Jose Valero
Lucas Fayolle

Lab session week 6 - Emergency medical incidents severity assignment based on Deep learning

Profesor : Juan Miguel Garcia-Gomez

Block I. Data preparation and preprocessing

1. **Load your data using the proper column delimiter.**

```
data = read_csv(path2load, delimiter=';', encoding='cp1252')
```

2. **Use the method one_hot_encode of class OneHotEncoder to generate One Hot Encoding variables from the life-threatening label.**

```
cols2encode = ['LIFE THREATENING']
encoder = OneHotEncoder()
data = encoder.one_hot_encode(data, cols2encode)
```

3. **Invoke the prepare method to prepare your 'OBSERVATIONS' of TextPreparator class by transforming sentences to lowercase, deleting punctuation marks, tokenizing words, and mapping abbreviations.**

```
text_prep = TextPreparator()
data = text_prep.prepare(data, 'OBSERVATIONS')
```

4. **Split your data using a holdout methodology, considering 80% of the data for training and the rest for testing.**

```
data_train, data_eval = train_test_split(data, test_size=0.2,
random_state=42)
```

5. **Observe the rest of the code in and identifying the lines where next steps are performed:**

Items	Lines (in our code)
Map words to indexes	46 to 51
Extract labels	54 and 55
Convert numbers to tensors	61 to 65
Generate datasets	68 and 69
Configure batch sizes	72 and 73
Generate dataloaders	74 and 75

Block II. Model training, selection and evaluation

Using *textmodel.py*:

1. **Change the LeakyReLu activation function by a relu activation function. You may have to make a search about how a relu is defined in PyTorch.**

```
activation_1 = ReLU()
```

Using *traineval.py*:

2. **Import your preprocessed data (data loaders).**

```
from textmodel import TextModel
from dataprep import vocab_size
from dataprep import dataloader_train, dataloader_eval
```

3. **Run some experiments, changing your hyperparameters, such as learning rate, architecture or even your batch size (defined in dataprepoc.py) until you find a good configuration based on the cross entropy values and the figures obtained (you will need to complete that part too). You may need to make a search about model modules (embedding, recurrent, output) in order to set properly hyperparam values.**

   The hyperparameters to be used in the experiments are:

   1) **Default**

```
_LEARNING_RATE =
_EMBEDDING_DIMENSION = 16
_NUMBER_GRU_UNITS = 2
_NUMBER_GRU_NEURONS = 32
_MAXIMUM_EPOCHS = 150
```

   2) **Experiment 1**

```
_LEARNING_RATE = 0.00005
_EMBEDDING_DIMENSION = 10
_NUMBER_GRU_UNITS = 2
_NUMBER_GRU_NEURONS = 32
_MAXIMUM_EPOCHS = 100
```

   3) **Experiment 2**

```
_LEARNING_RATE = 0.00001
_EMBEDDING_DIMENSION = 16
_NUMBER_GRU_UNITS = 2
_NUMBER_GRU_NEURONS = 32
_MAXIMUM_EPOCHS = 150
```

4) Experiment 3

```
_LEARNING_RATE = 0.0001
_EMBEDDING_DIMENSION = 8
_NUMBER_GRU_UNITS = 2
_NUMBER_GRU_NEURONS = 32
_MAXIMUM_EPOCHS = 80
```

4. Get some metrics like the area under curve, accuracy or macro F1-score (at least) to estimate model performance in the test set.

```
auc = skmet.roc_auc_score(labs_eval, pred_eval)
accuracy = skmet.accuracy_score(labs_eval, pred_eval)
f1_score_macro = skmet.f1_score(labs_eval, pred_eval, average=
'macro')
print(auc, accuracy, f1_score_macro)
```

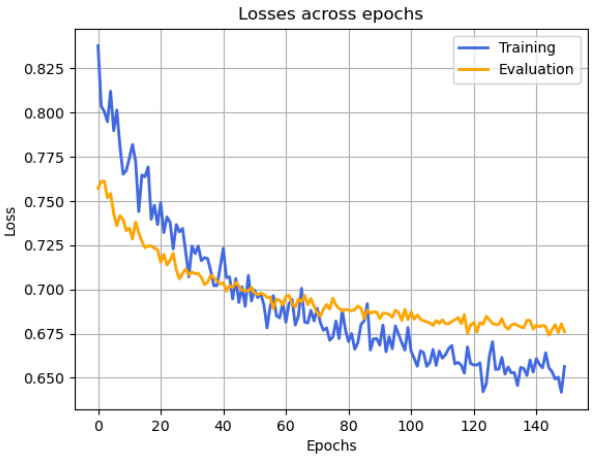
5. Results of the experiments

1. Default

The metrics of the model with the "default" parameters (those of the original file) are as follows:

- auc: 0.5641475732939147
- accuracy: 0.6311787072243346
- f1_score_macro: 0.5563690113903139

In turn, the loss over the epochs is as follows:

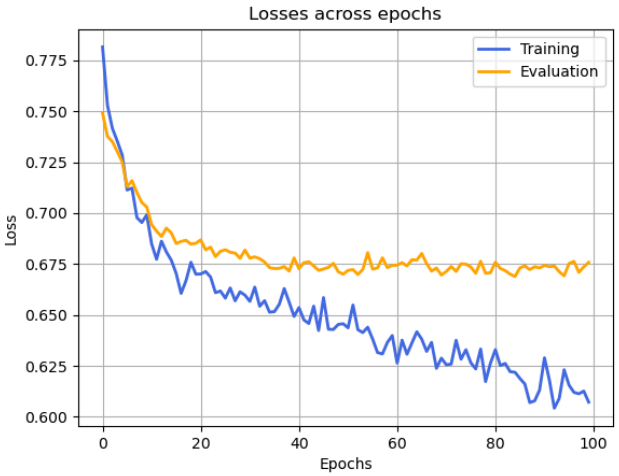


2. Experiment 1

The metrics of the model with the parameters of the experiment 1 are as follows:

- auc: 0.5806541019955654
- accuracy: 0.5893536121673004
- f1_score_macro: 0.5758064516129032

In turn, the loss over the epochs is as follows:

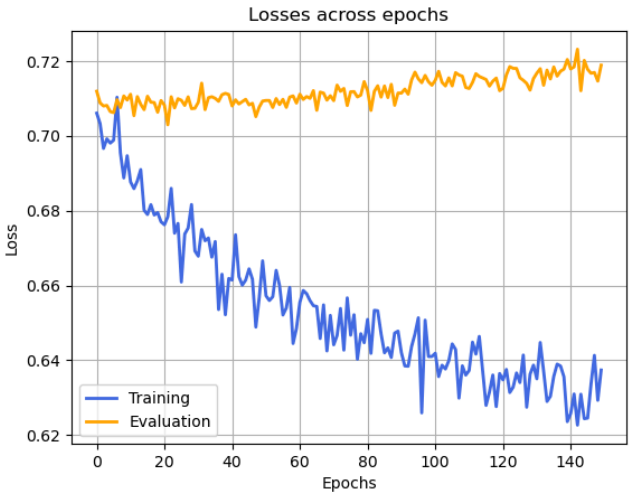


3. Experiment 2

The metrics of the model with the parameters of the experiment 2 are as follows:

- auc: 0.5375708302537571
- accuracy: 0.5855513307984791
- f1_score_macro: 0.5359299370275039

In turn, the loss over the epochs is as follows:

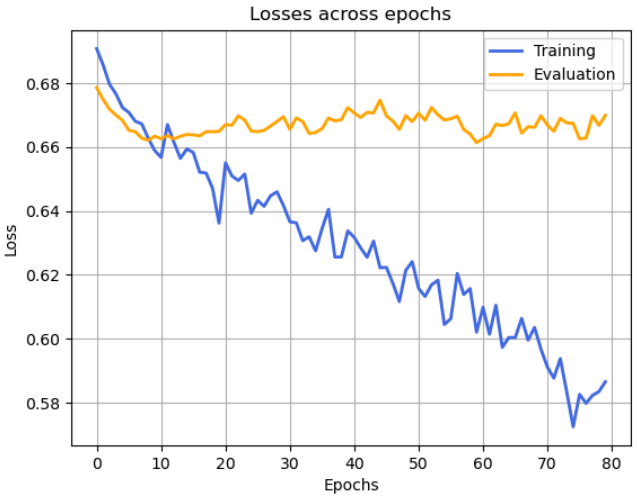


4. Experiment 3

The metrics of the model with the parameters of the experiment 2 are as follows:

- auc: 0.5829329884207932
- accuracy: 0.6121673003802282
- f1_score_macro: 0.5834989442305303

In turn, the loss over the epochs is as follows:



6. Discussion of the results

When comparing the models in terms of both metrics and loss curves, we can see significant differences in their behavior across the experiments:

- **Metrics**

  The **default configuration** yields an accuracy of 0.6311, which is the highest among the experiments, but its AUC (0.5641) and F1-macro (0.5563) are relatively low. This suggests that while the model has a good overall ability to predict correctly, its ability to balance the classification between classes is limited. In other words, although it predicts well at a global level, it struggles with maintaining class balance, which could be problematic in scenarios where balanced classification is crucial.

  In **Experiment 1**, there is a slight improvement in AUC (0.5806) and F1-macro (0.5758) compared to the default configuration. This indicates that the model achieves better class balance in this experiment, but at the expense of overall accuracy, which drops to 0.5893. While accuracy is lower, the higher F1-macro suggests that the model is better at handling imbalanced class predictions, which might be preferable in certain situations.

  In contrast, **Experiment 2** shows the worst performance across all metrics, with an AUC of 0.5375 and F1-macro of 0.5359, along with an accuracy of 0.5855. These results suggest that the model is struggling to balance class predictions effectively, and its overall predictive ability is quite limited.

  Lastly, **Experiment 3** demonstrates the best overall performance. With an AUC of 0.5829 and F1-macro of 0.5834, this experiment shows the most balanced classification performance across the classes. Although its accuracy (0.6121) is lower than the default configuration, the model is better at classifying between the classes, making it the most effective and balanced setup for this task. The improvement in both AUC and F1-macro suggests that the model is more robust in dealing with class imbalances.

- **Loss**

  In terms of loss, the **default configuration** shows a steady decrease in both training and evaluation loss, indicating that the model is learning effectively. However, the loss values do not reach particularly low levels, which aligns with the moderate AUC and F1-macro scores. There are no clear signs of overfitting, but the values indicate only average performance.

  In **Experiment 1**, the evaluation loss stabilizes around epoch 40, while the training loss continues to decrease. This suggests that the model is continuing to learn from the training data but its ability to generalize on the evaluation set is limited after a certain point.

  Experiment 2 shows a problematic loss curve, with evaluation loss stalling early and even increasing after a point, which is a clear sign of overfitting. Meanwhile, the training loss continues to decrease, indicating that the model is overfitting the training data without improving its generalization capabilities.

  On the other hand, **Experiment 3** displays a more favorable trend. Training loss decreases steadily, while evaluation loss stabilizes without rising, indicating a good balance between learning and generalization. This aligns with the AUC and F1-macro metrics, which are the highest among all experiments, showing that the model is effectively classifying the data in a more balanced way.

In conclusion, **Experiment 3** delivers the best overall performance, with the highest metrics in terms of AUC and F1-macro, and stable loss curves that suggest a well-balanced model.