NLP RATIONALE

Adrián Salas Rodrigues

INTELLIGENT SYSTEMS

1. Problem to Solve

For this work, I used a dataset of messages classified as spam or ham. What I did is to develop, using a BERT model and its features (text preprocessing and word-embedding), a spam classifier that is going to predict if a given message is spam or ham. The BERT model classifies the dataset by using pre-trained weights downloaded from the TensorFlow Hub repository.

	label	body
628	spam	New TEXTBUDDY Chat 2 horny guys in ur area 4 j...
1664	ham	Ãœ v ma fan...
1687	spam	Free Top ringtone -sub to weekly ringtone-get ...
1595	ham	Pls confirm the time to collect the cheque.
1263	ham	Ok. No wahala. Just remember that a friend in ...

The dataset can be found here:

https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection

The GitHub code can be found here:

https://github.com/AdrianSalas500/Spam-Classifier-NLP-.git

The steps for developing the spam classifier were:

1. Install dependencies

```
!pip install tensorflow
!pip install tensorflow-text
```

```
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_text as text
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

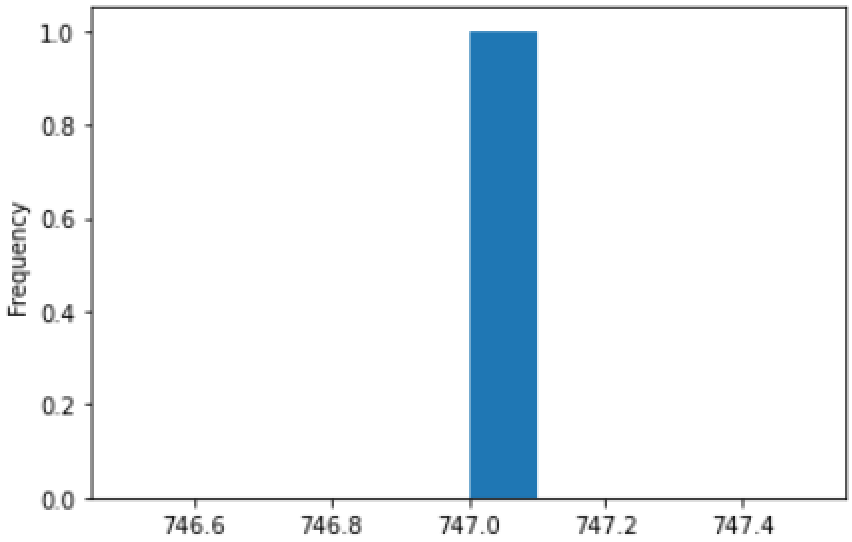
2. Import the dataset

```
df=pd.read_excel('SMSSpamCollection.xlsx')
```

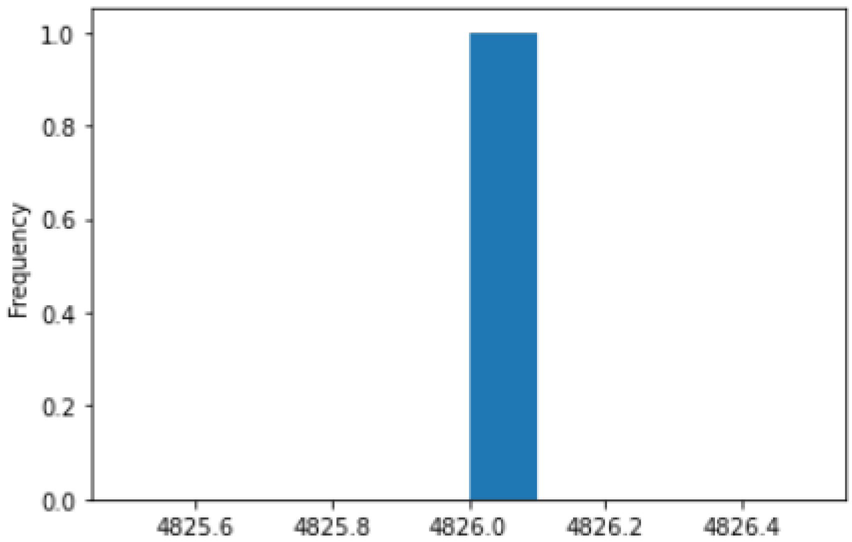
3. Balance spam and ham classes and obtain a new balanced dataset

We have 747 spam emails and 4826 ham emails. The ham messages are significantly higher, so in order to balance the two classes, we reduce the number of ham messages to 747.

Spam samples



Ham samples



3.1. Create two data frames (one for each class)

```
spam_df = df[df['label']=='spam']
ham_df = df[df['label']=='ham']
```

3.2. Balance the ham dataset and create a new balanced dataframe

```
ham_df_balanced = ham_df.sample(spam_df.shape[0])
```

```
dfb = pd.concat([ham_df_balanced, spam_df])
```

3.3. In this table you can see that now spam and ham have the same number of messages (747)

```
dfb['label'].value_counts()
```

```
ham     747
spam    747
Name: label, dtype: int64
```

4. Download the BERT models from the TensorFlow Hub repository (for preprocessing and encoding)

   We download two BERT models, one to perform preprocessing and the other one for encoding.

```
preprocess = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3")
encoder = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4")
```

5. Initialize the BERT and neural network layers

   5.1. BERT layers

   We use preprocess as the input for this layer. Then, the encoder is going to convert the preprocessed text in vectors (output of the layer).

```
tinput = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
tpreprocess = preprocess(tinput)
tencoder = encoder(tpreprocess)
```

   5.2.Neural Network layers

   The output is going to be fed in the neural network layers, that are two, the Dropout layer, and the Dense layer.

```
layer = tf.keras.layers.Dropout(0.1, name="dropout")(tencoder['pooled_output'])
layer = tf.keras.layers.Dense(1, activation='sigmoid', name="output")(layer)
```

   5.3. Final model

   We add the input and output layers to construct the final model

```
model = tf.keras.Model(inputs=[tinput], outputs = [layer])
```

6. Split the dataset using the train_test_split function

   Using sklearn, we split the dataset in two sets (training and testing sets)

```
dfb['spam_or_ham'] = dfb['label'].apply(lambda x: 1 if x=='spam' else 0)
X_train, X_test, y_train, y_test = train_test_split(dfb['body'],dfb['spam_or_ham'], stratify=dfb['spam_or_ham'])
```

7. Fit the model with 10 epochs

```
model.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
35/35 [==============================] - 202s 6s/step - loss: 0.6754 - accuracy: 0.5813
Epoch 2/10
35/35 [==============================] - 212s 6s/step - loss: 0.5448 - accuracy: 0.7946
Epoch 3/10
35/35 [==============================] - 248s 7s/step - loss: 0.4546 - accuracy: 0.8616
Epoch 4/10
35/35 [==============================] - 215s 6s/step - loss: 0.4021 - accuracy: 0.8732
Epoch 5/10
35/35 [==============================] - 201s 6s/step - loss: 0.3733 - accuracy: 0.8884
Epoch 6/10
35/35 [==============================] - 196s 6s/step - loss: 0.3504 - accuracy: 0.8857
Epoch 7/10
35/35 [==============================] - 194s 6s/step - loss: 0.3314 - accuracy: 0.8973
Epoch 8/10
35/35 [==============================] - 198s 6s/step - loss: 0.3135 - accuracy: 0.9098
Epoch 9/10
35/35 [==============================] - 198s 6s/step - loss: 0.2941 - accuracy: 0.9125
Epoch 10/10
35/35 [==============================] - 214s 6s/step - loss: 0.2830 - accuracy: 0.9116
```

   The model obtains a **91,16 %** of accuracy.

8. Evaluate the model in the testing dataset to obtain an array of 0´s and 1´s predicting if a message is spam or ham

```
y_pred = model.predict(X_test)
y_pred = y_pred.flatten()
y_pred = np.where(y_pred > 0.5, 1, 0)
y_pred
```

```
array([1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0,
       1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0])
```

2. Experiment(s) done

To assess that the model was classified correctly, test messages were used.

```
test_dataset = [
 'You can win a lot of money, register in the link below',
 'You have an iPhone 10, spin the image below to claim your prize and it will be delivered in your door step',
 'You have an offer, the company will give you 50% off on every item purchased.',
 'Hey Bravin, do not be late for the meeting tomorrow will start lot exactly 10:30 am',
 "See you monday, we have alot to talk about the future of this company ."
]

model.predict(test_dataset)
```

This array of samples is composed by three spam messages and two ham ones.

3. Analysis of results

After defining the samples, I used the model to predict the results. These results were an array of numbers, in which a number above 0.5 indicated that the message is considered spam, and a number below 0.5 is ham.

```
array([[0.6969731 ],
       [0.9157373 ],
       [0.53140694],
       [0.6665542 ],
       [0.15635076]], dtype=float32)
```

As can be seen in the image, 4 out of 5 samples were correctly predicted, giving an accuracy rate of 80 %.