P0: Simple models with Keras

Goal: implement **three models** for multiclass text classification on the Women's E-commerce clothing reviews dataset, two of them simple feed-forward models using a `Tokenizer` and `TextVectorizer`, respectively, and the third a Convolutional Neural Network (CNN) using a `TextVectorizer` layer and embeddings.

Teams: one person or two.

Due date: Check virtual campus.

1. Data preparation

The first step is to downlad the dataset (a `csv` file) from *GitHub*. Suggestions:

- You can use the utility function `tensorflow.keras.utils.get_file()` to download the file. You should set an absolute path to save the file, taking into account that, in *Google Colaboratory*, you have direct acces to the folder `/content/`.
- There are many ways to load a `csv` in memory. One simple way is to use `csv.reader()`.

```
with open(path, newline='') as f:
    reader = csv.reader(f)
    data = list(reader)
```

The resulting data estructure (`data`) is a python list of lists (the reviews).

```
Empieza a programar o a crear código con IA.
```

```
Empieza a programar o a crear código con IA.
```

Once you have the rows of the `csv` file in a data structure (remember that the first one is the names of the attributes of the data set, and must be discarded) you have to preprocess the data for its use as an input to the neural networks:

1. Extract the textual data from the rows, included in the fields `Title` and `Review Text`, and join both fields if title is not empty.
2. Convert the field `Rating`, whose content are integers in the interval [1,5] into three classes: negative (ratings 1,2), neutral (rating 3) and positive (ratings 4,5).
3. The dataset contains about 23,000 reviwes. Reserve the first 18,000 for training, and the rest for validation.

```
Empieza a programar o a crear código con IA.
```

2. Perceptron with Tokenizer.

In the first model, you are going to use a `Tokenizer()` object to process the training and validation texts, transforming each review into binary vectors (of length *n*, where *n* is the size of the vocabulary) in which the positions of the words appearing in the review will be coded as `1` (clue: you can use the method `texts_to_matrix()` for this). You can set a maximum size for the vocabulary (parameter `num_words`), but it is not necessary.

Remember that you have to use the `fit_on_texts()` method in order to build the vocabulary of the tokenizer from the training data.

In addition, you have to convert vectors with the labels (negative=0, neutral=1, positive=2) from the training and validation sets to a data type which make possible to use them with the loss function `categorical_crossentropy` (clue: you may want to use the utility function `tensorflow.keras.utils.to_categorical()`).

```
Empieza a programar o a crear código con IA.
```

Now it is time to create the `Sequential` architecture of out first model. In this case, a simple perceptron with three layers (input, hidden, output) will suffice. A few pointers:

- You will need to set the `input_shape` of the first layer of the network to the size of the vocabulary in the `Tokenizer`.
- The number of units and the activation function in the output layer must be appropiate for a three-class classification problem.

```
Empieza a programar o a crear código con IA.
```

Now compile and train the model. You can use any optimizer you want, but the loss function must be `categorical_crossentropy`, the metric used will be `accuracy`, and you will provide the validation data for the computation of the validation loss and validation accuracy at the end of each epoch of training, with the argument `validation_data`.

The model will train for 10 epochs.

Expect a validation accuracy of 0.80-0.83, approximately.

```
Empieza a programar o a crear código con IA.
```

¿Does the validation accuracy grow with each epoch?

(you can write your answer here)

3. Perceptron with a TextVectorizer layer.

Now you are going to implement a new neural network, with two differences with respect to the previous one:

- We will use a `TextVectorizer` Layer instead of a `Tokenizer`.
- The loss function will be `sparse_categorical_crossentropy`.

Your first task is to set the `TextVectorization` layer. Remember you have to create the layer and call the method `adapt()` on the training data before adding the layer to the new model. You can use the default values when creating the layer if you wish, except for `output_mode` that has to be set to `'multi_hot'`, so a binary vector the size of the vocabulary is generated for each example, as `Tokenizer` did in the first model.

```
Empieza a programar o a crear código con IA.
```

Now you can create the your second `Sequential` model, adding its layers one by one. Obviously, the previously created `TextVectorizer` goes first. There is not need to define an input layer. You can add the rest of the layers after the text vectorizer.

```
Empieza a programar o a crear código con IA.
```

Once the topology of the new model is set, you will set the datasets, compile and train it. Important:

- Remember that you are supposed to use `sparse_categorical_crossentropy`, so the label vectors for both training and validation will have to be of the appropiate type and dimensions.
- `TextVectorizer` will not accept its training (or validation) input as a list of strings. If you are using lists to store your input strings, convert those lists to numpy arrays with np.array().

You can use whichever optimizer you prefer, but you will use accuracy to measure the performance of the model, provide the validation data through the argument `validation_data`, and train for 10 epochs.

```
Empieza a programar o a crear código con IA.
```

```
Empieza a programar o a crear código con IA.
```

¿Is the new model any better than the previous one?

(you can write your answer here)

4. CNN with TextVectorizer layer and word embeddings

Finally, you are going to train a third model with the following components:

- A `TextVectorizer` layer.
- An `Embedding` layer.
- One or more `Conv1D` layers.
- A `GlobalMaxPooling1D` layer.
- One or more `Dense` layers for the computation of results.
- A output layer with the appropiate activation function for a multiclass classifier.

You will use the functional API.

Our goal is to process the input texts token by token using a Convolutional Neural Network (CNN) and embeddings. The first step is to define the `TextVectorizer` layer. This time the output of this layer will be a vector of integer numbers (the input for the `Embedding` layer), with one integer for each token in the input text, so `output_mode` must be set to `int` or omitted (since `int` is the default value for this parameter). In addition, all sequences of integers (words) given to the embedding layer must have the same length. To ensure that, you will use the parameter `output_sequence_length` in the definition of the `TextVectorizer` (i.e. `output_sequence_length=100`). That will cut sequences longer than the value of `output_sequence_length` and pad shorter ones with zeros.

Once the layer is defined, it will be trained with the method `adapt()`.

```
Empieza a programar o a crear código con IA.
```

You have to start the defintion of the model with an `Input` layer, e.g.:

```
inputs = keras.Input(shape=(1,),dtype=tf.string)
```

then you can add the `TextVectorizer`, `Conv1D`, ... layers.

The `Embedding` layer has at least two parameters: the size of the vocabulary and the size of the embeddings. For the vocabulary you have two choices: set it in avance when creating the layer, via de `max_tokens` parameter, or to let all tokens of the training set be part of the vocabulary. In the latter case, you can get the vocabulary size from the layer, using the method `vocabulary_size()`.

You must the set embedding dimension to a integer value, e.g. `30`.

In the `Conv1D` layer you have to set two parameters, `filters` and `kernel_size`. Both are integers. The first can have any integer value (e.g., `64` of `128`) but the higher is set, the bigger the number of computations will be, while the second should be small compared to the length of the sequences of words (e.g. `3` or `5`).

We finish with the output layer:

```
outputs = tf.keras.layers.Dense(...)(x)
```

where `x` is the output of the previous layer. At this point we can define the model:

```
model_functional = keras.Model(inputs=inputs, outputs=outputs)
```

```
Empieza a programar o a crear código con IA.
```

You can use exactly the same datasets than in the previous model for training and validation, and the optimizer of you preference, but you will use accuracy as performance metric and sparse categorical crossentropy as loss function, provide the validation data through the argument `validation_data`, and train the model for 10 epochs.

```
Empieza a programar o a crear código con IA.
```

¿Does the new model perform any better than the previous two?

(you can write your answer here)