Tunis El Manar University

National Engineering School of Tunis

Information and Communication Technologies Departement

Deep Learning Report :

Textual data sentiment analysis

Elaborated by :

Chaima JELJLI

Houssem KHALFI

Class:

3ATEL2-DASEC

Academic Year : **2022-2023**

1. Installing and importing the packages needed.

2. Data:

a. Loading the data and checking the data distribution.

We loaded the data from our dataset which is sentiment140. It contains 1,600,000 tweets extracted using the twitter api. Those tweets have been annotated (0 = negative, 4 = positive) and they can be used to detect sentiment .

It contains the following 6 fields:

- sentiment: the polarity of the tweet (0 = negative, 4 = positive)
- ids: The id of the tweet (2087)
- date: the date of the tweet (Sat May 16 23:58:44 UTC 2009)
- flag: The query (lyx). If there is no query, then this value is NO_QUERY.
- user: the user that tweeted (robotickilldozr)
- text: the text of the tweet (Lyx is cool)

In our case, we only need the columns of sentiment and text from the dataset.
Then we plotted the distribution to see if we have an equal number of positive and negative tweets or not. They were both equal so our dataset is not skewed.

b. Data preprocessing:

we did a text preprocessing:

1. **Lower Casing:** Each text is converted to lowercase.
2. **Replacing URLs:** Links starting with 'http' or 'https' or 'www' are replaced by '<url>'.
3. **Replacing Usernames:** Replace @Usernames with word '<user>'.
4. **Replacing Consecutive letters:** 3 or more consecutive letters are replaced by 2 letters.
5. **Replacing Emojis:** Replace emojis by using a regex expression. [eg: ':)' to '<smile>']
6. **Replacing Contractions:** Replacing contractions with their meanings. [eg: "can't" to 'can not']

   We have a CSV file that contains the contractions and their replacement.

7. **Removing Non-Alphabets**: Replacing characters except Digits, Alphabets and pre-defined Symbols with a space.

c. Split training and validation.

1. **Training Data:** The dataset upon which the model would be trained on contains 95% data.

2. **Test Data:** The dataset upon which the model would be tested against contains 5% data.

3. Model:

a. Our model

We are going to use a sequential model.

1. **Embedding Layer:** We're using the predefined layer from Tensorflow in our model.

Arguments :

- **input_dim:** Size of the vocabulary.
- **output_dim:** Dimension of the dense embedding.
- **weights:** Initializes the embedding matrix**.**
- **trainable:** Specifies whether the layer is trainable or not.

2. **Bidirectional:** Bidirectional wrapper for RNNs. It means the context is carried from both left to right and right to left in the wrapped RNN layer.
3. **LSTM:** Long Short Term Memory, it's a variant of RNN which has a memory state cell to learn the context of words which is further along the text to carry contextual meaning rather than just neighboring words as in case of RNN.

Arguments :

- **units:** Positive integer, dimensionality of the output space.
- **dropout:** Fraction of the units to drop for the linear transformation of the inputs.
- **return_sequence:** Whether to return the last output in the output sequence, or the full sequence.

4. **Conv1D:** This layer creates a convolution kernel that is convolved with the layer input over a single dimension to produce a tensor of outputs.

Arguments :

- **filters:** The dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size:** Specifies the length of the 1D convolution window.
- **activation:** Specifies the activation function to use.

5. **GlobalMaxPool1D:** Downsamples the input representation by taking the maximum value over the different dimensions.
6. **Dense:** Dense layer adds a fully connected layer in the model. The argument passed specifies the number of nodes in that layer.

Our last dense layer has the activation "Sigmoid", which is used to transform the input to a number between 0 and 1. Sigmoid activations are generally used when we have 2 categories to output in.

```
def getModel():
    embedding_layer = Embedding(input_dim = vocab_length,
                                output_dim = Embedding_dimensions,
                                weights=[embedding_matrix],
                                input_length=input_length,
                                trainable=False)

    model = Sequential([
        embedding_layer,
        Bidirectional(LSTM(100, dropout=0.3, return_sequences=True)),
        Bidirectional(LSTM(100, dropout=0.3, return_sequences=True)),
        Conv1D(100, 5, activation='relu'),
        GlobalMaxPool1D(),
        Dense(16, activation='relu'),
        Dense(1, activation='sigmoid'),
    ],
    name="Sentiment_Model")
    return model
```

b. Set the optimizers and tune the hyperparameters related to them.

```
training_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

c. Set the metrics and the loss function.

```
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

callbacks = [ReduceLROnPlateau(monitor='val_loss', patience=5, cooldown=0),
             EarlyStopping(monitor='val_accuracy', min_delta=1e-4, patience=5)]
```

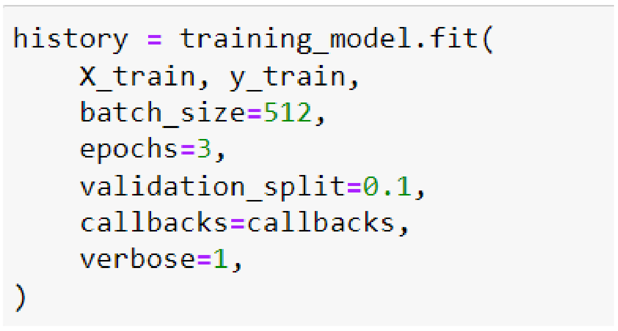
4. Training:

During the training, we used callbacks:ReduceLROnPlateau and EarlyStopping

-**ReduceLROnPlateau:** Reduces Learning Rate whenever the gain in performance metric specified stops improving.

-**EarlyStopping:** Stop training when a monitored metric has stopped improving.

We started with these hyperparameters while fitting the model:

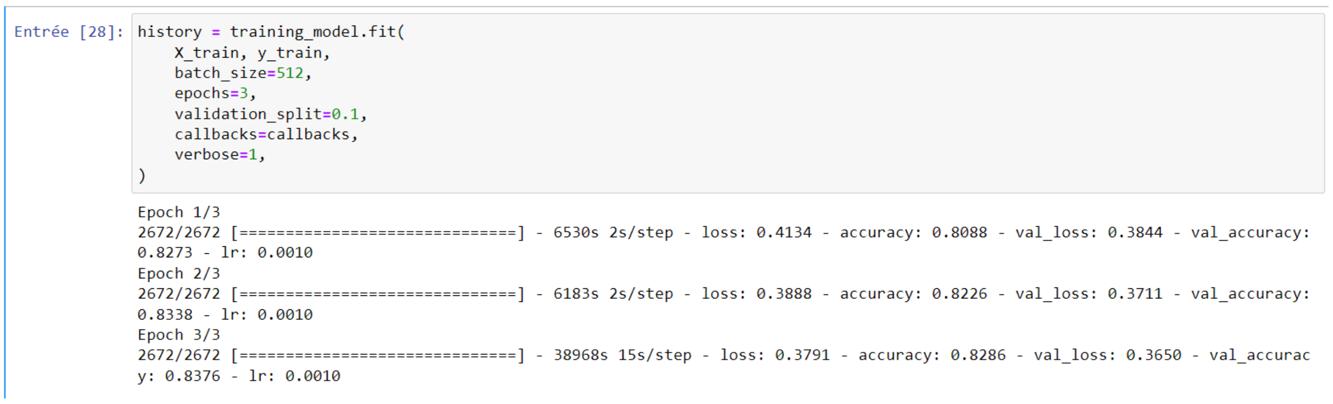


```
history = training_model.fit(
    X_train, y_train,
    batch_size=512,
    epochs=3,
    validation_split=0.1,
    callbacks=callbacks,
    verbose=1,
)
```

-The batch size : is a number of samples processed before the model is updated.
-Number of epochs: is the number of complete passes through the training dataset.
-validation_split

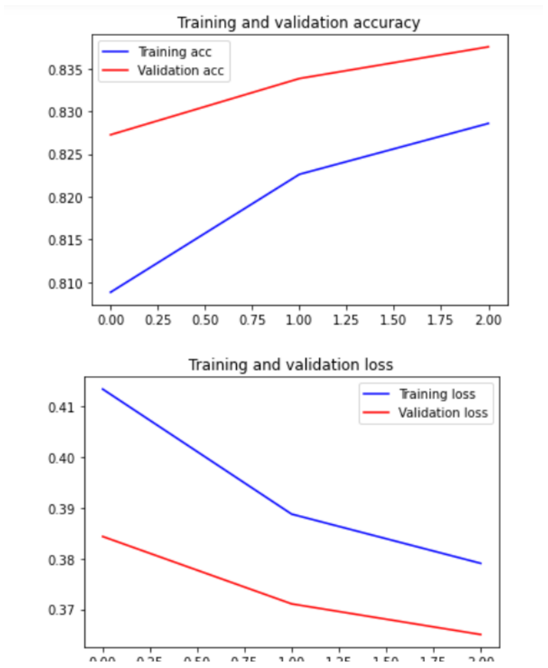
5. Model evaluation:

-We evaluated our model with accuracy as a metric

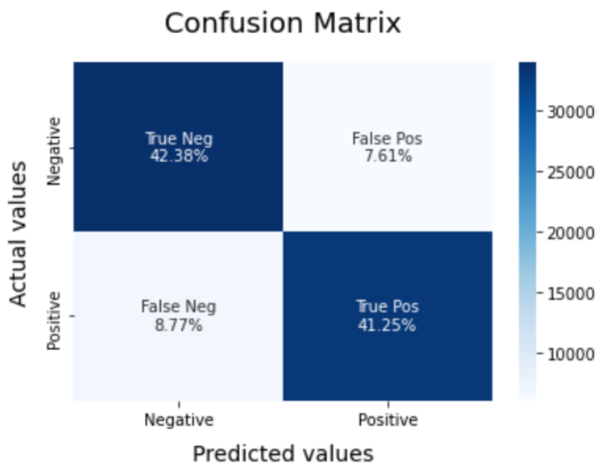


```
Entrée [28]: history = training_model.fit(
                 X_train, y_train,
                 batch_size=512,
                 epochs=3,
                 validation_split=0.1,
                 callbacks=callbacks,
                 verbose=1,
             )

             Epoch 1/3
             2672/2672 [==============================] - 6530s 2s/step - loss: 0.4134 - accuracy: 0.8088 - val_loss: 0.3844 - val_accuracy:
             0.8273 - lr: 0.0010
             Epoch 2/3
             2672/2672 [==============================] - 6183s 2s/step - loss: 0.3888 - accuracy: 0.8226 - val_loss: 0.3711 - val_accuracy:
             0.8338 - lr: 0.0010
             Epoch 3/3
             2672/2672 [==============================] - 38968s 15s/step - loss: 0.3791 - accuracy: 0.8286 - val_loss: 0.3650 - val_accurac
             y: 0.8376 - lr: 0.0010
```

We reached 0.8286 as a training accuracy and 0.837 as a validation accuracy
-Then we plotted the train and validation loss and accuracy curves



-We plotted the confusion metrics also to understand how our model is performing on both classification types.



-Then we evaluated our model with other metrics : precision, recall and F1-score

```
Entrée [32]: # Print the evaluation metrics for the dataset.
             print(classification_report(y_test, y_pred))
```

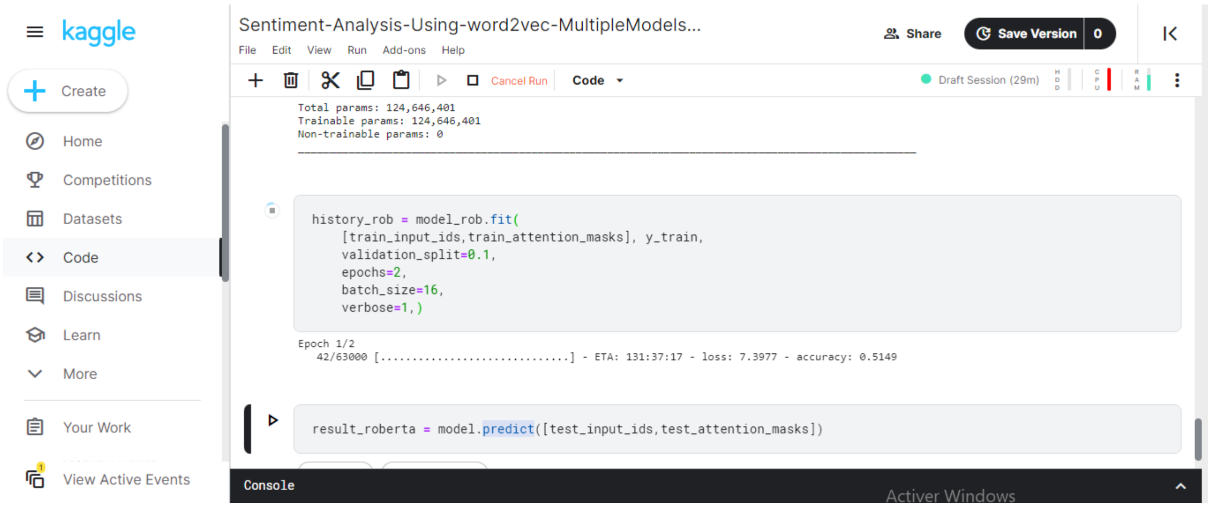
```
              precision    recall  f1-score   support

           0       0.83      0.85      0.84     39989
           1       0.84      0.82      0.83     40011

    accuracy                           0.84     80000
   macro avg       0.84      0.84      0.84     80000
weighted avg       0.84      0.84      0.84     80000
```

6. Pretrained model:

In our case, we used RoBERTa (Robustly Optimized BERT Pre-training Approach) which is a variant of the BERT (Bidirectional Encoder Representations from Transformers) with changes in the pretraining procedure.
It's a transformer-based language model that uses self-attention to process input sequences and generate contextualized representations of words in a sentence.

Training of the model :

- Epochs = 4
- Batch Size = 30



```
Total params: 124,646,401
Trainable params: 124,646,401
Non-trainable params: 0
```

```
history_rob = model_rob.fit(
    [train_input_ids,train_attention_masks], y_train,
    validation_split=0.1,
    epochs=2,
    batch_size=16,
    verbose=1,)
```

```
Epoch 1/2
   42/63000 [..............................] - ETA: 131:37:17 - loss: 7.3977 - accuracy: 0.5149
```

```
result_roberta = model.predict([test_input_ids,test_attention_masks])
```

7. Improvements:

- From the training curve we can conclude that our model doesn't have bias nor is it overfitting. The accuracy curve has flattened but is still rising, which means training for more epochs can yield better results.

- We tried to improve our model by tuning its hyperparameters