produced by BERT were projected via PCA into a space of 134 dimensions.

The use of text embeddings as model input suggested to name the architectures to be presented 'TE_GCN', i.e. 'Text Embeddings Graph Convolutional Network'.

Data preparation

The layers used to create the model are those provided by the torch_geometric library, which operate on objects of type Data(x, edge_index, y). For this reason, it was necessary to transform the graph into a structure of this type. Since igraph does not expose a method for directly creating an object of type Data from a graph, the object was created manually by assigning:

- the embedding matrix at variable x
- The edge_index variable was assigned the edge_list structure provided by igraph, which contains information on the connectivity of the graph in COO (Coordinate format), i.e. type tuples (source, destination) for each arc. In fact, the edge_list structure is a sparse representation of the adjacency matrix
- the 'value' column of the node dataframe at variable y

Attention is drawn to the order in which the nodes are presented in the dataset. In particular, by exploring the values of the target variable y, it emerges that the data are arranged in an orderly fashion, i.e. there are first all liberal blogs and then all conservative blogs. To avoid model bias due to these groupings, a shuffle is performed on the data. It has to be emphasized that, since this is a graph structure, the position of the nodes in the dataset is crucial for the correct representation of the arcs; in this regard, once the position of the nodes in the dataset had changed, all the arcs in edge_index had to be updated so that they took into account the new position of the nodes in the dataset.

The preprocessing concludes with the division of the graph into training, validation and test set according to a 60-20-20 proportion, using the train_test_split method provided by the sklearn library. The object of type Data now contains 3 new lists that are in fact Boolean masks which mask the nodes that are not to be considered during a certain phase.

Initially, the method 'RandomNodeSplit' provided by the torch_geometric library was used to divide the dataset into training, validation and test sets. However, this approach was later discarded due to the lack of the possibilities in RandomNodeSplit to stratify the sets according to the target variable and to insert a seed for reproducibility, which are present in train_test_split.

Introduction to GCNs

Once the data preprocessing phase was over, it was necessary to choose the type of architecture to be adopted in the model. The graph structure of the dataset suggested the use of a Graph Neural Network, an architecture in which each node of the graph is represented by an embedding that is iteratively updated based on information from neighbouring nodes or arcs according to the 'message passing' paradigm. The updating process occurs in two stages:

1. Aggregation, during which information is gathered from neighbouring nodes through a generic function that is learnt during training.
$$m_i^{(l)} = \sum_{j \in N(i)} f^{(l)}(h_j^{(l-1)}, h_i^{(l-1)}, e_{ij})$$
2. Updating the embedding of the current node, through a function that is learnt during training
$$h_i^{(l)} = g^{(l)}(h_i^{(l-1)}, m_i^{(l)})$$

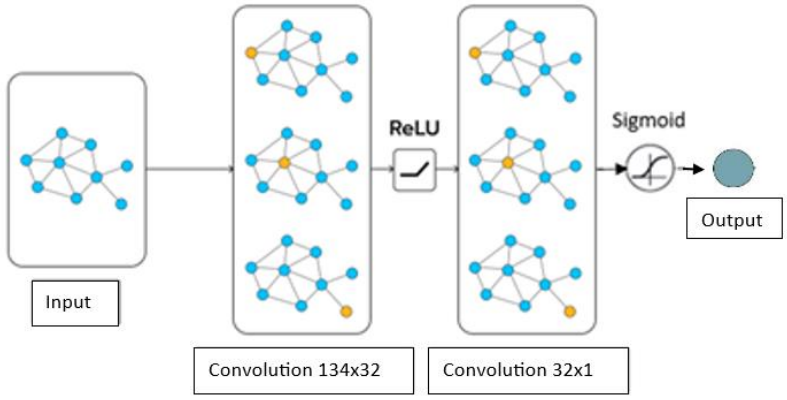
In particular, the architecture implemented is a Graph Convolutional Network (GCN), a specific type of GNN that applies the concept of convolution to the nodes of a graph. The convolution operation can be thought of as 'message passing' between neighbouring nodes. By repeating the convolution operation N times, a node obtains information about another node N steps away from it. Specifically, the two steps become:

1. Aggregation: calculation of the weighted average of the attributes of neighbouring nodes and the current node at the previous step
$$m_i^{(l)} = \sum_{j \in N(i)} \frac{1}{\sqrt{d_i d_j}} h_j^{(l-1)}$$
2. Update the embedding of the current node
$$h_i^{(l)} = \sigma\left(W^{(l)} m_i^{(l)}\right)$$

The reason for adopting a GCN lies in the nature of the data, as it is clear that for the purposes of prediction, it is important to take into account the blogs that have mentioned the blog on which the prediction is to be made.

Defining the model

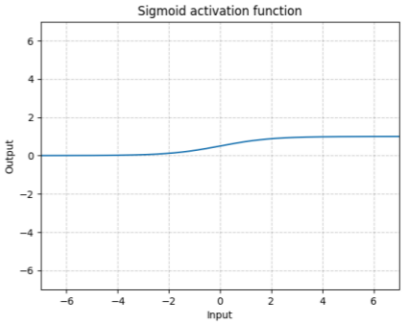
For the implementation of the model, it was chosen to start from one of the simplest of the best-performing models on a dataset similar to Polblogs, such as Cora, according to the classification at 'Cora Benchmark (Node Classification) | Papers With Code'. The selected model was proposed in 2017 by Thomas N. Kipf and Max Welling in the paper "Semi-Supervised Classification with Graph Convolutional Networks" and its implementation in PyTorch can be found at the following repository "GitHub - tkipf/pygcn: Graph Convolutional Networks in PyTorch". After an analysis phase, the model was adapted to the task to be solved and was defined as follows:



The input size is equal to the embedding size after dimensionality reduction, i.e. 134; the output size is 1, since the task is binary classification.

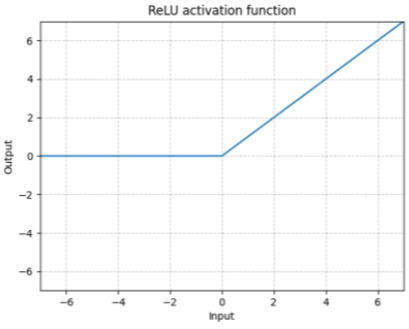
The main difference to the model proposed by Thomas N. Kipf and Max Welling is the use of the sigmoid as the final activation function, since the task is of binary classification unlike the classification of nodes in CORA where there are seven different classes.

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + exp(-x)}$$



Similar to the initial model, ReLU (Rectified Linear Unit) was chosen as the activation function between the intermediate levels, due to its computational simplicity and its ability to avoid the vanishing gradient problem.

$$\text{ReLU}(x) = (x)^+ = \max(0,x)$$



The use of Droput was also maintained to avoid overfitting.

Training and performance evaluation

In accordance with the activation function chosen for the output layer, Binary Cross-Entropy (BCE) was used as the cost function

$$l(x, y) = L = \{l_1, \dots, l_N\}^\intercal, \quad l_n = -w_n[y_n \log x_n + (1 - y_n) \log (1 - x_n)]$$

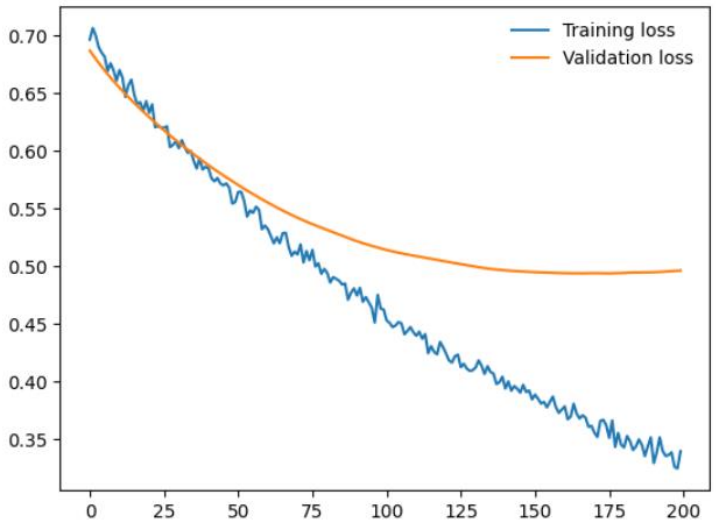
The optimisation algorithm adopted is Adam (Adaptive Moment Estimation), due to its ability to calculate adaptive learning rates for each model parameter by exploiting both first-order and second-order momentum.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \qquad \hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

The model described was trained on the training set and evaluated at each epoch on the validation set for 200 epochs.

The process gave rise to the following learning curves, from which it can be seen that the model learns correctly from the data for the first 100 epochs and then goes into overfitting, as the loss on the training data continues to decrease while that on the validation data remains constant and then starts to increase again.



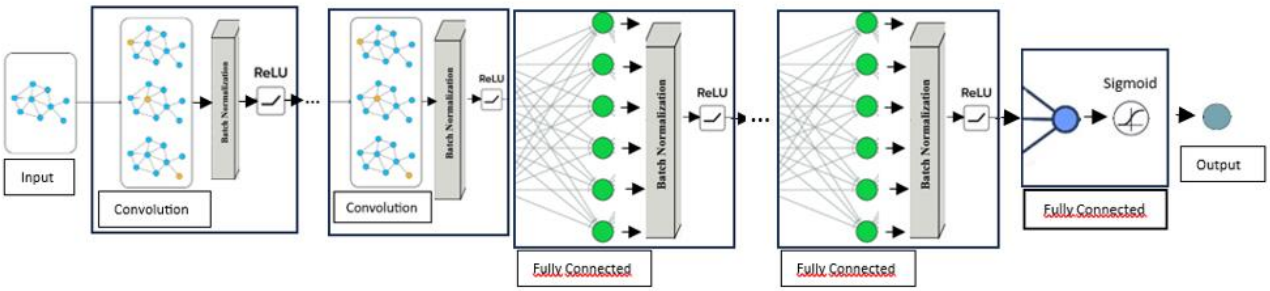
The final accuracy measured on the test set is 77.5% and can be considered a good result.

Architecture improvement

The proposed model performs well, but there are several points on which action can be taken to improve it:

- addition of normalization levels to keep gradients stable during training, reducing the risk of vanishing gradients
- addition of linear levels between the last convolution level and the output level to improve the model's classification capability

The modified architecture looks as follows:



As will be better explained shortly, the number and size of levels were considered as a hyper-parameter.

In addition, the following changes were made to the training loop with the aim of speeding up learning and at the same time avoiding overfitting:

- initialization of weights to avoid vanishing or exploding gradients. The chosen technique is Xavier (Glorot) initialisation, which initialises weights taking into account the number of inputs and outputs of each neuron

$$w \propto \frac{np.random.randn()}{n_{in} + n_{out}}$$

$$\mathcal{N}(0, \sigma^2) \quad \Rightarrow \quad \sigma = gain \times \sqrt{\frac{2}{n_{in} + n_{out}}}$$

- adoption of early stopping, i.e. stopping learning when the performance of the model on the validation set deteriorates
- use of AdamW as an optimization algorithm, a variant of Adam with regularization

Training the new model and validation

Once the new architecture was defined, the training phase took place by means of a grid search, i.e. the definition and training of several models each with a different combination of hyperparameter values with the aim of finding the best combination. The hyperparameters of the model are:

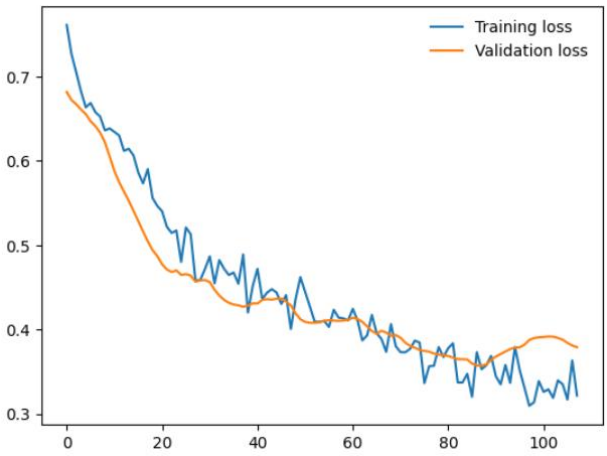
- number and size of convolution levels
- number and size of linear levels
- dropout probability
- learning rate
- weight decay

As many models as there were possible combinations due to the different values of each of the hyper-parameters were then trained and validated on the validation set to select the best model. During training, the model with the highest measure of accuracy achieved so far was saved in the file 'best_model_full.pth', saved together with its training loss and validation loss values.

At the end of the validation phase, the checkpoint of the best model was loaded, with an accuracy on the validation set of 87%. The best combination of hyper-parameters turned out to be the following:

- 4 convolution levels with 16 neurons each
- 2 linear levels with 16 and 8 neurons
- 30% dropout probability
- Learning rate 0.01
- Weight decay 1e-04

The selected model has the following learning curves



Compared to the curves obtained in the first architecture, the following differences can be observed:

- the training loss drops more slowly as a result of all the techniques adopted to avoid overfitting
- training stops when the validation loss does not improve for at least 30 epochs, due to early stopping
- the validation loss goes down and does not go up again as a consequence of early stopping

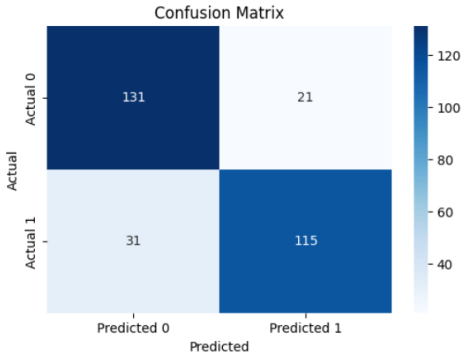
It is also noticeable that the best performing model is the one with a larger number of convolutional levels with a few units in each, in accordance with the concept of feature extraction in deep learning. Thus, it emerges that deep learning models perform better with a greater number of small levels than models with a few large levels. In the context of convolution, having 4 levels means that nodes in the last level are classified by exploiting information from nodes up to 4 steps apart.

Test results

After having identified the best model among those evaluated in the validation phase, predictions were made on the test set, saving in the variable 'probabilities' a tensor of the type 'Probability class 0, Probability class 1'. The percentages assigned to each of the two classes are shown below for a sample of 20 samples from the test set; a comparison between the predicted class and the actual class is shown alongside.

```
First 20 rows test_output:
Percentage for Liberal: 0.05, Percentage for Conservative: 0.95        Prediction: 1, Target: 1
Percentage for Liberal: 0.05, Percentage for Conservative: 0.95        Prediction: 1, Target: 1
Percentage for Liberal: 0.93, Percentage for Conservative: 0.07        Prediction: 0, Target: 0
Percentage for Liberal: 0.85, Percentage for Conservative: 0.15        Prediction: 0, Target: 0
Percentage for Liberal: 0.58, Percentage for Conservative: 0.42        Prediction: 0, Target: 0
Percentage for Liberal: 0.91, Percentage for Conservative: 0.09        Prediction: 0, Target: 0
Percentage for Liberal: 0.05, Percentage for Conservative: 0.95        Prediction: 1, Target: 1
Percentage for Liberal: 0.81, Percentage for Conservative: 0.19        Prediction: 0, Target: 0
Percentage for Liberal: 0.05, Percentage for Conservative: 0.95        Prediction: 1, Target: 1
Percentage for Liberal: 0.05, Percentage for Conservative: 0.95        Prediction: 1, Target: 1
Percentage for Liberal: 0.88, Percentage for Conservative: 0.12        Prediction: 0, Target: 1
Percentage for Liberal: 0.90, Percentage for Conservative: 0.10        Prediction: 0, Target: 0
Percentage for Liberal: 0.05, Percentage for Conservative: 0.95        Prediction: 1, Target: 1
Percentage for Liberal: 0.92, Percentage for Conservative: 0.08        Prediction: 0, Target: 0
Percentage for Liberal: 0.86, Percentage for Conservative: 0.14        Prediction: 0, Target: 1
Percentage for Liberal: 0.06, Percentage for Conservative: 0.94        Prediction: 1, Target: 1
Percentage for Liberal: 0.05, Percentage for Conservative: 0.95        Prediction: 1, Target: 1
Percentage for Liberal: 0.10, Percentage for Conservative: 0.90        Prediction: 1, Target: 1
Percentage for Liberal: 0.05, Percentage for Conservative: 0.95        Prediction: 1, Target: 1
Percentage for Liberal: 0.92, Percentage for Conservative: 0.08        Prediction: 0, Target: 0
```

The confusion matrix and classification report are presented as follows:



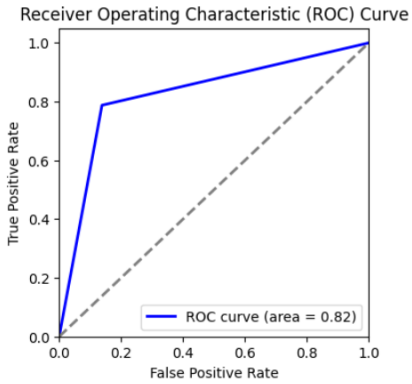
Test set Classification Report:

	precision	recall	f1-score	support
0	0.81	0.86	0.83	152
1	0.85	0.79	0.82	146
accuracy			0.83	298
macro avg	0.83	0.82	0.82	298
weighted avg	0.83	0.83	0.83	298

An interpretation of the data obtained shows that:

- The accuracy on test data is 0.83 (82.6 rounded to 83), which indicates that the model is performing well
- The precision is high for both classes, with 0.81 for liberal blogs and 0.85 for conservative blogs, indicating a good ability of the model to avoid false positives. The accuracy score for class 1 is higher, indicating a better ability of the model to correctly classify conservative blogs
- The recall for liberal blogs is very high (0.86), meaning that the model correctly identified most liberal blogs. The recall for conservative blogs is slightly lower (0.79), suggesting that there are more false negatives for this class
- The F1-Score is very similar for both classes (0.83 and 0.82), representing a good balance between precision and recall

The ROC curve is presented below, recording an Area Under the Curve value of 0.82



An analysis of the model's performance shows that it is an improved version of the architecture initially proposed. A comparison of the two learning curves shows how the new model is able to learn faster and how, thanks to early stopping, it terminates the learning process at the best moment of trade-off between training loss and validation loss, avoiding overfitting.

In conclusion, the model has good classification capabilities but still has points for improvement.

Critical points and points for improvement

The most critical decisions to be made during data pre-processing were:

- the choice of input to be provided to BERT for the creation of the embedding for all those sites that are no longer reachable. After several attempts, it was decided to adopt the name of the site as it was often very explicit about the political orientation of the blog.
- The choice of the technique for reducing the dimensionality of the embeddings and the choice of the dimension to which the embeddings should be reduced. The search phase for encoder models that could project the embeddings into a space with fewer dimensions was very difficult as no trained models were found with 768-dimensional embeddings as input and embeddings as output with fewer dimensions. Concerning the number of dimensions, thanks to PCA it was possible to find the optimal number of dimensions of the new space in which to project the embeddings.
- consider or exclude isolated nodes. It was decided to consider both approaches, of which the first is the one proposed and the second will be presented shortly.

With regard to architectural choices, the choice of the number of levels and their size was critical. During the learning process, it was observed that models with several small levels performed better than models with a few large levels. This underlines the ability of deep learning models to obtain high-level representations from low-level features.

Points of improvement of the model are:

- use a complete dataset, with no missing sites
- explore other techniques for obtaining portions of text from blogs
- explore other text embedding techniques
- explore more complex GNN architectures using other types of layers
- cross-validation to avoid the choice of the best model being 'biased' towards the validation set

Exclusion of isolated nodes

As mentioned above, the provided graph contains isolated nodes, namely 266 blogs that do not mention or are not mentioned by any other blog. Since the main strength of GCNs lies in the message passing mechanism, through which nodes obtain information from the nodes they are connected to, it was chosen to evaluate the performance of the trained model by considering the 1224 non-isolated nodes as nodes in the dataset.

The preprocessing phase was similar to that done on the complete dataset. In addition, both architectures were trained, i.e. both the initial GCN-inspired one and the improved one.

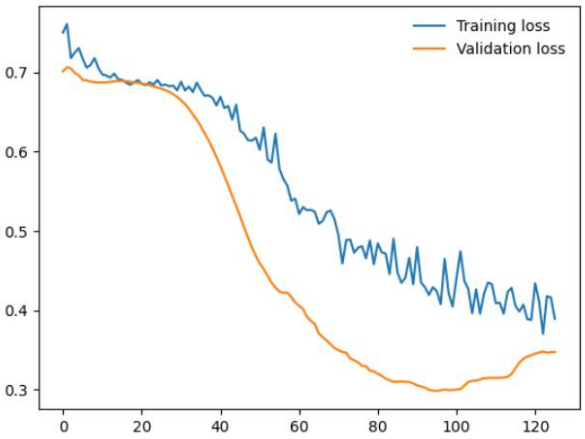
It is specified that one is aware of the fact that using a smaller number of samples while maintaining the same architecture leads to an overfitting scenario. However, it was decided not to make any substantial changes to the architecture for two reasons:

1. to evaluate the performance of models with the same architecture trained on two different but similar datasets
2. due to the fact that during validation, the grid search can select a smaller number of layers if such a model performs better than others

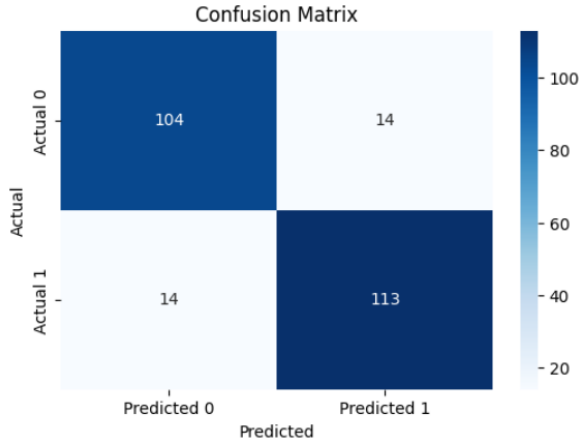
The results were very promising, as the accuracy with the first architecture was 88.16%, far higher than the accuracy obtained with the best model found on the complete dataset.

With regard to the results of the validation phase, the same hyper-parameters were selected as for the model trained on the entire dataset. The only difference was the choice of a higher dropout value, which makes a lot of sense since, having less data, the model must be simpler to avoid overfitting.

The best model identified from the validation phase, saved in the file 'best_model_partial.pth' achieved an accuracy on the test set of 88.57%, and produced the following learning curves.

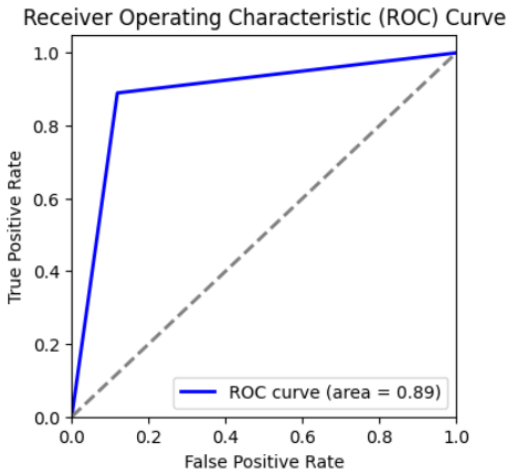


The following confusion matrix, classification report and ROC curve were also obtained:



Test set Classification Report:

	precision	recall	f1-score	support
0	0.88	0.88	0.88	118
1	0.89	0.89	0.89	127
accuracy			0.89	245
macro avg	0.89	0.89	0.89	245
weighted avg	0.89	0.89	0.89	245



In the end, it can be seen that all performance values have improved and they are much more balanced between the two classes compared to previous results.