Homework 4 - Data Mining - Sapienza

Ivan Fardin 1747864

January 10^{th}, 2021

Contents

1 Problem 1

1.1

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & ... & \mathbf{A}_{1d} \\ & \vdots & \\ \mathbf{A}_{n1} & ... & \mathbf{A}_{nd} \end{bmatrix}, \mathbf{A}^T = \begin{bmatrix} \mathbf{A}_{11} & ... & \mathbf{A}_{n1} \\ & \vdots & \\ \mathbf{A}_{1d} & ... & \mathbf{A}_{nd} \end{bmatrix} \Rightarrow \mathbf{A}\mathbf{A}^T = \begin{bmatrix} \mathbf{A}_{11}^2 + ... + \mathbf{A}_{1d}^2 & ... & \\ & \ddots & \\ & ... & \mathbf{A}_{n1}^2 + ... + \mathbf{A}_{nd}^2 \end{bmatrix}$$

Where in the transpose of the matrix I kept fixed the indexes (but clearly $\mathbf{A} \in \Re^{n \times d}$ and $\mathbf{A}^T \in \Re^{d \times n}$) to highlight the squares obtained in the diagonal with the dot product because $\mathbf{A}_{ij} = \mathbf{A}_{ji}^T$
(essentially in the diagonal each line is multiplied by itself $\Rightarrow Tr(\mathbf{A}\mathbf{A}^T) = \sum_{i=1}^{n}\sum_{j=1}^{d} \mathbf{A}_{ij}^2$).

From the definition of the Frobenius norm, I have

$$||\mathbf{A}||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{d} \mathbf{A}_{ij}^2} \iff ||\mathbf{A}||_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{d} \mathbf{A}_{ij}^2 = Tr(\mathbf{A}\mathbf{A}^T)$$

1.2

$\mathbf{U} \in \Re^{n \times n}$ and $\mathbf{V} \in \Re^{d \times d}$ are orthogonal matrices therefore their columns and rows are orthonormal vectors

$$\mathbf{u}_i^T \mathbf{u}_j = \begin{cases} 1 & if\ i = j \\ 0 & otherwise \end{cases} \Rightarrow \mathbf{U}^T\mathbf{U} = \mathbf{I}_n$$

$$\mathbf{v}_i^T \mathbf{v}_j = \begin{cases} 1 & if\ i = j \\ 0 & otherwise \end{cases} \Rightarrow \mathbf{V}^T\mathbf{V} = \mathbf{I}_d$$

From the proof of 1.1, I have

$$||\mathbf{U}\ \mathbf{A}\ \mathbf{V}||_F^2 = Tr((\mathbf{U}\ \mathbf{A}\ \mathbf{V})^T\mathbf{U}\ \mathbf{A}\ \mathbf{V}) = Tr(\mathbf{V}^T\mathbf{A}^T\mathbf{U}^T\mathbf{U}\ \mathbf{A}\ \mathbf{V}) = Tr(\mathbf{A}^T\mathbf{I}_n\ \mathbf{A}\ \mathbf{V}\ \mathbf{V}^T) = Tr(\mathbf{A}^T\mathbf{A}\ \mathbf{I}_d) =$$
$$= Tr(\mathbf{A}^T\mathbf{A}) = ||\mathbf{A}||_F^2 \iff ||\mathbf{U}\ \mathbf{A}\ \mathbf{V}||_F = ||\mathbf{A}||_F$$

1.3

Given $\mathbf{U} \in \Re^{n \times n}$ and $\mathbf{V} \in \Re^{d \times d}$ orthogonal matrices, from 1.2 and the Singular Value Decomposition (recalling that:

- we call $\mathbf{u} \in \Re^n$ left singular vector, $\mathbf{v} \in \Re^d$ right singular vector and σ singular value if $\mathbf{A}\mathbf{v} = \sigma\mathbf{u}$ and $\mathbf{A}^T\mathbf{u} = \sigma\mathbf{v}$.
- if $\mathbf{A} \in \Re^{n \times d}$ has rank $r \leq n, d$, then there exist r non-zero singular values),

I have

$$||\mathbf{U}\ \mathbf{A}\ \mathbf{V}^T||_F^2 = Tr((\mathbf{U}\ \mathbf{A}\ \mathbf{V}^T)^T\mathbf{U}\ \mathbf{A}\ \mathbf{V}) = Tr(\mathbf{V}\ \mathbf{A}^T\mathbf{U}^T\mathbf{U}\ \mathbf{A}\ \mathbf{V}) = Tr(\mathbf{V}\ \mathbf{A}^T\mathbf{U}\ \mathbf{U}^T\mathbf{A}\ \mathbf{V}) =$$

$$= Tr(\mathbf{V}(\mathbf{\Sigma}\ \mathbf{V}\ \mathbf{V}^T\mathbf{\Sigma}^T)\ \mathbf{V}^T) = Tr((\mathbf{\Sigma}\ \mathbf{I}_d\ \mathbf{\Sigma}^T)\ \mathbf{I}_d) =$$

$$= Tr(\mathbf{\Sigma}\ \mathbf{\Sigma}^T) = ||\mathbf{\Sigma}||_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{d}\mathbf{\Sigma}_{ij} = \sum_{i=1}^{r}\sigma_i^2 = ||\mathbf{A}||_F^2 \iff ||\mathbf{U}\ \mathbf{A}\ \mathbf{V}||_F = ||\mathbf{A}||_F$$

2 Problem 2

2.1

2.1.1

$rank(\mathbf{L}) = rank(\mathbf{R}) = k-1, rank(\mathbf{D}) = k, rank(\mathbf{F}) \leq min\{k-d, k\} \Rightarrow rank(\mathbf{Y}) \leq k$

2.1.2

$$\mathbf{A} - \mathbf{X} = \mathbf{U} \begin{bmatrix} \mathbf{L} + \mathbf{E} - \mathbf{D} + \mathbf{R} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{bmatrix} \mathbf{V}^T \quad , \quad \mathbf{A} - \mathbf{Y} = \mathbf{U} \begin{bmatrix} \mathbf{E} - \mathbf{D} & \mathbf{0} \\ \mathbf{G} & \mathbf{H} \end{bmatrix} \mathbf{V}^T$$

$$||\mathbf{A} - \mathbf{X}||_F^2 = ||\mathbf{A} - \mathbf{Y}||_F^2 + ||\mathbf{L}||_F^2 + ||\mathbf{R}||_F^2 + ||\mathbf{F}||_F^2 =$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{d} (\mathbf{A}_{ij} - \mathbf{Y}_{ij})^2 + \sum_{i=1}^{k} \sum_{j=1}^{k} (\mathbf{L}_{ij} + \mathbf{R}_{ij})^2 + \sum_{i=1}^{k} \sum_{j=1}^{d-k} \mathbf{F}_{ij}^2 > \sum_{i=1}^{n} \sum_{j=1}^{d} (\mathbf{A}_{ij} - \mathbf{Y}_{ij})^2$$

Since a square number is ≥ 0, to have the Frobenius norm as small as possible, I have to put

$$\mathbf{L} = \mathbf{R} = \mathbf{F} = \mathbf{0}$$

2.1.3

Similarly, if I have

$$\mathbf{A} = \mathbf{U} \begin{bmatrix} \mathbf{L} + \mathbf{E} + \mathbf{R} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{bmatrix} \mathbf{V}^T \quad , \quad \mathbf{Y} = \mathbf{U} \begin{bmatrix} \mathbf{L} + \mathbf{D} + \mathbf{R} & \mathbf{0} \\ \mathbf{G} & \mathbf{0} \end{bmatrix} \mathbf{V}^T$$

$$\Rightarrow \mathbf{A} - \mathbf{X} = \mathbf{U} \begin{bmatrix} \mathbf{L} + \mathbf{E} - \mathbf{D} + \mathbf{R} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{bmatrix} \mathbf{V}^T \quad , \quad \mathbf{A} - \mathbf{Y} = \mathbf{U} \begin{bmatrix} \mathbf{E} - \mathbf{D} & \mathbf{F} \\ \mathbf{0} & \mathbf{H} \end{bmatrix} \mathbf{V}^T$$

$$||\mathbf{A} - \mathbf{X}||_F^2 = ||\mathbf{A} - \mathbf{Y}||_F^2 + ||\mathbf{L}||_F^2 + ||\mathbf{R}||_F^2 + ||\mathbf{G}||_F^2 =$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{d} (\mathbf{A}_{ij} - \mathbf{Y}_{ij})^2 + \sum_{i=1}^{k} \sum_{j=1}^{k} (\mathbf{L}_{ij} + \mathbf{R}_{ij})^2 + \sum_{i=1}^{n-k} \sum_{j=1}^{k} \mathbf{G}_{ij}^2 > \sum_{i=1}^{n} \sum_{j=1}^{d} (\mathbf{A}_{ij} - \mathbf{Y}_{ij})^2$$

So, to have the Frobenius norm as small as possible, I have to put $\mathbf{L} = \mathbf{R} = \mathbf{G} = \mathbf{0}$

2.2

From **2.1** $\mathbf{L} = \mathbf{R} = \mathbf{F} = \mathbf{G} = \mathbf{0}$
from **1.2** $||\mathbf{U}\ \mathbf{A}\ \mathbf{V}||_F = ||\mathbf{A}||_F$
recalling that $\mathbf{U}^T\mathbf{U} = \mathbf{I}_n$ and $\mathbf{V}^T\mathbf{V} = \mathbf{I}_d$

$$||\mathbf{A} - \mathbf{X}||_F = ||\mathbf{U}^T\mathbf{A}\ \mathbf{V} - \mathbf{U}^T\mathbf{X}\ \mathbf{V}||_F = ||\mathbf{U}^T\mathbf{U}\ \mathbf{A}^*\ \mathbf{V}^T\mathbf{V} - \mathbf{U}^T\mathbf{U}\ \mathbf{X}^*\ \mathbf{V}^T\mathbf{V}||_F = ||\mathbf{A}^* - \mathbf{X}^*||_F =$$

$$= || \begin{bmatrix} \mathbf{E} & \mathbf{0} \\ \mathbf{0} & \mathbf{H} \end{bmatrix} - \begin{bmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} ||_F = || \begin{bmatrix} \mathbf{E} - \mathbf{D} & \mathbf{0} \\ \mathbf{0} & \mathbf{H} \end{bmatrix} ||_F$$

The Singular Value Decomposition of $\mathbf{A}$ is $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}_\mathbf{A}\mathbf{V}^T = \sum_{i=1}^{r} \sigma_{Ai}\mathbf{u}_i\mathbf{v}_i^T$ where r is the rank of $\mathbf{A}$.
The Singular Value Decomposition of $\mathbf{X}$ is $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}_\mathbf{X}\mathbf{V}^T = \sum_{i=1}^{k} \sigma_{Xi}\mathbf{u}_i\mathbf{v}_i^T$ where $k \leq r$ is the rank of $\mathbf{X}$ (because $\mathbf{E}, \mathbf{D} \in \Re^{k\times k}$ so their max rank can be k but $\mathbf{H} \in \Re^{(d-k)\times(n-k)}$ has rank $min\{d-k, n-k\} \geq 0$).

From **1.3** $\quad ||\mathbf{A}||_F^2 = ||\mathbf{\Sigma}_\mathbf{A}||_F^2 = \sum_{i=1}^{r} \sigma_{Ai}^2$ and $||\mathbf{X}||_F^2 = ||\mathbf{\Sigma}_\mathbf{X}||_F^2 = \sum_{i=1}^{k} \sigma_{Xi}^2$

So,

$$||\mathbf{A} - \mathbf{X}||_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{d}(\mathbf{A}_{ij} - \mathbf{X}_{ij})^2 = \sum_{i=1}^{n}\sum_{j=1}^{d}(\sigma_{Ai} - \sigma_{Xi})^2 = \sum_{i=1}^{k}(\sigma_{Ai} - \sigma_{Xi})^2 + \sum_{i=k+1}^{r}\sigma_{Ai}^2$$

Since I want the best rank-k approximation to the matrix $\mathbf{A}$, I have to minimize the Frobenius norm

$$min||\mathbf{A} - \mathbf{X}||_F^2 = min\sum_{i=1}^{k}(\sigma_{Ai} - \sigma_{Xi})^2 + \sum_{i=k+1}^{r}\sigma_{Ai}^2 \Rightarrow \sigma_{Ai} = \sigma_{Xi} \iff \mathbf{D} = \mathbf{E} \iff \mathbf{X} = \mathbf{A}_k$$

$$\Rightarrow ||\mathbf{A} - \mathbf{A}_k||_F^2 = \sum_{i=k+1}^{r}\sigma_{Ai}^2 = ||\mathbf{H}||_F$$

3 Problem 3

The task of this exercise is about emergency awareness enhancement using tweets.
The dataset I used is the Disaster Tweets (`https://www.kaggle.com/vstepanenko/disaster-tweets`) from Kaggle which contains tweets collected from several disaster situations (volcanic eruption of Taal, coronavirus, etc).
So, given only the text of the tweet, I have to predicted whether it is a disaster-related tweet or not.

I solved it in two ways:

- I developed an **LSTM neural network** able to perform **binary text classification** in PyTorch,
- I did **Transfer Learning using BERT**

then I compared the results.

3.1 LSTM neural network

Once, the **Torch** libraries were installed, I loaded the data using a **Field** object for the column of the tweet's text and a **LabelField** object for the target one. The tweets are preprocessed with a custom tokenizer and then tokens are normalized.

Subsequently, the dataset was split into train and validation sets and vocabularies were built to be able to convert the texts into integer sequences to feed the neural net. In order to do that I exploited two pretrained and very famous word embeddings: **fastText** and **GloVe** (Global Vectors for Word Representation).
Word embedding is one of the most popular representations of document vocabulary and is able to capture the context of a word in a document, semantic and syntactic similarity, relation with other words, etc. As it sounds good for our classification task, I preferred to avoid stopwords removal (except http/https links) and stemming steps because they can lose valuable information, which would help the neural network to figure things out.

Then, I checked if the dataset is unbalanced and since it is, I calculated **class weights** in order to assign a greater weight to the minor class and rebalance the dataset. The weights of the classes can be computed in several ways:

- Proportion of samples labeled wrt the number of samples,
- Proportion of samples labeled wrt class with the max number of samples,
- Inverse of Number of Samples (INS),
- Inverse of Square Root of Number of Samples (ISNS),
- Effective Number of Samples (ENS),
- Inverse of distributions,

- ...

and based on the results, I selected the **Effective Number of Sample (ENS)** method that, instead of relying on the total number of samples present in each class, is based on the fact that as the number of samples increases, the additional benefit of a newly added data will diminish.

In order to prepare the data for proper loading, I used the **BucketIterator** class, which creates batches of input data in such a way that the entire dataset is not loaded at once.

Once the data are prepared, I defined the LSTM (Long Short-Term Memory) neural network model. LSTMs are a variant of RNNs (Recurrent Neural Networks) that are able to learn long term dependencies in the input sequences (feedback connections), that are lost by standard RNNs that are suitable for short term relationships. I developed this type of network because RNNs and LSTM are the most powerful neural networks for working with text.

The architecture of the neural network is the following:

- **Embedding layer**: this layer transforms the word into a vector (creating and embedding). It defines a look up table where each row is the embedding of a word. It converts the sequence in input to a dense vector representation.
- **LSTM layers**: layers implementing the LSTM functionalities.
- **Linear layer**: fully connected layer, in which each node is connected to all the nodes of the previous layer.
- **Pack padding**: dealing with padded sequences can be tricky. This module tells the LSTM drop the state of the padded elements, which do not bring any useful info.
- **Dropout**: randomly switch off neurons (during training) to prevent overfitting.

(embedding): Embedding(5302, 600)
(lstm): LSTM(600, 512, num_layers=7, batch_first=True, dropout=0.7, bidirectional=True)
(fc): Linear(in_features=1024, out_features=1, bias=True)
(act): Sigmoid()

Where the embedding layer is initialized with the pretrained embeddings from fastText and GloVe.
The LSTM consists of 7 internal bidirectional layers (characterized by a high dropout to reduce overfitting) whose outputs are received by a hidden fully connected layer.
The **bidirectional layers** run the inputs in two ways: one from past to future and one from future to past. In this way, when the LSTM runs backward preserves information from the future, and by using the two hidden states combined, it's able to preserve information from both past and future at any point in time.

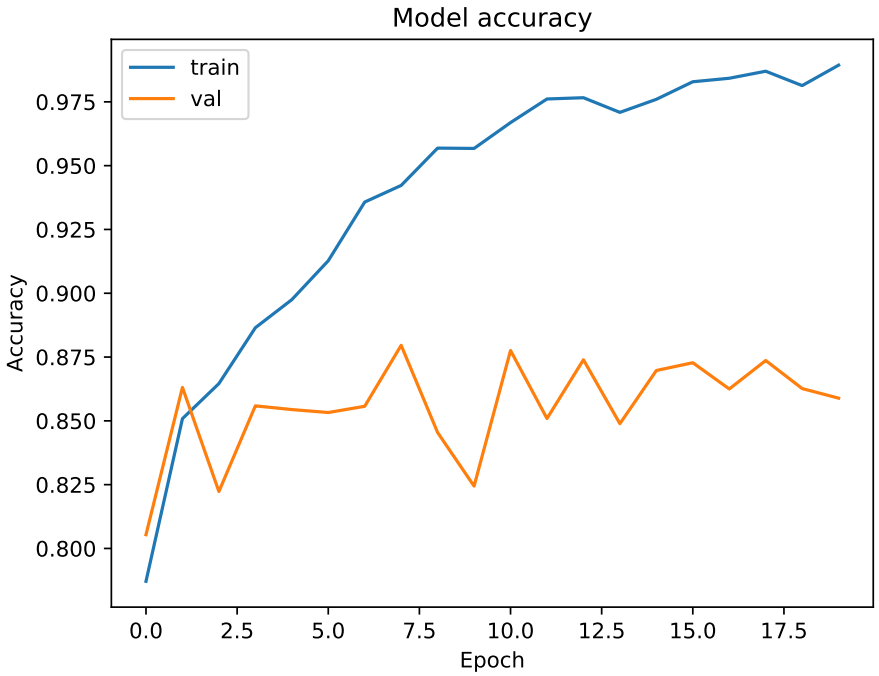
As optimizer, I used **Adam** (which consists of a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments) with a learning rate of 0.0001.

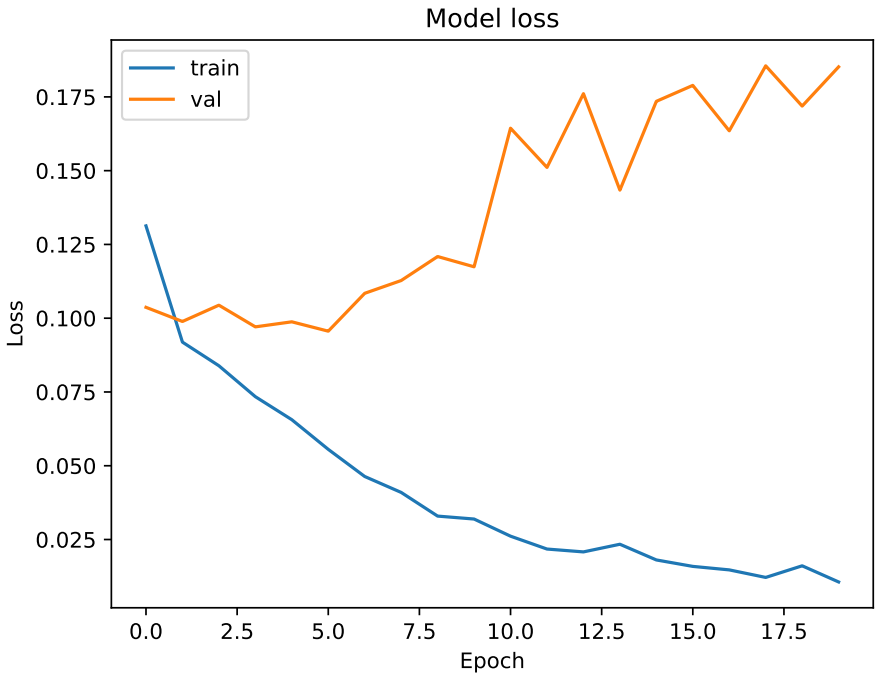
As loss function, I relied on the weighted **Binary Cross Entropy (BCE)** measure between the target and the output that is used in binary classification tasks with the **sigmoid** activation function. The weights are the class weights discussed above, so the dataset rebalancing operation is performed by the loss function.

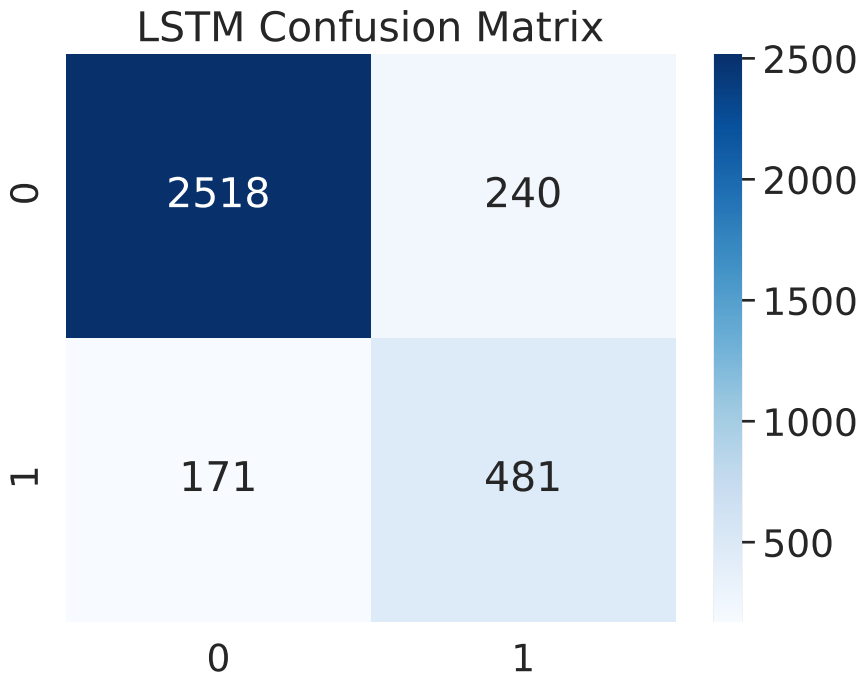
In the end, the training and evaluation phases consist of iterating through the batches evaluating the predictions via the weighted BCE loss function and the accuracy. The difference is that model weights and dropout layers are updated during training.

classes	precision	recall	f1-score	support
0	0.94	0.91	0.92	2758
1	0.67	0.74	0.70	652

	precision	recall	f1-score	support
accuracy			0.88	3410
macro avg	0.80	0.83	0.81	3410
weighted avg	0.88	0.88	0.88	3410




Model loss
train
val
0.175
0.150
0.125
0.100
0.075
0.050
0.025
Loss
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch


LSTM Confusion Matrix
2518
240
171
481
0
1
0
1
2500
2000
1500
1000
500

3.2 Transfer Learning using BERT

To implement Transfer Learning using BERT, I used the **Transformers** library that contains PyTorch, Tensorflow and Flax implementations of an impressive number of Transfer Learning models.

Google has made available a range of BERT models and as in this homework I had to investigate tweets (from different countries) to predict if a tweet is a disaster-related tweet or not and the bert-base-multilingual-uncased is not recommended (see `https://huggingface.co/transformers/pretrained_models.html`), I therefore used the **bert-base-multilingual-cased** model.

Each model comes with its own tokenizer, so I used the **BERT tokenizer** which splits texts into word pieces and downloaded the vocabulary employed during pretraining or fine-tuning a given model. So, the tokenizer will split the tweets in tokens the same way for the pretraining corpus, and it will use the same correspondence token to index (vocabulary) as during pretraining.

Once, the **Torch** and **Transformers** libraries were installed, I instantiated the pretrained BERT model with a final layer for sequence classification on top since the task is to classify tweets.

Then, I loaded the data dropping the rows with a null value in the text or target field and split the remaining data into train and test sets. So, I prepared data for BERT by presenting every document as a **BertInputItem** object, which contains all the information BERT needs:

- **A list of input IDs**: Every text has been split up into subword units, which are shared between all the languages in the multilingual model. When a word appears frequently enough in a combined corpus of all languages, it is kept intact. If it is less frequent, it is split up into subword units that do occur frequently enough across all languages. This allows our model to process every text as a sequence of strings from a finite vocabulary of limited size. The [CLS] token is added at the beginning of every document. The vector at the output of this token will be used by the BERT model for its sequence classification tasks: it serves as the input of the final, task-specific part of the neural network.
- **the input mask**: the input mask tells the model which parts of the input it should look at and which parts it should ignore. Since I made texts of a fixed length of tokens, some tweets may be cut off after this limit while other may be padded with extra tokens. In this latter case, these padding tokens will receive a mask value of 0, which means BERT should not take them into account for its classification task.
- **the segment IDs**: some NLP task take several sequences as input and the segment IDs tell BERT which sequence every token belongs to. However, in a text classification task, there's only one segment, so all the input tokens receive segment ID = 0.
- **The label ID**: the ID of the label of the document.

Then, I checked if the dataset is unbalanced and since it is, I calculated **class weights** (as in the LSTM model) in order to assign a greater weight to the minor class and rebalance the dataset.

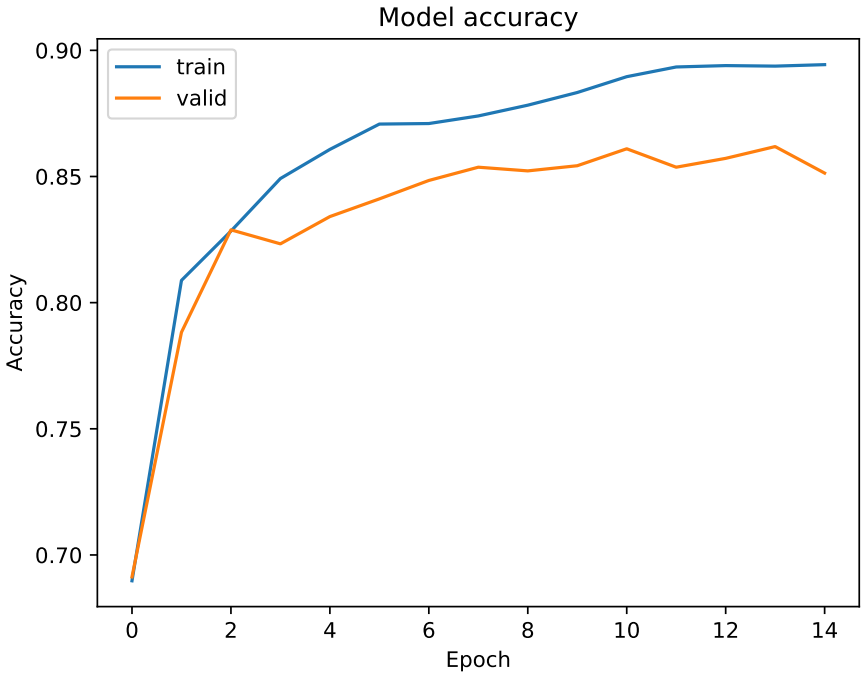
In order to prepare the data for proper loading, I used the **DataLoader** class, which creates batches of input data in such a way that the entire dataset is not loaded at once. To sample data I decided to use the **WeightedRandomSampler** to rebalance the dataset because it samples elements with given probabilities (weights), so this time class weights are not used in the loss function.

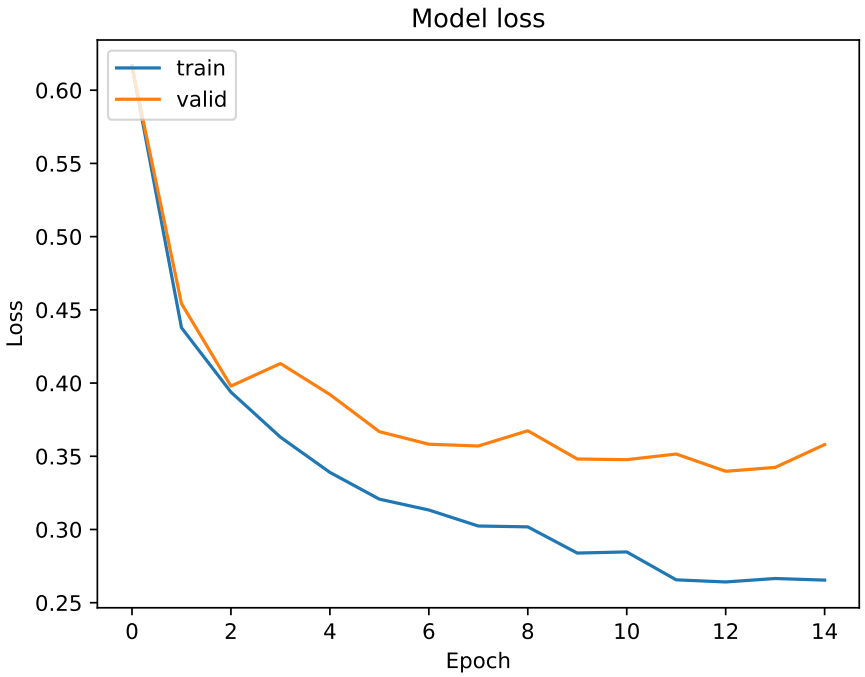
As optimizer, I used **Adam** with weight decay (which consists of a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments) with a learning rate of 0.0000005.
As loss function, I relied on the default **Cross Entropy** loss between the predictions and the passed labels.

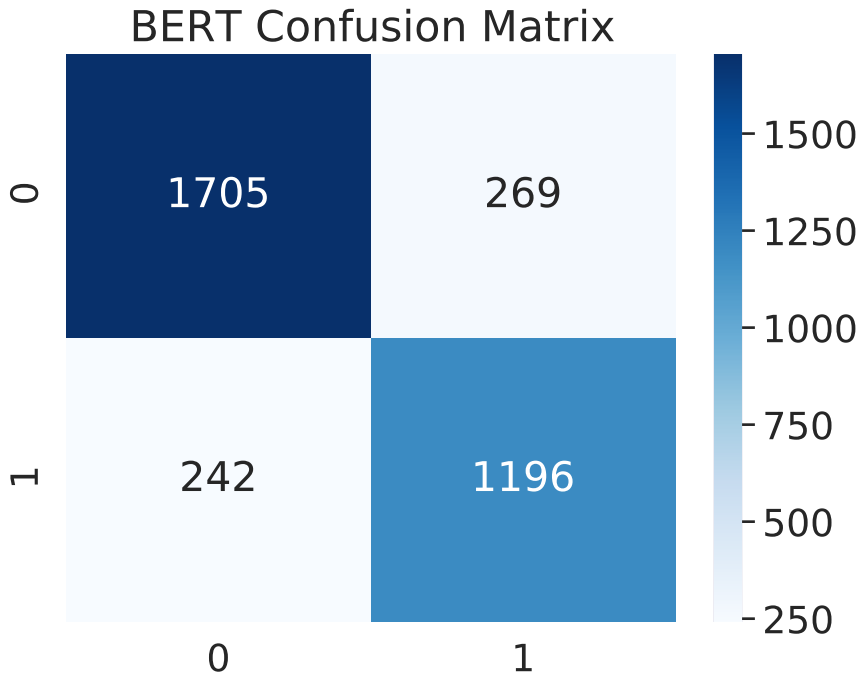
In the end, the training and evaluation phases consist of iterating through the batches evaluating the predictions via the CE loss function and the accuracy. The difference is that model weights and dropout layers are updated during training.

classes	precision	recall	f1-score	support
0	0.88	0.86	0.87	1973
1	0.82	0.83	0.82	1438

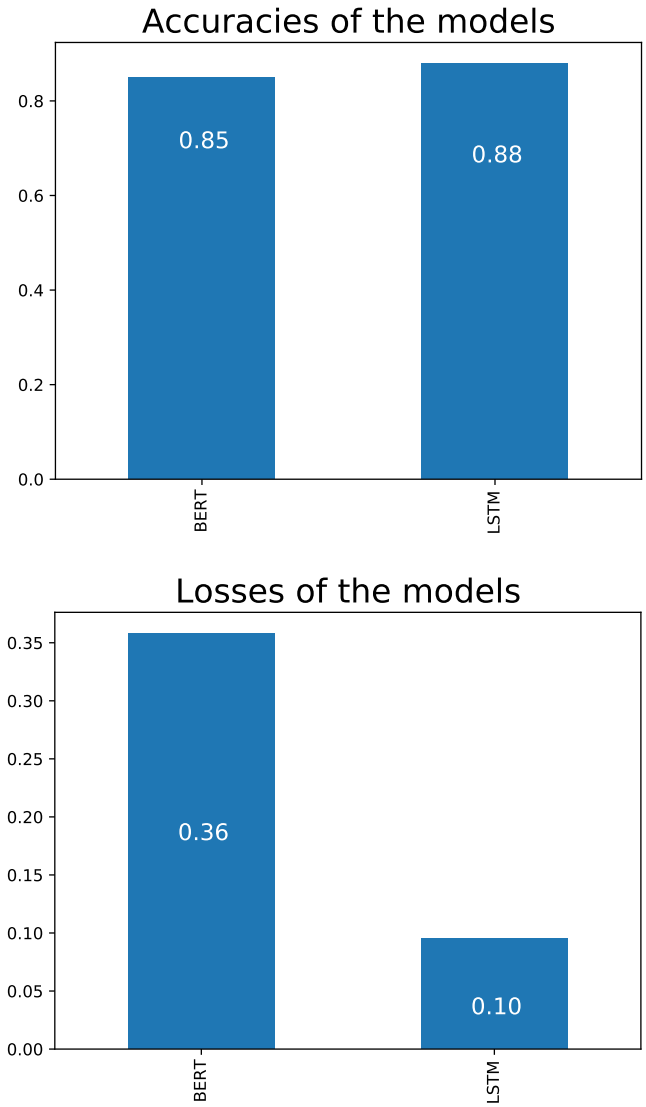
	precision	recall	f1-score	support
accuracy			0.85	3412
macro avg	0.85	0.85	0.85	3412
weighted avg	0.85	0.85	0.85	3412




Model loss
train
valid
Loss
Epoch
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0
2
4
6
8
10
12
14


BERT Confusion Matrix
1705
269
242
1196
0
1
0
1
1500
1250
1000
750
500
250

3.3 Comparison



The classic evaluation metrics show that the LSTM model has a lower loss than the BERT one while the other values are very similar.
I also noticed that BERT overfits in few epochs and much earlier than the LSTM model, so I used a lower learning rate.
Arguably, the pretrained weights are not so useful for the homework input data (tweets come from several countries), and potentially training the BERT model from scratch could produce better results. Another initial model rather than **bert-base-multilingual-cased** model or a very accurate fine-tuning can improve BERT results too. However, both models produce very good results reaching an accuracy of about 90% and precision and recall of about 80%; the confusion matrices confirm.