**NAME: REGISTRATION NO:**

**JAMES KAMAU TU01-BE213-0367/2017**

**GRACE WAFULA SANYA TU01-BE213-0345/2017**

**DANIEL KABUGI TU01-BE213-0378/2013**

**ROTICH AUSEBIOUR TU01-BE213-0380/2016**

**NICHOLAS YEGON TU01-BE213-0100/2017**

**CODE DOCUMENTATION**

import tensorflow as tf

import tensorflow\_hub as hub

import tensorflow\_text as text

import pandas as pd

From the above code, we have imported the following packages:

* tensorflow: It is the machine learning package used to build the neural network. It will create the input and output layers of our machine learning model.
* tensorflow\_hub: It contains a pre-trained machine model used to build our text classification. Our pre-trained model is BERT. We will re-use the BERT model and fine-tune it to meet our needs.
* tensorflow\_text: It will allow us to work with text. In this tutorial, we are solving a text-classification problem.
* pandas: We used Pandas to load our dataset. We also use Pandas for data manipulation and analysis. It gives us a clear overview of how our dataset is structured.

df = pd.read\_csv("Testy.csv")

We runned this command to load the dataset. We used our own dataset named Testy.

df.head(5

We used this command to display the first five rows of the dataset to make sure the dataset was fetched and also to see the structure of the dataset.

This was the output as shown below.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | **Task** | **Topic** | **Appointments** | **Tender** | **public\_holidays** | **Job\_vacancies** | | --- | --- | --- | --- | --- | --- | --- | | **140** | Public participation is one of the national va... | Tender | 0 | 2 | 0 | 0 | | **98** | The County Governments Act, 2012 at section 11... | Tender | 0 | 2 | 0 | 0 | | **91** | The Commission takes note of the fundamental r... | Tender | 0 | 2 | 0 | 0 | | **123** | The Commission is cognizant of the fundamental... | Tender | 0 | 2 | 0 | 0 | | **52** | IN EXERCISE of the powers conferred by section... | Appointments | 1 | 0 | 0 | 0 | |

From the image above, our dataset has 4 categories: Appointments, Tender, Job vacancies and Public holidays. The dataset also has a task column representing the Kenya Gazette content.

Let’s see the individual value count for the Appointments, Public holidays, tender and job vacancies for Kenya Gazette content.

df['Topic'].value\_counts()

The output from the above code is shown below.

Public holidays 32

Appointments 30

Tender 99

Appointments 30

Name: Topic, dtype: int64

To create the Dataframe for the Appointments, Public holidays, Tender and Job vacancies, we used the following code.

df\_Appointments = df[df['Topic']=='Appointments']

df\_public\_holidays = df[df['Topic']=='public\_holidays']

df\_Job\_vacancies = df[df['Topic']=='Job\_vacancies']

df\_Tender = df[df['Topic']=='Tender']

df\_balanced = pd.concat([df\_Appointments, df\_public\_holidays, df\_Tender, df\_Job\_vacancies])

The pd.concat method concatenated df\_Appointments, df\_Public holidays, df\_Tender and df\_Job vacancies into a single data frame. It saved the dataset into a variable df\_balanced.

df\_balanced['Topic'].value\_counts()

This above line of code is used to check the value count for each category from the variable df\_balanced.

Appointments 30

Public holidays 32

Tender 99

Appointments 30

Name: Topic, dtype: int64

We need to label our dataset into 1, 2, 3 and 4. 1 will represent the data samples that belong to the Appointment class, 2 will represent the data samples that belong to the Tender class, 3 will represent the data samples that belong to the Public\_holidays class and 4 will represent the data samples that belong to the job vacancies.

We used the code below.

df\_balanced['Appointments']=df\_balanced['Topic'].apply(lambda x: 1 if x=='Appointments' else 0)

df\_balanced['Tender']=df\_balanced['Topic'].apply(lambda x: 2 if x=='Tender' else 0)

df\_balanced['public\_holidays']=df\_balanced['Topic'].apply(lambda x: 3 if x=='public\_holidays' else 0)

df\_balanced['Job\_vacancies']=df\_balanced['Topic'].apply(lambda x: 4 if x=='Job\_vacancies' else 0)

From the code above, we used lambda to write our logic. The apply method ran the written logic. This enabled us to label our dataset.

To see the output of 5 data samples, we ran this code:

df\_balanced.sample(5)

The output is shown below.

|  | **Task** | **Topic** | **Appointments** | **Tender** | **public\_holidays** | **Job\_vacancies** |
| --- | --- | --- | --- | --- | --- | --- |
| **140** | Public participation is one of the national va... | Tender | 0 | 2 | 0 | 0 |
| **98** | The County Governments Act, 2012 at section 11... | Tender | 0 | 2 | 0 | 0 |
| **91** | The Commission takes note of the fundamental r... | Tender | 0 | 2 | 0 | 0 |
| **123** | The Commission is cognizant of the fundamental... | Tender | 0 | 2 | 0 | 0 |
| **52** | IN EXERCISE of the powers conferred by section... | Appointments | 1 | 0 | 0 | 0 |

We split our dataset into two sets, the first set will be used for training and the second set will be used for testing.

We then split our dataset using the train\_test\_split, which we imported as follows:

from sklearn.model\_selection import train\_test\_split

To split this dataset, we used this code:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Task'],df\_balanced['Appointments'], stratify=df\_balanced['Appointments'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Task'],df\_balanced['Tender'], stratify=df\_balanced['Tender'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Task'],df\_balanced['public\_holidays'], stratify=df\_balanced['public\_holidays'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_balanced['Task'],df\_balanced['Job\_vacancies'], stratify=df\_balanced['Job\_vacancies'])

In the code above, we used stratify to ensure equal distribution of classes in the train and test sample. This ensured we had an equal amount of Appointments, Public holidays, Tender and Job vacancies text after splitting. After splitting the dataset, we now started working with BERT.

BERT stands for Bidirectional Encoder Representations from Transformers. BERT models help machines understand and interpret the meaning of the text. It uses immediately preceding text to understand the context. It also checks the relationships of words within a sentence to give the actual meaning of words.

BERT then converts a given sentence into an embedding vector. Embedding vector is used to

represent the unique words in a given document. BERT ensures words with the same meaning

will have a similar representation.

The BERT process undergoes two stages: Preprocessing and encoding.

Preprocessing

Preprocessing is the first stage in BERT. This stage involves removing noise from our dataset. In this stage, BERT will clean the dataset. It also removes duplicate records from the dataset.

It will also format the dataset so that it can be easy to use during model training. This will increase the model performance.

Encoding

Because machine learning does not work well with the text, we need to convert the text into real numbers. This process is known as encoding. BERT will convert a given sentence into an embedding vector.

We first downloaded the BERT model.

BERT models are usually pre-trained. They are available in [TensorFlow Hub](https://www.tensorflow.org/hub). TensorFlow Hub contains all the pre-trained machine learning models that are downloaded.

We downloaded two models, one to perform preprocessing and the other one for encoding. The links for the models are shown below.

bert\_preprocess = hub.KerasLayer("https://tfhub.dev/tensorflow/bert\_en\_uncased\_preprocess/3")

bert\_encoder = hub.KerasLayer(<https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4>)

After downloading the model, let’s start building our model using TensorFlow. We will start by initializing the BERT layers.

text\_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')

preprocessed\_text = bert\_preprocess(text\_input)

outputs = bert\_encoder(preprocessed\_text)

In the code above, we created an input layer using tf.keras.layers.Input method. We then used the preprocessed\_text as input for this layer.

The bert\_encoder function converts the preprocessed text into embedding vectors. This is the output of this layer. The outputs were then fed into the neural network layers.

l = tf.keras.layers.Dropout(0.1, name="dropout")(outputs['pooled\_output'])

l = tf.keras.layers.Dense(1, activation='softmax', name="output")(l)

The neural network has two layers, the Dropout layer, and the Dense layer.

#### ‘Dropout’ layer

This layer was used to prevent model overfitting. We used 0.1% of the neurons to handle overfitting. Overfitting happens when a model perfectly learns from training data but performs poorly in testing. We also give it the name dropout.

We then added the input for this layer as a function using (outputs['pooled\_output']). This input was the output of the BERT layers.

#### ‘Dense’ layer

It only has one neuron. We also initialized the activation function as softmax. Softmax is used when we have output values where the probabilities of each value are proportional to the relative scale of each value in the vector. In our case, when making predictions, the prediction probability will lie between 0 and 4. That’s why it is best suited.

We also named the layer as output because this was our output layer.

We now added the input and output layers to construct the final model as shown below:

model = tf.keras.Model(inputs=[text\_input], outputs = [l])

The model used the text\_input as inputs and had only one single output.

We displayed the model summary so that we could see all the input and output layers used.

model.summary()

We now proceeded to compiling our model.

During this stage, we set the optimizer, the loss function, and the metrics for our model as shown below.

* The Optimizer is used to improve the model performance and reduce errors that occur during model training. We used the adam optimizer.
* Metrics will be used to check the model performance so that we can know how we trained our model. We set the BinaryAccuracy(name='accuracy') which was used to calculate the accuracy score of the model.
* The Loss function is used to calculate the model error during the training phase. We used categorical\_crossentropy as our loss function because our output is more than binary.

We now set these parameters.

METRICS = [

tf.keras.metrics.BinaryAccuracy(name='accuracy'),

tf.keras.metrics.Precision(name='precision'),

tf.keras.metrics.Recall(name='recall')

]

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=METRICS)

After compiling the model, we can now fit it into our dataset.

In this stage, the model learns from the training data samples. The model will identify patterns in the training dataset and gain knowledge.

model.fit(X\_train, y\_train, epochs=10)

We specified the number of epochs as 10. The model then iterates through the dataset ten times and prints the accuracy score after each iteration.

Next, we used the model to make predictions.

To evaluate the model, we will use the model to classify the data samples in the testing dataset. They should be classified into either Appointments or public holidays.

We used the following code:

y\_predicted = model.predict(X\_test)

y\_predicted = y\_predicted.flatten()

The model.predict method will give the prediction results which are in a 2D array, but we want our results in a 1D array. To convert the result from the 2D to 1D array we used the y\_predicted.flatten() function.

Since we used a softmax activation function, the prediction probabilities will lie between 0.0 to 4.0. So, if the prediction result is > 0.5 the output should be 1, and if it is < 0.5, the output should be 0 and so on and so forth.

We used NumPy to help us create this logic.

import numpy as np

y\_predicted = np.where(y\_predicted > 0.5, 1, 0)

y\_predicted = np.where(y\_predicted > 1, 2, 1)

y\_predicted = np.where(y\_predicted > 1.5, 3, 2)

y\_predicted = np.where(y\_predicted > 2, 4, 3)

y\_predicted

We now are able to use this model to make a single prediction using input texts.

We used the following text to make predictions:

sample\_dataset = ["IN EXERCISE of the authority conferred in me by the Constitution of Kenya, the County Governments Act and the Public Appointments (County Assemblies Approval) Act, I, appoint"]

The text above shows an example of Kenya Gazette content. We used our model to classify this content as either Appointments, Public holidays, Tender and Job vacancies.

To make run the prediction, we used this code:

model.predict(sample\_dataset)

The prediction results are shown below.

array([[1.]], dtype=float32)

From the output above, the sample text was classified as Appointments. It had a prediction probability that is 1.

### **Conclusion**

In this research, we learned how to build a text classification model. The model was able to classify Kenya Gazette Content as Appointments or public holidays. We started by using BERT to convert a given sentence into an embedding vector. This was done using the pre-trained BERT models.

We created our model using TensorFlow and initialized all the input and output layers. We followed all the stages of building the neural network and finally came up with a text classification model. Finally, we used the model to make predictions, the model was able to give accurate predictions.

**NOTE**

We also made the corrections of making the classification based on 4 categories by modifying this code.