

```
# import libraries
import tensorflow as tf
import pandas as pd
from tensorflow import keras

import tensorflow_datasets as tfds
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)

2.15.0

# get data files
!wget https://cdn.freecodecamp.org/project-data/sms/train-data.tsv
!wget https://cdn.freecodecamp.org/project-data/sms/valid-data.tsv

train_file_path = "train-data.tsv"
test_file_path = "valid-data.tsv"

--2024-01-31 01:31:09-- https://cdn.freecodecamp.org/project-data/sms/train-data.tsv
Resolving cdn.freecodecamp.org (cdn.freecodecamp.org)... 104.26.2.33, 104.26.3.33, 172.67.70.149, ...
Connecting to cdn.freecodecamp.org (cdn.freecodecamp.org)|104.26.2.33|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 358233 (350K) [text/tab-separated-values]
Saving to: 'train-data.tsv'

train-data.tsv      100%[=====>] 349.84K  --.-KB/s   in 0.02s

2024-01-31 01:31:09 (15.2 MB/s) - 'train-data.tsv' saved [358233/358233]

--2024-01-31 01:31:09-- https://cdn.freecodecamp.org/project-data/sms/valid-data.tsv
Resolving cdn.freecodecamp.org (cdn.freecodecamp.org)... 104.26.2.33, 104.26.3.33, 172.67.70.149, ...
Connecting to cdn.freecodecamp.org (cdn.freecodecamp.org)|104.26.2.33|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 118774 (116K) [text/tab-separated-values]
Saving to: 'valid-data.tsv'

valid-data.tsv      100%[=====>] 115.99K  --.-KB/s   in 0.009s

2024-01-31 01:31:09 (12.8 MB/s) - 'valid-data.tsv' saved [118774/118774]

# Reading Data & Setting Column Labels
train_data = pd.read_table(train_file_path, header=None, names=["spam", "message"])
test_data = pd.read_table(test_file_path, header=None, names=["spam", "message"])

train_data.head()
```

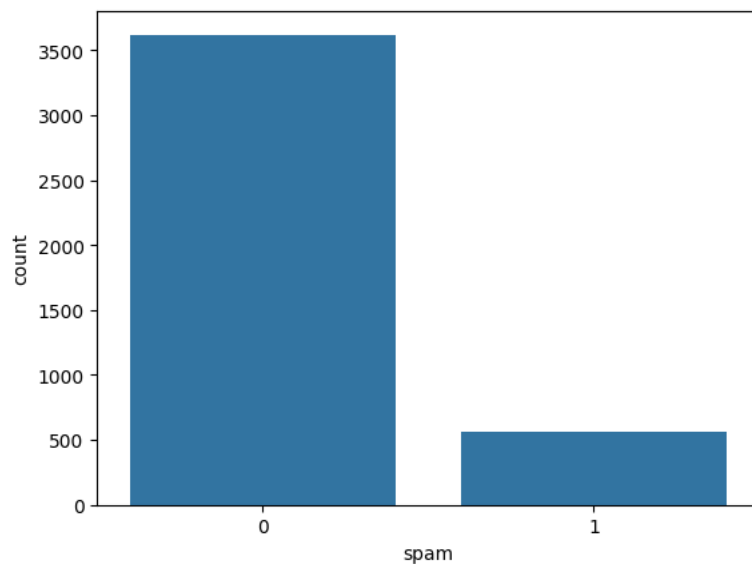
	spam	message
0	ham	ahhhh...just woken up!had a bad dream about u ...
1	ham	you can never do nothing
2	ham	now u sound like manky scouse boy steve,like! ...
3	ham	mum say we wan to go then go... then she can s...
4	ham	never y lei... i v lazy... got wat? dat day ü ...

```
# Converting Categorical Data to Integers
train_data["spam"] = train_data["spam"].replace({"ham": 0, "spam": 1})
test_data["spam"] = test_data["spam"].replace({"ham": 0, "spam": 1})
train_data["spam"].value_counts()

0      3619
1       560
Name: spam, dtype: int64

# Modeling HamOrSpam Data
import seaborn as sns

sns.countplot(x='spam', data=train_data)
plt.show()
```



```
# Calculating HamOrSpam & Creating Data Frame
ham_msg = train_data[train_data["spam"] == 0]
neg = ham_msg.shape[0]
print(f"negative: {neg}")
```

```
spam_msg = train_data[train_data["spam"] == 1]
pos = spam_msg.shape[0]
print(f"positive: {pos}")
```

```
total = neg + pos
print(f"total: {total}")
```

```
balanced_data = train_data
```

```
negative: 3619
positive: 560
total: 4179
```

```
# Cleaning Data: Removing Punctuation
```

```
import string
punctuations_list = string.punctuation
def remove_punctuations(text):
    temp = str.maketrans('', '', punctuations_list)
    return text.translate(temp)
```

```
balanced_data['message'] = balanced_data['message'].apply(lambda x: remove_punctuations(x))
```

```
# Perform the same pre-processing on test data as on training data.
test_data['message'] = test_data['message'].apply(lambda x: remove_punctuations(x))
```

```
balanced_data.head()
```

	spam	message	
0	0	ahhhhjust woken uphad a bad dream about u thos...	
1	0	you can never do nothing	
2	0	now u sound like manky scouse boy stevelike i ...	
3	0	mum say we wan to go then go then she can shun...	
4	0	never y lei i v lazy got wat dat day ü send me...	

```
# Cleaning Data: Removing Stop Words
```

```
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')

stop_words = stopwords.words('english')
new_stopwords = ["u", "ur"]
stop_words.extend(new_stopwords)

def remove_stopwords(text):
    imp_words = []

    # Storing the important words
    for word in str(text).split():
        word = word.lower()

        if word not in stop_words:
            imp_words.append(word)

    output = " ".join(imp_words)



    return output

balanced_data['message'] = balanced_data['message'].apply(lambda text: remove_stopwords(text))

# Perform the same pre-processing on test data as on training data.
test_data['message'] = test_data['message'].apply(lambda text: remove_stopwords(text))

balanced_data.head()
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.
```

	spam	message	
0	0	ahhhhjust woken uphad bad dream thoso dont lik...	
1	0	never nothing	
2	0	sound like manky scouse boy stevelike travelli...	
3	0	mum say wan go go shun bian watch da glass exh...	
4	0	never lei v lazy got wat dat day ü send da url...	

```
# Displaying Data as WordCloud
```

```
from wordcloud import WordCloud
def plot_word_cloud(data, typ):
    email_corpus = " ".join(data['message'])

    plt.figure(figsize=(7, 7))

    wc = WordCloud(
        background_color='black',
        max_words=100,
        width=800,
        height=400,
        collocations=False
    ).generate(email_corpus)

    plt.imshow(wc, interpolation='bilinear')
    plt.title(f'WordCloud for {typ} emails', fontsize=15)
    plt.axis('off')
    plt.show()

plot_word_cloud(balanced_data[balanced_data['spam'] == 0], typ='Non-Spam')
plot_word_cloud(balanced_data[balanced_data['spam'] == 1], typ='Spam')
```



```
'free entry £250 weekly competition text word win 80086 18 tc wwwtxttowincouk',
'pls stop bootydelious 32f inviting friend reply yes434 no434 see wwwsmsacubootydelious stop send stop frnd 62468',
'tell bad character dnt lik ill try change ltgt add tat 2 new year resolution waiting replybe frankgood morning'],
dtype=object)
```

```
# Tokenizing & Padding Text
```

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

```
# Tokenize the text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_X)
```

```
# Convert text to sequences
train_sequences = tokenizer.texts_to_sequences(train_X)
test_sequences = tokenizer.texts_to_sequences(test_X)
```

```
# Pad sequences to have the same length
max_len = 100
```

```
train_sequences = pad_sequences(
    train_sequences,
    maxlen=max_len,
    padding='post',
    truncating='post'
)
```

```
test_sequences = pad_sequences(
    test_sequences,
    maxlen=max_len,
    padding='post',
    truncating='post'
)
```

```
print(train_sequences[0])
```

```
[3468 2262 3469 293 663 3470 7 12 76 94 11 86 1731 48
 203 2 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]
```

```
# Build the model
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Embedding(
        input_dim=len(tokenizer.word_index) + 1,
        output_dim=32,
        mask_zero=True,
    ),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.LSTM(16),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
# Print the model summary
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, None, 32)	257568
dropout (Dropout)	(None, None, 32)	0
lstm (LSTM)	(None, 16)	3136
dense (Dense)	(None, 16)	272
dense_1 (Dense)	(None, 1)	17

```
=====
Total params: 260993 (1019.50 KB)
Trainable params: 260993 (1019.50 KB)
```

Non-trainable params: 0 (0.00 Byte)

Compiling Model

```
model.compile(
    loss = tf.keras.losses.BinaryCrossentropy(),
    metrics=['accuracy'],
    optimizer= tf.keras.optimizers.Adam(learning_rate=0.01, beta_1=0.9, beta_2=0.999)
)
```

Setting up Early Stopping & Reducing Learning Rate to reduce Overfitting

```
from keras.callbacks import EarlyStopping, ReduceLR0nPlateau
```

callback

```
es = EarlyStopping(
    patience=4,
    monitor = 'val_accuracy',
    restore_best_weights = True
)
```

```
lr = ReduceLR0nPlateau(
    patience = 2,
    monitor = 'val_loss',
    factor = 0.2,
    verbose = 1
)
```

Weights

```
weight_for_0 = (1 / neg) * (total / 2.0)
```

```
weight_for_1 = (1 / pos) * (total / 2.0)
```

```
class_weight = {0: weight_for_0, 1: weight_for_1}
```

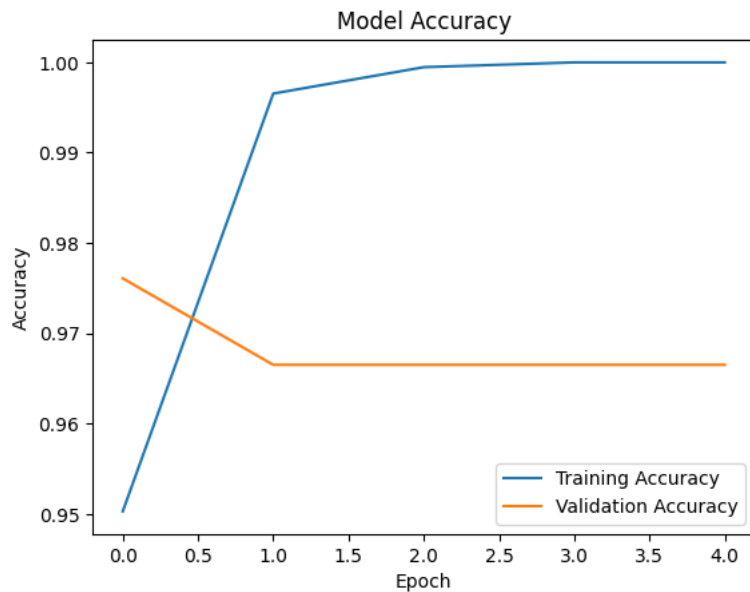
Training

```
history = model.fit(
    x=train_sequences,
    y=train_Y,
    validation_split=0.1,
    epochs=20,
    batch_size=32,
    callbacks = [lr, es],
    class_weight=class_weight
)
```

```
Epoch 1/20
118/118 [=====] - 26s 140ms/step - loss: 0.2216 - accuracy: 0.9503 - val_loss: 0.0719 - val_accurac
Epoch 2/20
118/118 [=====] - 7s 58ms/step - loss: 0.0167 - accuracy: 0.9965 - val_loss: 0.1164 - val_accuracy:
Epoch 3/20
118/118 [=====] - ETA: 0s - loss: 0.0021 - accuracy: 0.9995
Epoch 3: ReduceLR0nPlateau reducing learning rate to 0.0019999999552965165.
118/118 [=====] - 4s 29ms/step - loss: 0.0021 - accuracy: 0.9995 - val_loss: 0.1454 - val_accuracy:
Epoch 4/20
118/118 [=====] - 5s 36ms/step - loss: 5.2192e-04 - accuracy: 1.0000 - val_loss: 0.1433 - val_accur
Epoch 5/20
118/118 [=====] - ETA: 0s - loss: 4.5570e-04 - accuracy: 1.0000
Epoch 5: ReduceLR0nPlateau reducing learning rate to 0.0003999999724328518.
118/118 [=====] - 3s 24ms/step - loss: 4.5570e-04 - accuracy: 1.0000 - val_loss: 0.1445 - val_accur
```

Displaying Model Accuracy over Epochs

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



Evaluating Model

```
test_loss, test_accuracy = model.evaluate(test_sequences, test_Y)
print('Test Loss :', test_loss)
print('Test Accuracy :', test_accuracy)
```

```
44/44 [=====] - 0s 6ms/step - loss: 0.0503 - accuracy: 0.9885
Test Loss : 0.05031256750226021
Test Accuracy : 0.9885057210922241
```

function to predict messages based on model

(should return list containing prediction and label, ex. [0.008318834938108921, 'ham'])

```
intToLabel = {0: "ham", 1: "spam"}
```

```
def predict_message(pred_text):
    pred_text = remove_punctuations(pred_text)
    pred_text = remove_stopwords(pred_text)

    sequence = tokenizer.texts_to_sequences([pred_text])
    sequence = pad_sequences(sequence, maxlen=max_len)

    prediction = model.predict(sequence)[0]

    label = "ham"
    if prediction >= 0.5:
        label = "spam"

    print((prediction[0], label))
    return (prediction[0], label)
```

```
pred_text = "how are you doing today?"
```

```
prediction = predict_message(pred_text)
```

```
1/1 [=====] - 1s 1s/step
(0.019467225, 'ham')
```

Run this cell to test your function and model. Do not modify contents.

```
def test_predictions():
    test_messages = ["how are you doing today",
                     "sale today! to stop texts call 98912460324",
                     "i dont want to go. can we try it a different day? available sat",
                     "our new mobile video service is live. just install on your phone to start watching.",
                     "you have won £1000 cash! call to claim your prize.",
                     "i'll bring it tomorrow. don't forget the milk.",
                     "wow, is your arm alright. that happened to me one time too"
                    ]

    test_answers = ["ham", "spam", "ham", "spam", "spam", "ham", "ham"]
    passed = True

    for msg, ans in zip(test_messages, test_answers):
        prediction = predict_message(msg)
        if prediction[1] != ans:
            passed = False

    if passed:
        print("You passed the challenge. Great job!")
    else:
```