PROJECT_3

Members:

1- Marim Ashraf Elsayed Mahmoud Amer.
2- Mawada Ashraf Elsayed Mahmoud Amer.
3- Youmna Alsayed Abdalatty Mohamed.

Read data

```
'''
1- Open the file
2- Read data(the data is cleaned)
3- Close the file
'''
# Open file
file = open('../input/dataset-clean-1/republic_clean.txt', 'r')
# Text file content (all data)
text = file.read()
# Close file
file.close()
```

```
#Print the first 500 characters from the text file
print(text[:500])
```

```
The Project Gutenberg EBook of The Republic, by Plato

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org


Title: The Republic

Author: Plato

Translator: B. Jowett
```

Cleaning data

- We want to change all text to words or tokens to use to train models. So we will clean the text to be read to use it.
- Steps:
    a. Replace multiple white spaces with a single white space.
    b. Keep only ASCII, no digits.
    c. Remove single letter chars.
    d. Replace -- with white space.
    e. Split all data into tokens by white space.
    f. Replace all non-english characters with a single space
    g. Convert all tokens into lowercase.
    h. Remove punctuation.
- We will implement each of these cleaning operations in a function. Below is the function clean_text() that takes a file of text as an argument and returns a list of clean tokens.

```
def clean_text(text):
    """ steps:
        - Remove all white spaces.
        - Keep only ASCII, no digits
        - remove single letter chars
        - Replace -- with white space.
        - Split all data into tokens by white space.
        - Remove all non-alphabetic tokens.
        - Convert all tokens into lowercase.
        - remove punctuation
    """

    RE_WSPACE = re.compile(r"\s+", re.IGNORECASE)  #White Space
    RE_ASCII = re.compile(r"[^A-Za-zÀ-ž ]", re.IGNORECASE)    #ASCII
    RE_SINGLECHAR = re.compile(r"\b[A-Za-zÀ-ž]\b", re.IGNORECASE) #Single charachter between two spaces
    RE_TAGS = re.compile(r"<[^>]+>")    #Tags

    #Replace '--' with a space ' '
    text = text.replace('--', ' ')
    #Remove tags
    text = re.sub(RE_TAGS, " ", text)
    #Remove any non english character with a single space.
    text = re.sub(RE_ASCII, " ", text)
    #Remove single charachter between two spaces
    text = re.sub(RE_SINGLECHAR, " ", text)
    #Replace White Space, Tags, ASCII and Single charachter between two spaces with single space
    text = re.sub(RE_WSPACE, " ", text)
    # split into tokens by white space
    tokens = text.split()
    # remove punctuation from each token
    tokens = [t.translate(str.maketrans('', '', string.punctuation)) for t in tokens]
    # make lower case
    tokens = [word.lower() for word in tokens]
    return tokens
```

```
# clean text
# print out some of the tokens
tokens = clean_text(text)
print(tokens[:200])
print('Total Tokens: %d' % len(tokens))
print('Unique Tokens: %d' % len(set(tokens)))
```

```
['the', 'project', 'gutenberg', 'ebook', 'of', 'the', 'republic', 'by', 'plato', 'this', 'ebook', 'is', 'for', 'the', 'use', 'of', 'anyone', 'anywhere', 'at', 'no',
'cost', 'and', 'with', 'almost', 'no', 'restrictions', 'whatsoever', 'you', 'may', 'copy', 'it', 'give', 'it', 'away', 'or', 're', 'use', 'it', 'under', 'the', 'term
s', 'of', 'the', 'project', 'gutenberg', 'license', 'included', 'with', 'this', 'ebook', 'or', 'online', 'at', 'www', 'gutenberg', 'org', 'title', 'the', 'republic',
'author', 'plato', 'translator', 'jowett', 'posting', 'date', 'august', 'ebook', 'release', 'date', 'october', 'last', 'updated', 'june', 'language', 'english', 'star
t', 'of', 'this', 'project', 'gutenberg', 'ebook', 'the', 'republic', 'produced', 'by', 'sue', 'asscher', 'the', 'republic', 'by', 'plato', 'translated', 'by', 'benja
min', 'jowett', 'note', 'the', 'republic', 'by', 'plato', 'jowett', 'etext', 'introduction', 'and', 'analysis', 'the', 'republic', 'of', 'plato', 'is', 'the', 'longes
t', 'of', 'his', 'works', 'with', 'the', 'exception', 'of', 'the', 'laws', 'and', 'is', 'certainly', 'the', 'greatest', 'of', 'them', 'there', 'are', 'nearer', 'appro
aches', 'to', 'modern', 'metaphysics', 'in', 'the', 'philebus', 'and', 'in', 'the', 'sophist', 'the', 'politicus', 'or', 'statesman', 'is', 'more', 'ideal', 'the', 'f
orm', 'and', 'institutions', 'of', 'the', 'state', 'are', 'more', 'clearly', 'drawn', 'out', 'in', 'the', 'laws', 'as', 'works', 'of', 'art', 'the', 'symposium', 'an
d', 'the', 'protagoras', 'are', 'of', 'higher', 'excellence', 'but', 'no', 'other', 'dialogue', 'of', 'plato', 'has', 'the', 'same', 'largeness', 'of', 'view', 'and',
'the', 'same', 'perfection', 'of', 'style', 'no', 'other', 'shows', 'an', 'equal']
Total Tokens: 211442
Unique Tokens: 10229
```

Splitting the data into sequences

- We will split tokens into sequences of 50 input words and 1 output word.
- Each line has 50 input + 1 output = 51 word.
- Printing the total number of the sequence, we can see that Total Sequences: 211391 training pattern.

```
'''
1- We will split tokens into sequences of 50 input words and 1 output word.
2- Each line has 50 input + 1 output = 51 word.
'''

# organize into sequences of tokens
sequences = list()
for i in range(51, len(tokens)):
    # select sequence of tokens
    seq = tokens[i-51:i]
    # convert into a line
    line = ' '.join(seq)
    # save the line
    sequences.append(line)
print('Total Sequences: %d' % len(sequences))
```

```
Total Sequences: 211391
```

- Save the sequence of lines in the file to use this file.
- The open function takes the filename as an input. Then lines are written as one per line then close the file.

```
'''
1- Add all lines with \n in the data variable.
2- Open file to write and save sequence data.
3- Write the data in the file.
'''
data = '\n'.join(sequences)
file = open('sequence_line_file.txt', 'w')
file.write(data)
file.close()
```

Train a statistical language model

In this section, we will need to prepare data or tokens to apply the embedding layers in the data.
Embedding layer steps: after tokenizing the sentences into words.

- Convert the text or tokenizer into integer numbers.
- Splitting the tokenizer into X and y.
- Create a one-hot encoded vector for each y.
- Pass X and, y as an input of the embedded layer.

```
# integer encode sequences of words
token = Tokenizer()
token.fit_on_texts(lines)
sequences = token.texts_to_sequences(lines)
sequences
```

a. To implement the word embedding layer, we should convert input sequences into integers.
b. Tokenizer converts all unique words into unique integer numbers, and then we will convert all input text into numbers by using these unique numbers.

c. We need to define the embedding layer so we need the size of the vocabulary. So we used word_index to list mapping words to their rank/index (int) and it after fit_text_tokenizer() is called on the tokenizer.

```
# vocabulary size
vocab_len = len(token.word_index) + 1
vocab_len
```

d. Splitting the data into inputs(X) and output(y).

```
#Split sequences into X and y
X, y = sequences[:,:-1], sequences[:,-1]
#One hot encoder(y)
y = to_categorical(y, num_classes=vocab_len)
```

Model

In this part, we will build our models using STML once, and once again, we will use GRU

a. LSTM

a. In this part, we will build our models using the LSTM model
b. the following steps that we used:
    i. Added embedding layer, it is very important to determine the vocabulary size and input sequences length. It takes
        1. input_dim = length of vocabulary
        2. output_dim = Dimension of the dense embedding
        3. input_length = Length of input sequences
    ii. Added LSTM(Long Short-Term) layer. It takes
        1. units = 115 units which mean dimensionality of the output space
        2. return_sequences used to return the last output
    iii. Added LSTM(Long Short-Term) layer. It takes
        1. units = 115 units which mean dimensionality of the output space
    iv. Added Dropout layer with a rate equal to 20 % that is used to prevent overfitting
    v. Added Dense layer
        1. 50 units that refer to the dimensionality of the output space
        2. ReLU activation function
    vi. 6. Added Dense layer
        1. Vocabulary length as units that refer to the dimensionality of the output space
        2. Softmax activation function

```
model = Sequential()
# Embedding layer used to convert each word into a fixed length vector
model.add(Embedding(vocab_len, 50, input_length=X.shape[1]))
# LSTM(Long Short-Term) is actually a kind of RNN architecture
model.add(LSTM(115, return_sequences=True))
model.add(LSTM(115))
model.add(Dropout(0.2))# 20% dropout
# Dense layer is the regular deeply connected neural network layer.
model.add(Dense(50, activation='relu'))
model.add(Dense(vocab_len, activation='softmax'))
# Display the structure of the model
print(model.summary())
```

c. Then print the summary of the model, compile (adam optimizer), and fit the model with batch size equal to 50, epochs = 70, and early stopping to prevent overfitting with 5 patience

```
# Training the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit model
model.fit(X, y, batch_size=50, epochs=70, callbacks=[
        tf.keras.callbacks.EarlyStopping(monitor='accuracy', patience=5)
    ])
```

Output result

```
4228/4228 [==============================] - 56s 13ms/step - loss: 3.9154 - accuracy: 0.2558
Epoch 62/70
4228/4228 [==============================] - 57s 13ms/step - loss: 3.9045 - accuracy: 0.2560
Epoch 63/70
4228/4228 [==============================] - 57s 13ms/step - loss: 3.9004 - accuracy: 0.2570
Epoch 64/70
4228/4228 [==============================] - 57s 13ms/step - loss: 3.8922 - accuracy: 0.2570
Epoch 65/70
4228/4228 [==============================] - 57s 13ms/step - loss: 3.8841 - accuracy: 0.2582
Epoch 66/70
4228/4228 [==============================] - 57s 13ms/step - loss: 3.8756 - accuracy: 0.2593
Epoch 67/70
4228/4228 [==============================] - 57s 13ms/step - loss: 3.8678 - accuracy: 0.2597
Epoch 68/70
4228/4228 [==============================] - 56s 13ms/step - loss: 3.8616 - accuracy: 0.2600
Epoch 69/70
4228/4228 [==============================] - 57s 13ms/step - loss: 3.8563 - accuracy: 0.2604
Epoch 70/70
4228/4228 [==============================] - 56s 13ms/step - loss: 3.8462 - accuracy: 0.2617
<keras.callbacks.History at 0x7ffa02d7d650>
```

b. GRU

a. In this part, we will build our models using the GRU model
b. the following steps that we used:
    i. Added embedding layer, it is very important to determine the vocabulary size and input sequences length. It takes
        1. input_dim = length of vocabulary
        2. output_dim = Dimension of the dense embedding
        3. input_length = Length of input sequences
    ii. Added GRU(Gated Recurrent Unit) layer. It takes
        1. units = 112 number of units which means dimensionality of the output space
        2. return_sequences used to return the last output
    iii. Added Dropout layer with a rate equal to 20 % that is used to prevent overfitting
    iv. Added Dense layer
        1. 50 units that refer to the dimensionality of the output space
        2. ReLU activation function
    v. Added Dense layer
        1. Vocabulary length as units that refer to the dimensionality of the output space
        2. Softmax activation function

```
model1 = Sequential()
# Embedding layer used to convert each word into a fixed length vector
model1.add(Embedding(vocab_len, 50, input_length=X.shape[1]))
# GRU (Gated Recurrent Unit) is a variation on the recurrent neural network design and
# It is similar to long-term short-term memory cells
model1.add(GRU(112))
model1.add(Dropout(0.2)) # 20% dropout
# Dense layer is the regular deeply connected neural network layer.
model1.add(Dense(50, activation='relu'))
model1.add(Dense(vocab_len, activation='softmax'))
# Display the structure of the model
print(model1.summary())
```

c. then print the summary of the model, compile (adam optimizer), and fit the model with batch size equal to 50, epochs = 100, and early stopping to prevent overfitting with 5 patience

```
# Training the model
model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit model
model1.fit(X, y, batch_size=100, epochs=100, callbacks=[
        tf.keras.callbacks.EarlyStopping(monitor='accuracy', patience=5)
    ])
```

Output result

```
Epoch 93/100
2114/2114 [==============================] - 22s 10ms/step - loss: 3.1494 - accuracy: 0.3458
Epoch 94/100
2114/2114 [==============================] - 22s 11ms/step - loss: 3.1432 - accuracy: 0.3470
Epoch 95/100
2114/2114 [==============================] - 22s 10ms/step - loss: 3.1366 - accuracy: 0.3480
Epoch 96/100
2114/2114 [==============================] - 22s 10ms/step - loss: 3.1415 - accuracy: 0.3472
Epoch 97/100
2114/2114 [==============================] - 22s 10ms/step - loss: 3.1373 - accuracy: 0.3478
Epoch 98/100
2114/2114 [==============================] - 22s 10ms/step - loss: 3.1319 - accuracy: 0.3477
Epoch 99/100
2114/2114 [==============================] - 22s 10ms/step - loss: 3.1276 - accuracy: 0.3480
Epoch 100/100
2114/2114 [==============================] - 22s 10ms/step - loss: 3.1248 - accuracy: 0.3486
<keras.callbacks.History at 0x7f9ccc215550>
```

LSTM accuracy: 26%

GRU accuracy: 35 %

Finally, the GRU gave us higher accuracy than the LSTM model. We tried multiple models, but it was these trials that gave us the highest accuracy. We tried to use more than 100 epochs, but these models gave us a crash in memory as they need more memory to run. We tried to solve this problem by running the code in kaggel nested in colab, but it gave us the same problem.

Reference

https://machinelearningmastery.com/how-to-develop-a-word-level-neural-language-model-in-keras/