PROJECT: SUPERVISED LEARNING ON YOUR OWN

A text classification work on the goemotion dataset

FAUSSURIER Yann

EPF |

Introduction :

In this report we will analyze the progression of a classification model on the Goemotions dataset.
The dataset contains 58 000 reddit commentary, carefully labeled with one emotion among the 27 following:
'confusion’, 'curiosity', 'desire', 'disappointment', 'disapproval', 'disgust', 'embarrassment', 'excitement', 'fear', 'gratitude', 'grief', 'joy', 'love', 'nervousness', 'neutral', 'optimism', 'pride', 'realization', 'relief', 'remorse', 'sadness','surprise', ‘admiration’, ‘amusement’, ‘anger’, ‘annoyance’, ‘approval’, ‘caring’

The dataset also contains the text of the reddit comment, which will be our input data, the objective of the classification problem will be to create a model which, given an input comment, is able the label the most relevant emotion(s) associated to the text.
Here is an example of a sample of the data given in the abstract of the project:

Sample Text	**Label(s)**
OMG, yep!!! That is the final answer. Thank you so much!	gratitude, approval
I’m not even sure what it is, why do people hate it	confusion
Guilty of doing this tbph	remorse
This caught me off guard for real. I’m actually off my bed laughing	surprise, amusement
I tried to send this to a friend but [NAME] knocked it away.	disappointment

The data also contains additional information on the comment like from which subreddit it came from, if it is an unclear example for the emotion it conveys. Some data can have up to 4 labels attributed to them, we will see that this information is essential to having a performant model.
The data was scrapped in 2020 by Dorottya Demszky, Dana Movshovitz-Attias, Jeongwoo Ko, Alan Cowen, Gaurav Nemade and Sujith Ravi.

The objective of this work is to obtain the best model, with the highest accuracy when tested on a test set, we will see what methodology we used to get to a good result. This report will be structured iteratively, which mean we will first analyze the first model we used and how we conceived it and its result. We will progressively show how did we manage to find better solution to improve our results.

The first model

For this first model, we decided to do single label classification, which mean that we will attribute only one emotion to a text, this decision was not the best now that look it back. But that is because when looking naively at the data, you would believe that every text had only one emotion attributed to them, because only one Boolean was set to 1, and I hadn't fully read the GoEmotions paper.
But the truth is that some reddit comments appear multiple times in the dataset, some of them can appear up to 4 times (which mean that there is 4 lines with an identical text), but they all have different emotions attributed to them.

This caused problem because during experimentations, I tried to use a softmax activation function and a categoricalCrossedEntropy for the model to associate only one emotion to a text.
But because a text would appear multiple times, the categorical classification would not work, and the loss would explode.
The sigmoid does not have problems with duplicate, that is why it is choosen here

The model:

For the first model used in this project I've decided to start with a simple sequential network
The model will consist of six layers, one input embedding layer, which is a keras layer mainly used in NLP cases, this layer will be followed by a flatten layer and a dropout layer.
We then add a dense layer with 35 units and a reLu activation function and we finish by adding another dropout layer and an output Dense layer with 28 units (corresponding to the 28 emotions).

We use the optimizer 'adam' and we will evaluate the fiability of this model with the accuracy.
Furthermore, we will add an early stopping callback which will monitor the loss, when the loss ceases to decrease, the model will stop training.

The preprocessing of the data:

For the model to correctly interpret input text data, we need to process it, to do so, we will use the tokenizer method which will segregate a given text into small chunks or tokens.
After fitting the tokenizer on the training sentences, we use the method text_to_sequence to convert every text to a sequence of integer.
This allow the text input to be much more understandable by the model.
Finally, we use the method pad sequence to make each sequence the same length.
We apply both this processing to the train set and the test set.

The hyperparameters:

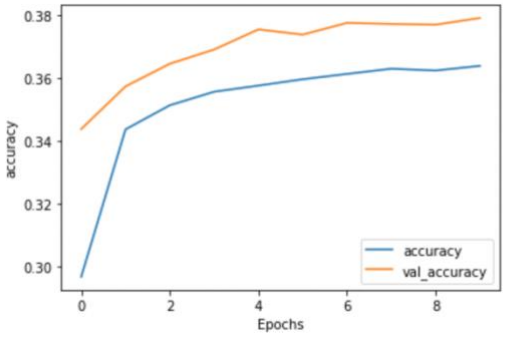
The hyperparameters were defined as follow:

```
vocab_size = 1000
embedding_dim = 16
max_length = 80
trunc_type='post'
padding_type='post'
oov_tok = "<OOV>"
training_size = 190000
```

Because the length of our dataset is 211225, we set training size to 190 000, allowing 21 225 for the test set, which seems like a good repartition.

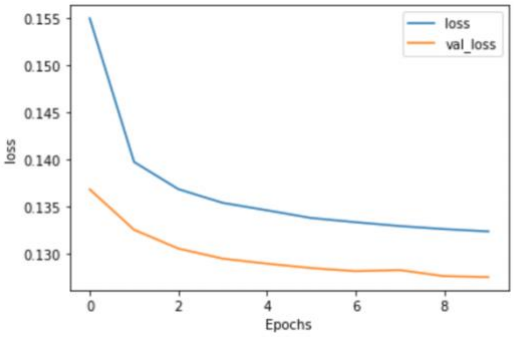
During our NLP class, we used the same hyperparameters, I decided to inspire myself from this work to set vocab_size and embedding_dim
Max length was set to 80 after numerous trial and error, it was found that it gave the best results.

The results:



The result gave us an accuracy of 37.5%.

This is an ok result but we can do much better
And we will see how with the modification of
The model and the next model.




```
Epoch 8/10
5938/5938 - 5s - loss: 0.1331 - accuracy: 0.3617 - val_loss: 0.1283 - val_ac
curacy: 0.3783 - 5s/epoch - 914us/step
Epoch 9/10
5938/5938 - 5s - loss: 0.1328 - accuracy: 0.3625 - val_loss: 0.1280 - val_ac
curacy: 0.3749 - 5s/epoch - 918us/step
Epoch 10/10
5938/5938 - 5s - loss: 0.1327 - accuracy: 0.3624 - val_loss: 0.1280 - val_ac
curacy: 0.3764 - 5s/epoch - 879us/step
```

An alternative version of the model:

The model above work properly but I was curious about testing something, what if I reduced the number of emotions by grouping them into some emotions that are more general, would the model work better?
This idea was induced by iterating through the data and thinking to myself that some emotions attributed to some text were blurry, and could be replaced easily with another emotion.

I thought that maybe 27 emotions were to many and that some text could simply not be classified with such precise emotions, and maybe the model would work better with much simpler emotions.
To do so I used almost the same model as above.

I reduced the 27 emotions to just 5 (Happy, Fear or anger, shock, love, neutral and sadness),

```
df['Happy']=np.where((df['optimism']==1) | (df['admiration']==1) | (df['excitement']==1) | (df['approva

df['Fear or anger']=np.where((df['anger']==1) | (df['annoyance']==1) | (df['disapproval']==1) | (df['fe

df['Shock']=np.where((df['confusion']==1) | (df['disappointment']==1) | (df['disgust']==1) | (df['embar

df['Love']=np.where((df['love']==1) | (df['desire']==1) | (df['caring']==1),1,0)

df['Neutral']=np.where((df['neutral']==1) | (df['realization']==1) | (df['disgust']==1) | (df['embarras

df['Sadness']=np.where((df['sadness']==1) | (df['remorse']==1) | (df['grief']==1),1,0)
```

The only following step was to do the same preprocessing and changing the number of units for the output dense layer of our model.

By doing so we obtain an accuracy of 50%, and I think that could have been much more with a more conscious grouping of the emotions, but because the goal of this work is to classify as best as possible with the 27 emotions, I decided not to continue exploring this path.

However it is interesting to see this result, and makes us think that it is hard to convey every sentiment of a tweet with 27 complex emotions.

The second and final model

Introduction

After making the first model, I was conscious that the next step for this problem was to change the model and consider the fact that some tweets had multiple emotions attributed to them, and thus that we needed to classify each text with multiple emotions.

The model

For this model, I decided to use a long short-term memory (LSTM) network, we also studied this kind of network in our last class of NLP, LSTMs layers have feedback connections which allows them to process entire sequence of data instead of iterating through each point.

This make LSTMs very good at processing certain type of data, and they are known to be particularly effective when dealing with text data.

The model will consist of our embedding input layer like before, we will keep the same preprocessing that we used with the tokenizer so we will keep this layer as input.
Then we add two Bidirectional LSTM layer. With respectively 150 and 75 units and the parameter return_sequence set to true for the first one,
After that we can directly set our output dense layer with 28 neurons, a sigmoid activation function.
We then again set an early stopping that monitor the loss. We use the optimizer rmsprop, which is an alternative to the adam optimizer that is known to accelerate the overall learning process. This will come in handy because LSTM models takes a lot of time to train.

Finally, we will use the precision as a matric here, because it is apparently a commonly used metric when it comes to multi-label classification

The preprocessing of the data:

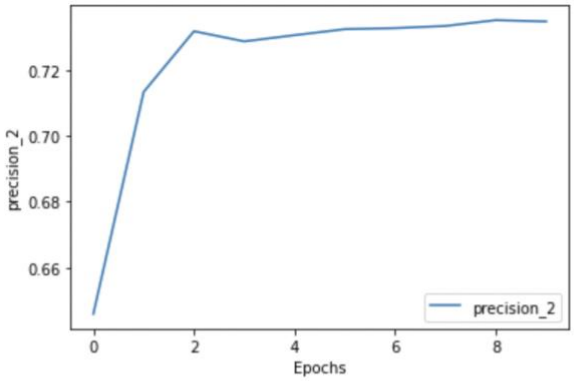
In order to go into the multilabel territory, we need to take every duplicate and merge into one line which have multiple Boolean set to 1.

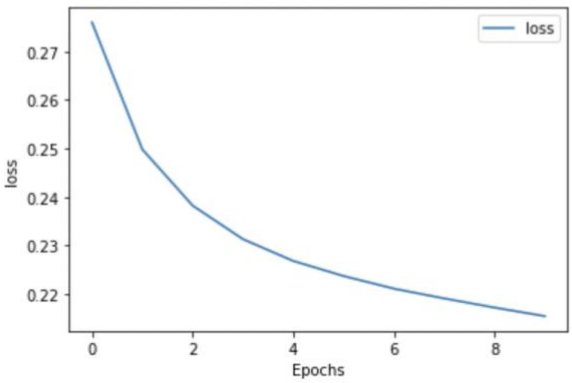
To do that we make take all emotions and we group them by text, afterward we apply a sum so that every duplicate adds themselves into one line:

Some duplicates are the same in the sense that they attribute the same emotion to the same text. This create a problem because our previous command is a sum and certain emotion Boolean get set to 2 or 3 instead of 0 and 1.
To palliate this problem, we change every emotion that have a number above 1 to 1.

The results:

This model offers a much better precision, 73,48% after 10 epochs, this could probably Improve even more as the earlyStopping didn't Stop the training, but because it took the Model 2 hours to get to 10 epochs, we will Consider it an acceptable result






```
Epoch 9/10
1805/1805 - 822s - loss: 0.2172 - precision_2: 0.7352 - 822s/epoch - 455ms/step
Epoch 10/10
1805/1805 - 830s - loss: 0.2154 - precision_2: 0.7348 - 830s/epoch - 460ms/step
```

Conclusion

To conclude, we can say that so far, the LSTM model gives the better result and that the problem is adapted to a multi-label solution.

I believe we could furthermore improve this model by tweaking the hyper parameters and maybe adding additional layers, we could use gridSearchCV to improve our last model, but the computing time would probably be insane

One thing that I could have done but realized a bit too late was that I could exclude all sample that have the “example_very_unclear” set to true from the train set. These samples could induce the model into a poorer training.

Lastly, I believe that there is deeper work to be done concerning the metrics, maybe there are metrics that I didn’t use that were a perfect fit for these problems