BERT tutorial: Classify spam vs no spam emails

```
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_text as text
```

Import the dataset (Dataset is taken from kaggle)

```
import pandas as pd

df = pd.read_csv("spam.csv")
df.head(5)
```

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

```
df.groupby('Category').describe()
```

	Message			
	count	unique	top	freq
Category				
ham	4825	4516	Sorry, I'll call later	30
spam	747	641	Please call our customer service representativ...	4

```
df['Category'].value_counts()
```

```
ham     4825
spam     747
Name: Category, dtype: int64
```

```
747/4825
```

```
0.15481865284974095
```

15% spam emails, 85% ham emails: This indicates class imbalance

```
df_spam = df[df['Category']=='spam']
df_spam.shape
```

```
(747, 2)
```

```
df_ham = df[df['Category']=='ham']
df_ham.shape
```

```
(4825, 2)
```

```
df_ham_downsampled = df_ham.sample(df_spam.shape[0])
df_ham_downsampled.shape
```

```
(747, 2)
```

```
df_balanced = pd.concat([df_ham_downsampled, df_spam])
df_balanced.shape
```

```
(1494, 2)
```

```
df_balanced['Category'].value_counts()
```

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

```
df_balanced['spam']=df_balanced['Category'].apply(lambda x: 1 if x=='spam' else 0)
df_balanced.sample(5)
```

	Category	Message	spam
4925	ham	We can go 4 e normal pilates after our intro...	0
4249	spam	accordingly. I repeat, just text the word ok o...	1
5006	ham	Guess which pub im in? Im as happy as a pig in...	0
2567	ham	You in your room? I need a few	0
14	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	0

Split it into training and test data set

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df_balanced['Message'],df_balanced['spam'], stratify=df_balanced['spam'])
```

```
X_train.head(4)
```

```
3354    I emailed yifeng my part oredi.. Can ü get it ...
466     great princess! I love giving and receiving or...
4154    URGENT!! Your 4* Costa Del Sol Holiday or £500...
3162    Mystery solved! Just opened my email and he's ...
Name: Message, dtype: object
```

Now lets import BERT model and get embeding vectors for few sample statements

```
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")
```

```
def get_sentence_embeding(sentences):
    preprocessed_text = bert_preprocess(sentences)
    return bert_encoder(preprocessed_text)['pooled_output']

get_sentence_embeding([
    "500$ discount. hurry up",
    "Bhavin, are you up for a volleybal game tomorrow?"]
)
```

```
<tf.Tensor: shape=(2, 768), dtype=float32, numpy=
array([[-0.8435169 , -0.51327276, -0.8884574 , ..., -0.74748874,
        -0.75314736,  0.91964483],
       [-0.87208366, -0.50543964, -0.94446677, ..., -0.858475  ,
        -0.7174535 ,  0.8808298 ]], dtype=float32)>
```

Get embeding vectors for few sample words. Compare them using cosine similarity

```
e = get_sentence_embeding([
    "banana",
    "grapes",
    "mango",
    "jeff bezos",
    "elon musk",
    "bill gates"
]
)
```

```
from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity([e[0]],[e[1]])
```

```
array([[0.9911089]], dtype=float32)
```

Values near to 1 means they are similar. 0 means they are very different. Above you can use comparing "banana" vs "grapes" you get 0.99 similarity as they both are fruits

```
cosine_similarity([e[0]],[e[3]])
```

```
array([[0.8470385]], dtype=float32)
```

Comparing banana with jeff bezos you still get 0.84 but it is not as close as 0.99 that we got with grapes

```
cosine_similarity([e[3]],[e[4]])
```

```
array([[0.98720354]], dtype=float32)
```

Jeff bezos and Elon musk are more similar then Jeff bezos and banana as indicated above

Build Model

There are two types of models you can build in tensorflow.

(1) Sequential (2) Functional

So far we have built sequential model. But below we will build functional model. More information on these two is here: https://becominghuman.ai/sequential-vs-functional-model-in-keras-20684f766057

```
# 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)

# Neural network layers
l = tf.keras.layers.Dropout(0.1, name="dropout")(outputs['pooled_output'])
l = tf.keras.layers.Dense(1, activation='sigmoid', name="output")(l)

# Use inputs and outputs to construct a final model
model = tf.keras.Model(inputs=[text_input], outputs = [l])
```

https://stackoverflow.com/questions/47605558/importerror-failed-to-import-pydot-you-must-install-pydot-and-graphviz-for-py

```
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
text (InputLayer)               [(None,)]            0

__________________________________________________________________________________________________
keras_layer (KerasLayer)        {'input_mask': (None 0           text[0][0]
__________________________________________________________________________________________________
keras_layer_1 (KerasLayer)      {'default': (None, 7 109482241   keras_layer[0][0]
                                                                 keras_layer[0][1]
                                                                 keras_layer[0][2]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 768)          0           keras_layer_1[0][13]
__________________________________________________________________________________________________
output (Dense)                  (None, 1)            769         dropout[0][0]
==================================================================================================
Total params: 109,483,010
Trainable params: 769
Non-trainable params: 109,482,241
__________________________________________________________________________________________________
```

```
len(X_train)
```

```
1120
```

```
METRICS = [
      tf.keras.metrics.BinaryAccuracy(name='accuracy'),
      tf.keras.metrics.Precision(name='precision'),
      tf.keras.metrics.Recall(name='recall')
]

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=METRICS)
```

Train the model

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

```
Epoch 1/10
35/35 [==============================] - 7s 189ms/step - loss: 0.3398 - accuracy: 0.8857 - precision: 0.8750 - recall: 0.9000 2s - l
```

```
Epoch 2/10
35/35 [==============================] - 6s 185ms/step - loss: 0.3271 - accuracy: 0.8857 - precision: 0.8649 - recall: 0.9143
Epoch 3/10
35/35 [==============================] - 7s 187ms/step - loss: 0.3093 - accuracy: 0.8920 - precision: 0.8844 - recall: 0.9018
Epoch 4/10
35/35 [==============================] - 7s 187ms/step - loss: 0.2920 - accuracy: 0.9071 - precision: 0.8986 - recall: 0.9179
Epoch 5/10
35/35 [==============================] - 7s 187ms/step - loss: 0.2837 - accuracy: 0.9098 - precision: 0.9076 - recall: 0.9125
Epoch 6/10
35/35 [==============================] - 7s 187ms/step - loss: 0.2741 - accuracy: 0.9062 - precision: 0.9027 - recall: 0.9107
Epoch 7/10
35/35 [==============================] - 7s 189ms/step - loss: 0.2643 - accuracy: 0.9089 - precision: 0.8962 - recall: 0.9250 4s - l
Epoch 8/10
35/35 [==============================] - 7s 186ms/step - loss: 0.2570 - accuracy: 0.9161 - precision: 0.9161 - recall: 0.9161
Epoch 9/10
35/35 [==============================] - 7s 196ms/step - loss: 0.2512 - accuracy: 0.9134 - precision: 0.9026 - recall: 0.9268
Epoch 10/10
35/35 [==============================] - 7s 193ms/step - loss: 0.2419 - accuracy: 0.9179 - precision: 0.9239 - recall: 0.9107
<tensorflow.python.keras.callbacks.History at 0x1db822fcf70>
```

```
model.evaluate(X_test, y_test)
```

```
12/12 [==============================] - 4s 194ms/step - loss: 0.2600 - accuracy: 0.9064 - precision: 0.8486 - recall: 0.9893
[0.2599719762802124,
 0.9064171314239502,
 0.8486238718032837,
 0.9893048405647278]
```

```
y_predicted = model.predict(X_test)
y_predicted = y_predicted.flatten()
```

```
import numpy as np

y_predicted = np.where(y_predicted > 0.5, 1, 0)
y_predicted
```

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

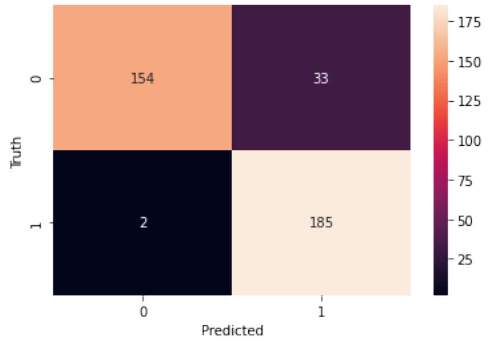
```
from sklearn.metrics import confusion_matrix, classification_report

cm = confusion_matrix(y_test, y_predicted)
cm
```

```
array([[154,  33],
       [  2, 185]], dtype=int64)
```

```
from matplotlib import pyplot as plt
import seaborn as sn
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(33.0, 0.5, 'Truth')
```




```
print(classification_report(y_test, y_predicted))
```

```
              precision    recall  f1-score   support

           0       0.99      0.82      0.90       187
           1       0.85      0.99      0.91       187

    accuracy                           0.91       374
   macro avg       0.92      0.91      0.91       374
weighted avg       0.92      0.91      0.91       374
```

Inference

```
reviews = [
    'Enter a chance to win $5000, hurry up, offer valid until march 31, 2021',
    'You are awarded a SiPix Digital Camera! call 09061221061 from landline. Delivery within 28days. T Cs Box177. M221BP. 2yr warranty.
    'it to 80488. Your 500 free text messages are valid until 31 December 2005.',
    'Hey Sam, Are you coming for a cricket game tomorrow',
    "Why don't you wait 'til at least wednesday to see if you get your ."
]
model.predict(reviews)
```

```
array([[0.8734353 ],
       [0.92858446],
       [0.8960864 ],
       [0.29311982],
       [0.13262196]], dtype=float32)
```