EPITA
NLP
Graded Project correction

Question 1: the best results have been reached when some specific pre-processing steps have been used on the dataset.
A way to do this could be to use the spaCy english model 'en_core_web_md' and, from it, you could proceed to lemmatization and stop word removal. Then, since neural networks deal with numbers and not strings, you could use TF-IDF to transform your dataset into relevant numerical vectors. From here, as one group of student did, you could use this kind of network

```
model = Sequential()
model.add(Dense(4096, activation='selu', kernel_initializer='lecun_normal',
input_shape=(X_train_.shape[1],), kernel_regularizer=tf.keras.regularizers.l2(0.01)))
model.add(Dense(2048, activation='selu', kernel_initializer='lecun_normal',
kernel_regularizer=tf.keras.regularizers.l2(0.01)))
model.add(Dense(1024, activation='selu', kernel_initializer='lecun_normal',
kernel_regularizer=tf.keras.regularizers.l2(0.1)))
model.add(Dense(64, activation='selu',))
model.add(Dense(6, activation='softmax'))
model.layers[-1].bias.assign(class_weights)
model.compile(optimizer='Adam', loss=tf.losses.categorical_crossentropy,
metrics=['accuracy'])
```

and get those results for each class:

	precision	recall	f1-score	support
0	0.85	0.83	0.84	275
1	0.79	0.82	0.80	224
2	0.86	0.89	0.88	695
3	0.63	0.76	0.69	159
4	0.92	0.83	0.87	581
5	0.66	0.61	0.63	66

Nota bene:

- it was important to check metrics such as *precision*, *recall* and *f1-score* on **each class** (each **feeling**) of the original dataset. An aggregated indicator only was not good enough to really understand how relevant is your model.
- there are six-category sentiments, this is the reason why there are six neurons on the last layer. This means that the labels needed to go through a one-hot encoded preprocessing step

Question 2: The idea there was just to stay close to what we saw in Course 3 exercises. For the preprocessing, you could have done:

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df_train['tweet'])
sequences = tokenizer.texts_to_sequences(df_train['tweet'])
max_length = max([len(tweet ) for tweet in df_train['tweet']])
padded = pad_sequences(sequences, maxlen=max_length, truncating="post", padding
="post")
vocab_size = len(tokenizer.word_index) + 1
```

Do not forget that instead of using *df_train["tweet"]*, you can use as well the preprocessed data computed for Question 1 (after lemmatization and stop word removal).

Again, from here you could have use a network such as

```
model = tf.keras.Sequential([
    Embedding(vocab_size, output_dim = 100, input_length=max_length),
    Bidirectional(LSTM(64)), #32
    Dropout(0.4),
    Dense(32, activation='leaky_relu', kernel_regularizer='l1_l2'),
    Dropout(0.4),
    Dense(6, activation='softmax')
])
```

For results looking like those

	precision	*recall*	*f1-score*	*support*
0	*0.94*	*0.87*	*0.90*	*275*
1	*0.85*	*0.93*	*0.89*	*224*
2	*0.93*	*0.92*	*0.93*	*695*
3	*0.76*	*0.84*	*0.80*	*159*
4	*0.96*	*0.96*	*0.96*	*581*
5	*0.63*	*0.55*	*0.59*	*66*

Question 3: Concerning the fine-tuning, there were different ways of doing it. The most natural was probably to use a BERT pre-trained model and fine-tune it.
One possibility was to take inspiration from *course4_classify_text_with_BERT_solution.ipynb* and add, and train, a classification layer taking as inputs the outputs of the BERT decoder (which corresponds to embedding values of the tweets).

You could have either use the function already available from HuggingFace to do this kind of automatic fine-tuning:

```
from transformers import TFAutoModelForSequenceClassification
model = TFAutoModelForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=6)
```

To find more details about this procedure, have a look for example at:
https://huggingface.co/docs/transformers/v4.17.0/en/tasks/sequence_classification

.
In the end, you can get this kind of results:

	precision	*recall*	*f1-score*	*support*
0	*0.89*	*0.94*	*0.91*	*275*
1	*0.85*	*0.92*	*0.88*	*224*
2	*0.92*	*0.95*	*0.94*	*695*
3	*0.87*	*0.71*	*0.78*	*159*
4	*0.96*	*0.94*	*0.95*	*581*
5	*0.81*	*0.70*	*0.75*	*66*

Question 4: the point was mainly to compare to results you got from the different models on the sentiment analysis dataset. It may obviously depend on how you had tweaked respectively each of them but, in the end, it was probably the fine-tuned one which was close to the best.

Concerning the *Data For Good* climate-change headline dataset, the point was for you to find a good model based on everything you have learned so far.

It was important to notice that this dataset is in French and is highly imbalanced (way more class 0 represented than class 1).

Since the data was in French, you cannot use the English models anymore. That means that you have to replace the spaCy 'en_core_web_md' model with the French equivalent '*fr_core_news_md*' if you had wanted to use spaCy. Concerning a pre-trained model approach, you had to to use French models such as camembert-base or flaubert_base_uncased which are French versions of BERT. You could have as well try some multi-language models like bert-base-multilingual-cased.

For example, that is the kind of approach that a Data For Good contributor used. He shared his fine tuned model on HuggingFace and you can find it there:
https://huggingface.co/pierre-loic/climate-news-articles

What he did was using a pre-trained FlauBERT model already able to make classification between topics such as “culture”, “sport”, “environment”, “economy”... and try to group topics into relevant ones and then fine-tuned it for the two classes climate-change headline situation. You can find the original pre-trained model there:
https://huggingface.co/lincoln/flaubert-mlsum-topic-classification

That is all for now, hope that helps.

Thank you for everything! :)