Humor Detection

Bradley Paliska, bp355

Data from Kaggle: 200K SHORT TEXTS FOR HUMOR DETECTION

https://www.kaggle.com/datasets/deepcontractor/200k-short-texts-for-humor-detection

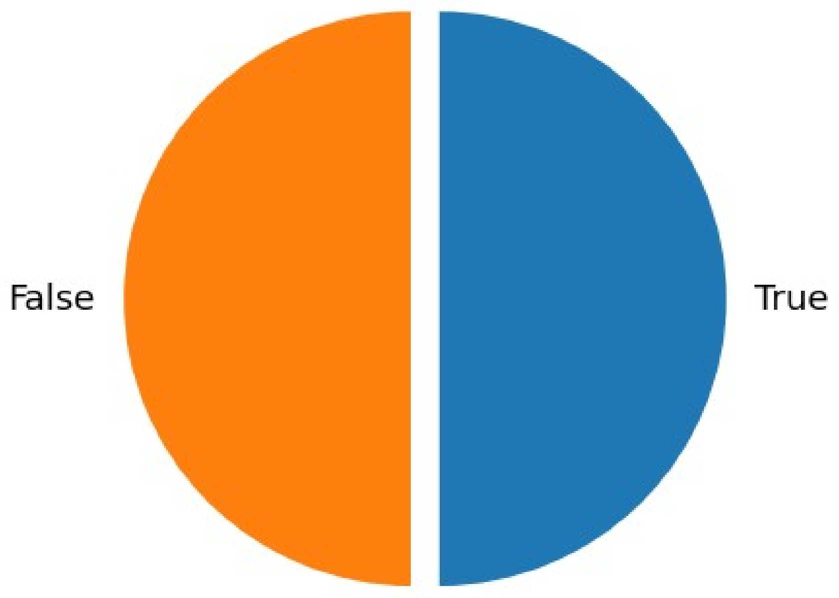
```
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
from wordcloud import WordCloud, STOPWORDS
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
import os
import random
from sklearn.model_selection import train_test_split
from transformers import BertTokenizer, BertForSequenceClassification,
Trainer, TrainingArguments
import torch
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import seaborn as sns

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.callbacks import *
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from collections import defaultdict

# Reading the CSV file into a DataFrame
df = pd.read_csv('dataset.csv')
print(df.head())
print(df.shape)
```

```
                                                text  humor
0  Joe biden rules out 2020 bid: 'guys, i'm not r...  False
1  Watch: darvish gave hitter whiplash with slow ...  False
2  What do you call a turtle without its shell? d...   True
3      5 reasons the 2016 election feels so personal  False
4  Pasco police shot mexican migrant from behind,...  False
(200000, 2)
```

```
df.humor.value_counts()
```

```
False    100000
True     100000
Name: humor, dtype: int64
```

```
plt.rcParams.update({'font.size': 13})
label = ['True', 'False']
count = [len(df[df['humor']==True]),
         len(df[df['humor']==False])]

plt.pie(count, labels = label,startangle=-90, explode=[0.1,0])
plt.show()
```



The data is evenly split: 100000 True and 100000 False

```
tokenizer = Tokenizer(filters='"&(),-/:;<=>[\\]_`{|}~\t\n0123456789',
                      lower=True, split=' ')
tokenizer.fit_on_texts(np.array(df['text']))
vocab_size = len(tokenizer.word_index) + 1
```

Initialize the Tokenizer with custom filters, set to lowercase and split on spaces, then fit it to the text data and calculate the vocabulary size

```
maxlen= 15
df['humor'] = df['humor'].apply(lambda x: {True:1, False:0}.get(x))
```

```
texts = np.array(df['text'])
texts = tokenizer.texts_to_sequences(texts)
for x in range(len(texts)):
    if len(texts[x])>maxlen:
        texts[x]=texts[x][:maxlen]
texts = pad_sequences(texts, maxlen=maxlen, dtype='float',
padding='post', value=0.0)
texts = np.array(texts)
labels = df['humor']
labels = np.array([float(j) for j in labels])

df.head() # 0= False and 1= True
```

```
                                                text  humor
0  Joe biden rules out 2020 bid: 'guys, i'm not r...      0
1  Watch: darvish gave hitter whiplash with slow ...      0
2  What do you call a turtle without its shell? d...      1
3      5 reasons the 2016 election feels so personal      0
4  Pasco police shot mexican migrant from behind,...      0
```

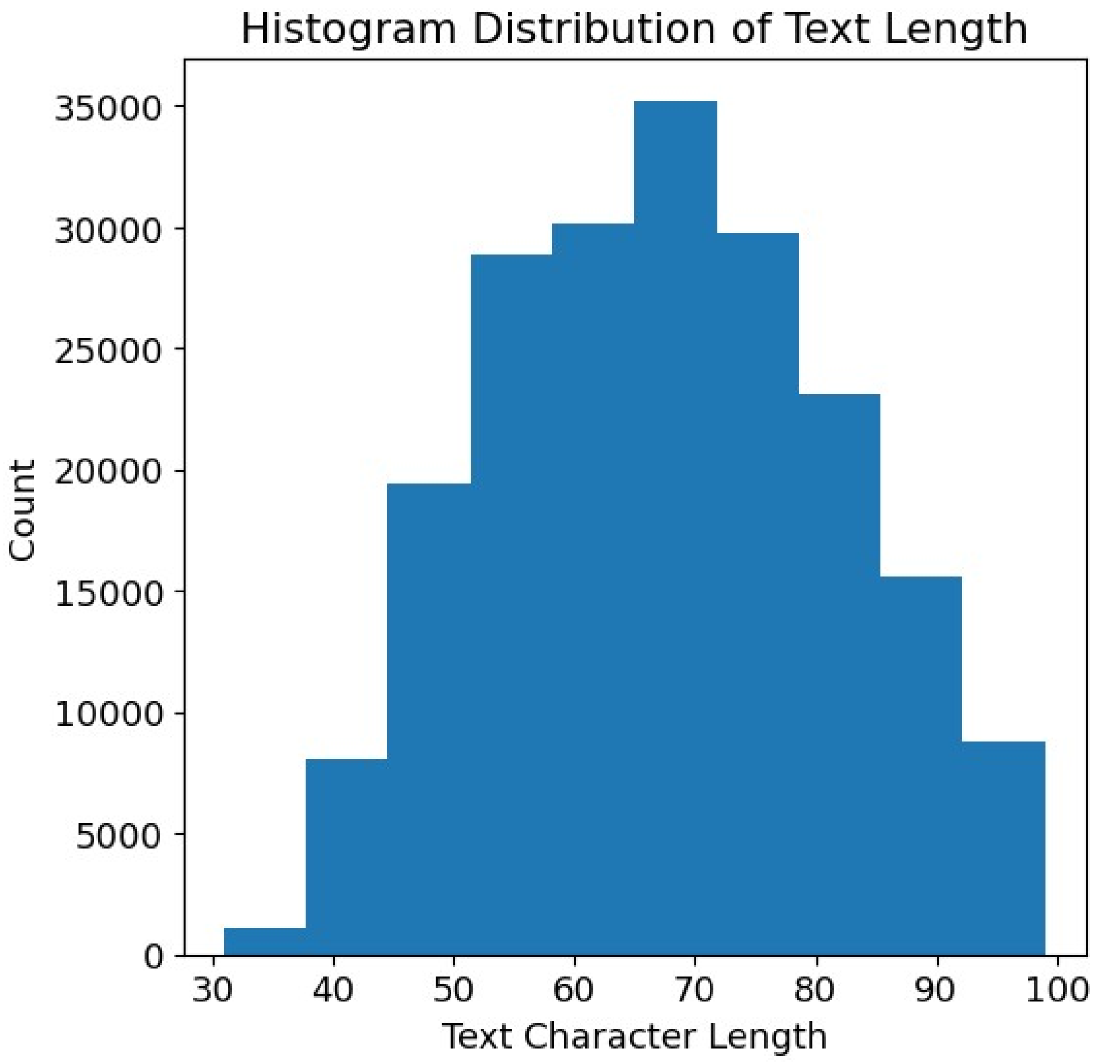
Set a maximum sequence length, convert humor labels to binary, tokenize and truncate/pad text sequences, and create arrays for text data and labels

```
# Step 1: Create a dictionary to store the count of occurrences for
each unique word
word_counts = defaultdict(int)

# Step 2: Iterate through the DataFrame and tokenize the text
for index, row in df.iterrows():
    text, humor = row['text'], row['humor']
    if humor == True:  # Considering only humorous words
        tokens = tokenizer.texts_to_sequences([text])[0]  # Use the
tokenizer you provided
        tokens = [tokenizer.index_word[token] for token in tokens]  #
Convert token indices back to words

        # Step 3: Update the dictionary with the count of occurrences
        for token in tokens:
            word_counts[token] += 1

# Step 4: Sort the words by their count of occurrences and display the
top results
sorted_words = sorted(word_counts.items(), key=lambda x: x[1],
reverse=True)

# Display the top 10 most humorous words by count
for word, count in sorted_words[:10]:
    print(f"{word}: {count}")
```

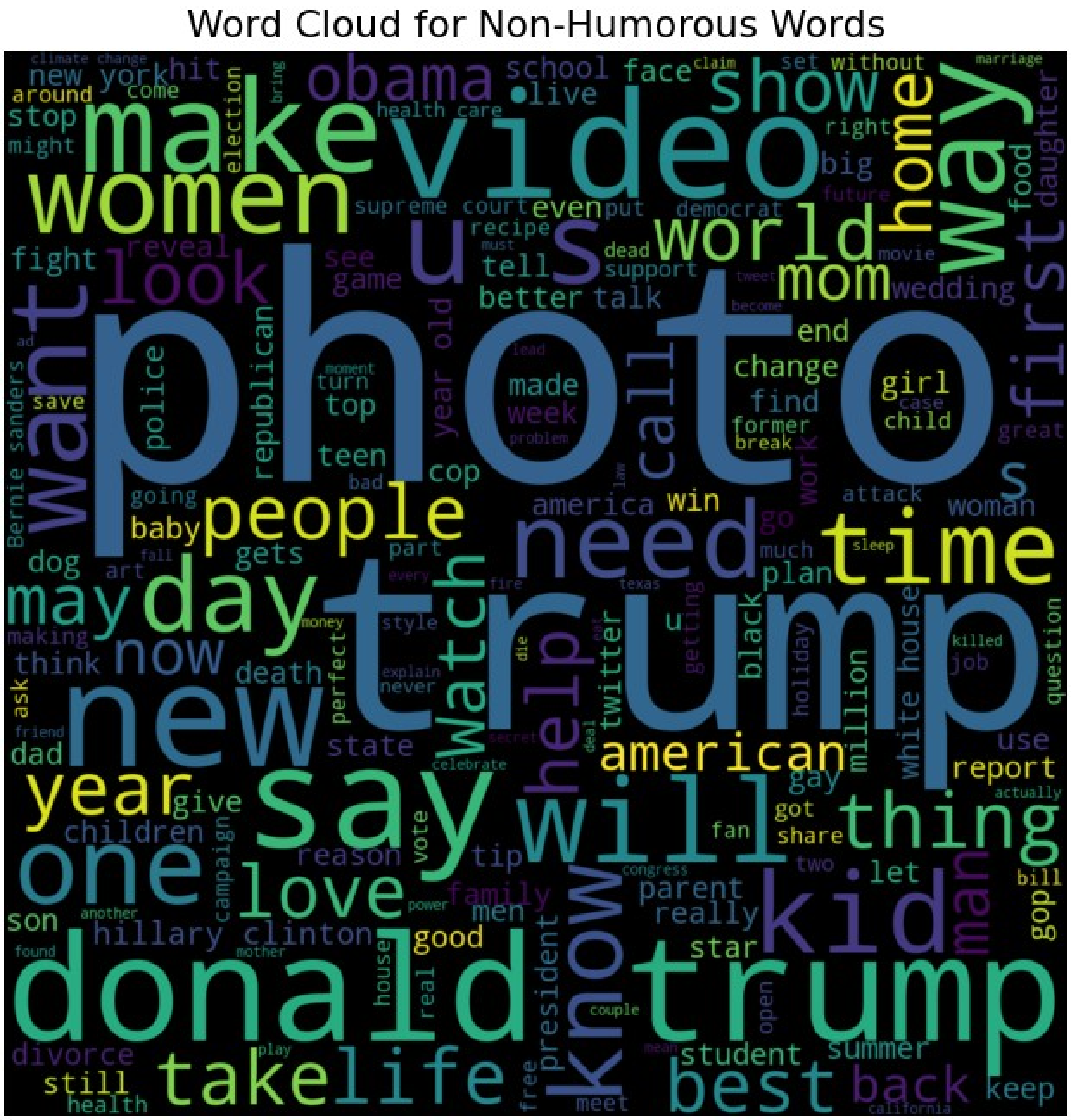
```
a: 59517
the: 50048
you: 28648
i: 26747
to: 24322
what: 22970
do: 18094
in: 15519
of: 15209
is: 14741
```

As expected a lot of basic words that are not very insighful.

```python
text_len = [len(x) for x in list(df['text'])]

fig, ax = plt.subplots(figsize=(6, 6))
plt.hist(text_len, bins=10)
plt.xlabel("Text Character Length")
plt.ylabel("Count")
plt.title("Histogram Distribution of Text Length")
plt.show()
```



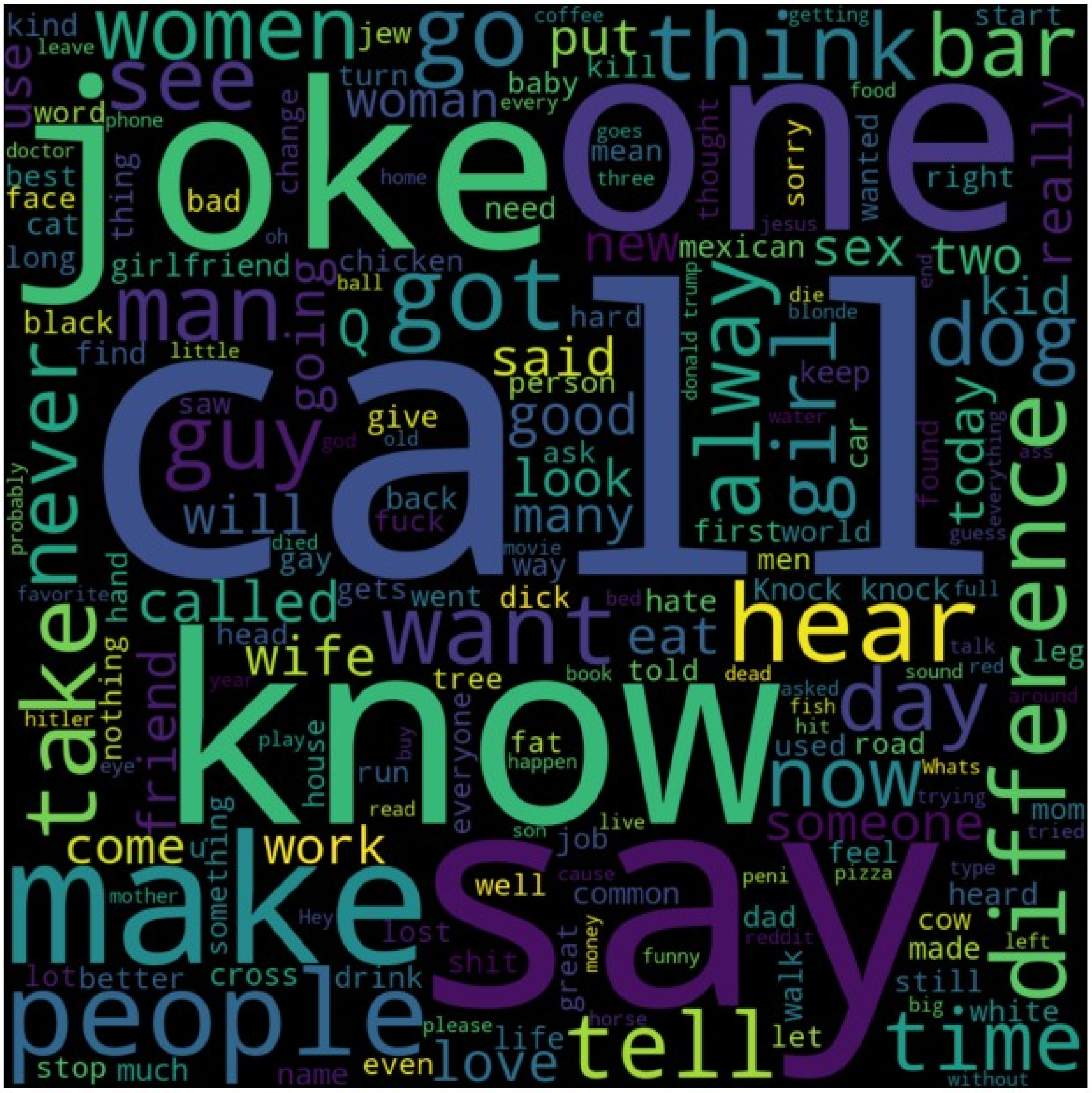
```
# List of additional stopwords to be excluded from the word cloud
list_words_add = ['none'] # wors to not include
STOPWORDS.update(list_words_add)

all_nonhumorous_words = ''
for idx, row in df.iterrows():
    if row['humor'] == 0:
        all_nonhumorous_words += ' ' + row['text'].strip()

wc = WordCloud(width=1024, height=1024, min_font_size=8,
stopwords=STOPWORDS).generate(all_nonhumorous_words)

# Display the word cloud
plt.figure(figsize=(8, 8))
plt.imshow(wc, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud for Non-Humorous Words')
plt.show()
```



```
list_words_add = ['none']
STOPWORDS.update(list_words_add)

all_humorous_words = ''
for idx, row in df.iterrows():
    if row['humor'] == 1:
        all_humorous_words += ' ' + row['text'].strip()
```

```
wc = WordCloud(width=1024, height=1024, min_font_size=8,
stopwords=STOPWORDS).generate(all_humorous_words)

# Display the word cloud
plt.figure(figsize=(8, 8))
plt.imshow(wc, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud for Humorous Words')
plt.show()
```

Word Cloud for Humorous Words



```
all_humorous_words[0:500]
```

```
" What do you call a turtle without its shell? dead. What is a pokemon
master's favorite kind of pasta? wartortellini! Why do native
americans hate it when it rains in april? because it brings
mayflowers. My family tree is a cactus, we're all pricks. How are
music and candy similar? we throw away the rappers. I just ended a 5
year relationship i'm fine, it wasn't my relationship :p Dating tip:
surprise your date! show up a day early. What do you call an
explanation of an asian cooking show? a wok"
```

```python
#Creating the Embedding Matrix
embedding_dim = 250

embedding_matrix = np.random.rand(vocab_size, embedding_dim)

print("Shape of the embedding matrix:", embedding_matrix.shape)
```

```
Shape of the embedding matrix: (111615, 250)
```

```python
x_train, x_val, y_train, y_val = train_test_split(texts, labels,
test_size=0.2, random_state=42)
```

```python
#Building the model
model = Sequential()

model.add(Input(shape=(maxlen)))
model.add(Embedding(vocab_size, 250, weights=[embedding_matrix],
input_length=maxlen, trainable=False))
model.add(Bidirectional(LSTM(64, activation='relu', dropout=0.15,
return_sequences=True), merge_mode='concat'))
model.add(TimeDistributed(Dense(64, activation='relu')))
model.add(LSTM(256, activation='relu', dropout=0.15,
return_sequences=False))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(2, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

model.summary()
```

```
2023-05-14 12:03:52.623884: I
tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow
binary is optimized with oneAPI Deep Neural Network Library (oneDNN)
to use the following CPU instructions in performance-critical
operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the
appropriate compiler flags.
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 15, 250)           27903750

 bidirectional (Bidirectiona  (None, 15, 128)          161280
 l)

 time_distributed (TimeDistr  (None, 15, 64)           8256
 ibuted)

 lstm_1 (LSTM)               (None, 256)               328704

 flatten (Flatten)           (None, 256)               0

 dense_1 (Dense)             (None, 256)               65792

 dropout (Dropout)           (None, 256)               0

 dense_2 (Dense)             (None, 2)                 514

=================================================================
Total params: 28,468,296
Trainable params: 564,546
Non-trainable params: 27,903,750
_________________________________________________________________

# Train the model for 20 epochs, displaying the loss, accuracy,
validation loss, and validation accuracy for each epoch
epochs = 20
batch_size = 800

# Define the ModelCheckpoint callback to save the best model
mc = ModelCheckpoint('model.h5',
monitor='val_sparse_categorical_accuracy', mode='max', verbose=1,
save_best_only=True)

# Train the model
history = model.fit(x_train, y_train, epochs=epochs,
batch_size=batch_size, validation_data=(x_val, y_val), callbacks=[mc])

Epoch 1/20
200/200 [==============================] - ETA: 0s - loss: 0.5201 -
accuracy: 0.7446WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 67s 329ms/step - loss:
0.5201 - accuracy: 0.7446 - val_loss: 0.3485 - val_accuracy: 0.8535
Epoch 2/20
200/200 [==============================] - ETA: 0s - loss: 0.3618 -
```

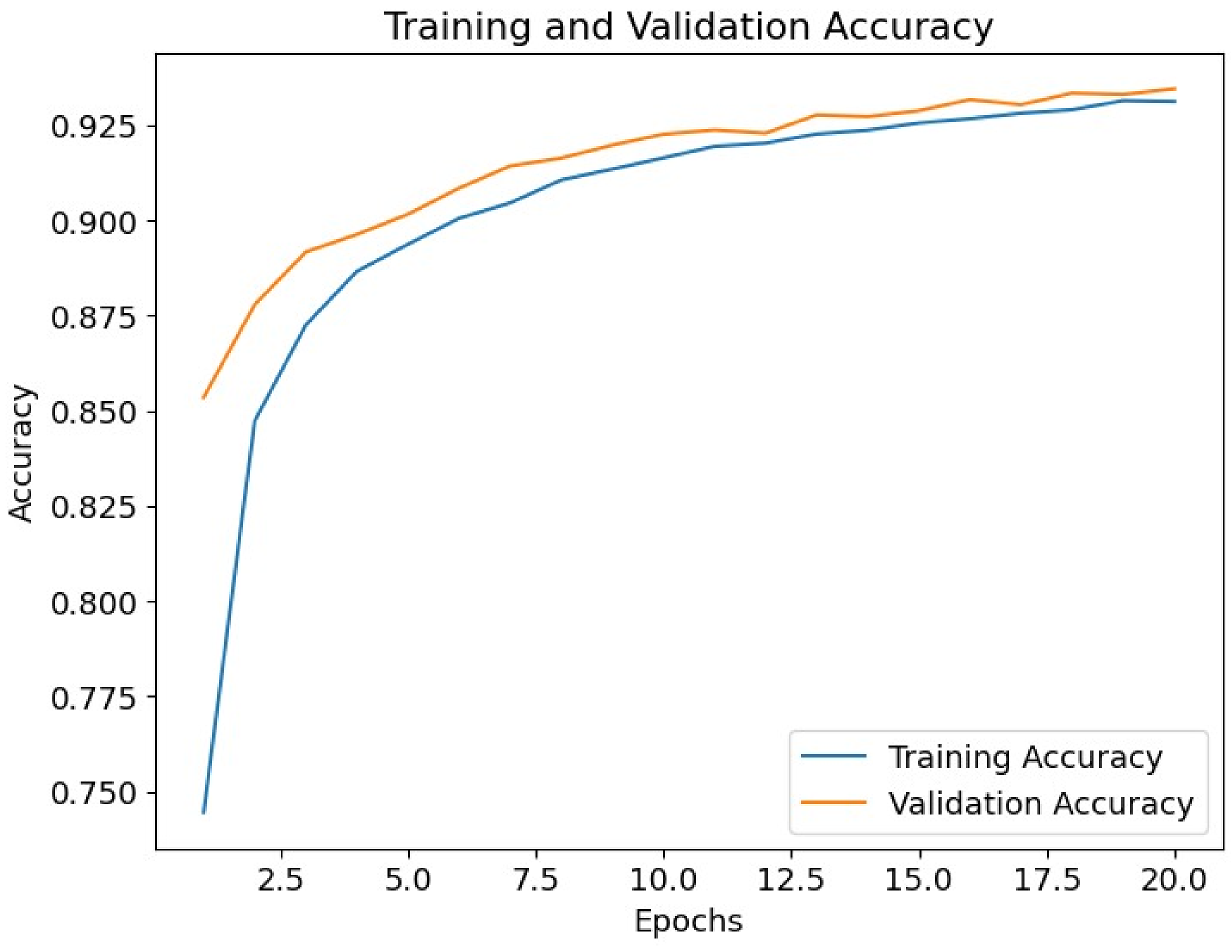
```
accuracy: 0.8473WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 65s 324ms/step - loss:
0.3618 - accuracy: 0.8473 - val_loss: 0.2997 - val_accuracy: 0.8778
Epoch 3/20
200/200 [==============================] - ETA: 0s - loss: 0.3111 -
accuracy: 0.8724WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 63s 317ms/step - loss:
0.3111 - accuracy: 0.8724 - val_loss: 0.2735 - val_accuracy: 0.8916
Epoch 4/20
200/200 [==============================] - ETA: 0s - loss: 0.2812 -
accuracy: 0.8865WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 63s 317ms/step - loss:
0.2812 - accuracy: 0.8865 - val_loss: 0.2554 - val_accuracy: 0.8962
Epoch 5/20
200/200 [==============================] - ETA: 0s - loss: 0.2659 -
accuracy: 0.8936WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 66s 329ms/step - loss:
0.2659 - accuracy: 0.8936 - val_loss: 0.2446 - val_accuracy: 0.9015
Epoch 6/20
200/200 [==============================] - ETA: 0s - loss: 0.2504 -
accuracy: 0.9004WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 65s 327ms/step - loss:
0.2504 - accuracy: 0.9004 - val_loss: 0.2293 - val_accuracy: 0.9084
Epoch 7/20
200/200 [==============================] - ETA: 0s - loss: 0.2395 -
accuracy: 0.9045WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 66s 332ms/step - loss:
0.2395 - accuracy: 0.9045 - val_loss: 0.2147 - val_accuracy: 0.9142
Epoch 8/20
200/200 [==============================] - ETA: 0s - loss: 0.2263 -
accuracy: 0.9105WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 65s 325ms/step - loss:
0.2263 - accuracy: 0.9105 - val_loss: 0.2107 - val_accuracy: 0.9162
Epoch 9/20
200/200 [==============================] - ETA: 0s - loss: 0.2202 -
accuracy: 0.9134WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 63s 317ms/step - loss:
0.2202 - accuracy: 0.9134 - val_loss: 0.2047 - val_accuracy: 0.9197
Epoch 10/20
200/200 [==============================] - ETA: 0s - loss: 0.2128 -
accuracy: 0.9163WARNING:tensorflow:Can save best model only with
```

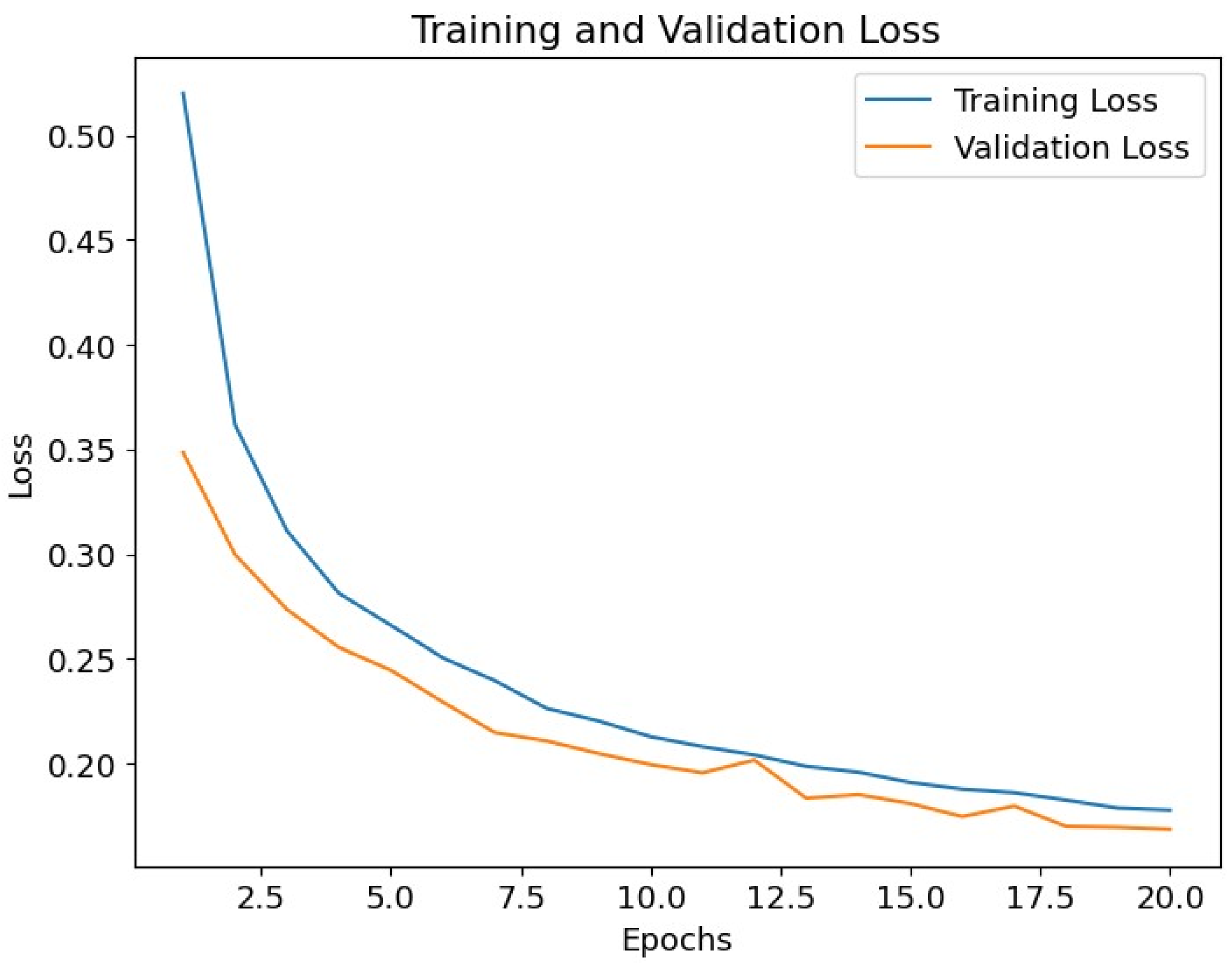
```
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 63s 317ms/step - loss:
0.2128 - accuracy: 0.9163 - val_loss: 0.1994 - val_accuracy: 0.9225
Epoch 11/20
200/200 [==============================] - ETA: 0s - loss: 0.2080 -
accuracy: 0.9194WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 65s 326ms/step - loss:
0.2080 - accuracy: 0.9194 - val_loss: 0.1955 - val_accuracy: 0.9237
Epoch 12/20
200/200 [==============================] - ETA: 0s - loss: 0.2040 -
accuracy: 0.9202WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 63s 316ms/step - loss:
0.2040 - accuracy: 0.9202 - val_loss: 0.2015 - val_accuracy: 0.9228
Epoch 13/20
200/200 [==============================] - ETA: 0s - loss: 0.1985 -
accuracy: 0.9226WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 63s 317ms/step - loss:
0.1985 - accuracy: 0.9226 - val_loss: 0.1833 - val_accuracy: 0.9276
Epoch 14/20
200/200 [==============================] - ETA: 0s - loss: 0.1957 -
accuracy: 0.9236WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 63s 315ms/step - loss:
0.1957 - accuracy: 0.9236 - val_loss: 0.1851 - val_accuracy: 0.9272
Epoch 15/20
200/200 [==============================] - ETA: 0s - loss: 0.1909 -
accuracy: 0.9255WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 65s 325ms/step - loss:
0.1909 - accuracy: 0.9255 - val_loss: 0.1808 - val_accuracy: 0.9287
Epoch 16/20
200/200 [==============================] - ETA: 0s - loss: 0.1877 -
accuracy: 0.9266WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 64s 322ms/step - loss:
0.1877 - accuracy: 0.9266 - val_loss: 0.1745 - val_accuracy: 0.9316
Epoch 17/20
200/200 [==============================] - ETA: 0s - loss: 0.1860 -
accuracy: 0.9280WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 65s 324ms/step - loss:
0.1860 - accuracy: 0.9280 - val_loss: 0.1796 - val_accuracy: 0.9303
Epoch 18/20
200/200 [==============================] - ETA: 0s - loss: 0.1824 -
accuracy: 0.9289WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
```

```
200/200 [==============================] - 69s 345ms/step - loss:
0.1824 - accuracy: 0.9289 - val_loss: 0.1699 - val_accuracy: 0.9333
Epoch 19/20
200/200 [==============================] - ETA: 0s - loss: 0.1786 -
accuracy: 0.9313WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 63s 315ms/step - loss:
0.1786 - accuracy: 0.9313 - val_loss: 0.1695 - val_accuracy: 0.9329
Epoch 20/20
200/200 [==============================] - ETA: 0s - loss: 0.1776 -
accuracy: 0.9311WARNING:tensorflow:Can save best model only with
val_sparse_categorical_accuracy available, skipping.
200/200 [==============================] - 63s 314ms/step - loss:
0.1776 - accuracy: 0.9311 - val_loss: 0.1684 - val_accuracy: 0.9344
```

The model was trained for 20 epochs, and during the training process, we can observe that both the training loss and validation loss decreased. This indicates that the model was learning effectively from the data. Moreover, the training and validation accuracies increased, reaching around 93.11% and 93.44%, respectively, by the end of the training. This suggests that the model has a good performance in predicting humor based on the given text data.

```python
# Get the training and validation accuracy
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

# Get the training and validation loss
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Plot the training and validation accuracy
plt.figure(figsize=(8, 6))
plt.plot(range(1, epochs+1), train_accuracy, label='Training
Accuracy')
plt.plot(range(1, epochs+1), val_accuracy, label='Validation
Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot the training and validation loss
plt.figure(figsize=(8, 6))
plt.plot(range(1, epochs+1), train_loss, label='Training Loss')
plt.plot(range(1, epochs+1), val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





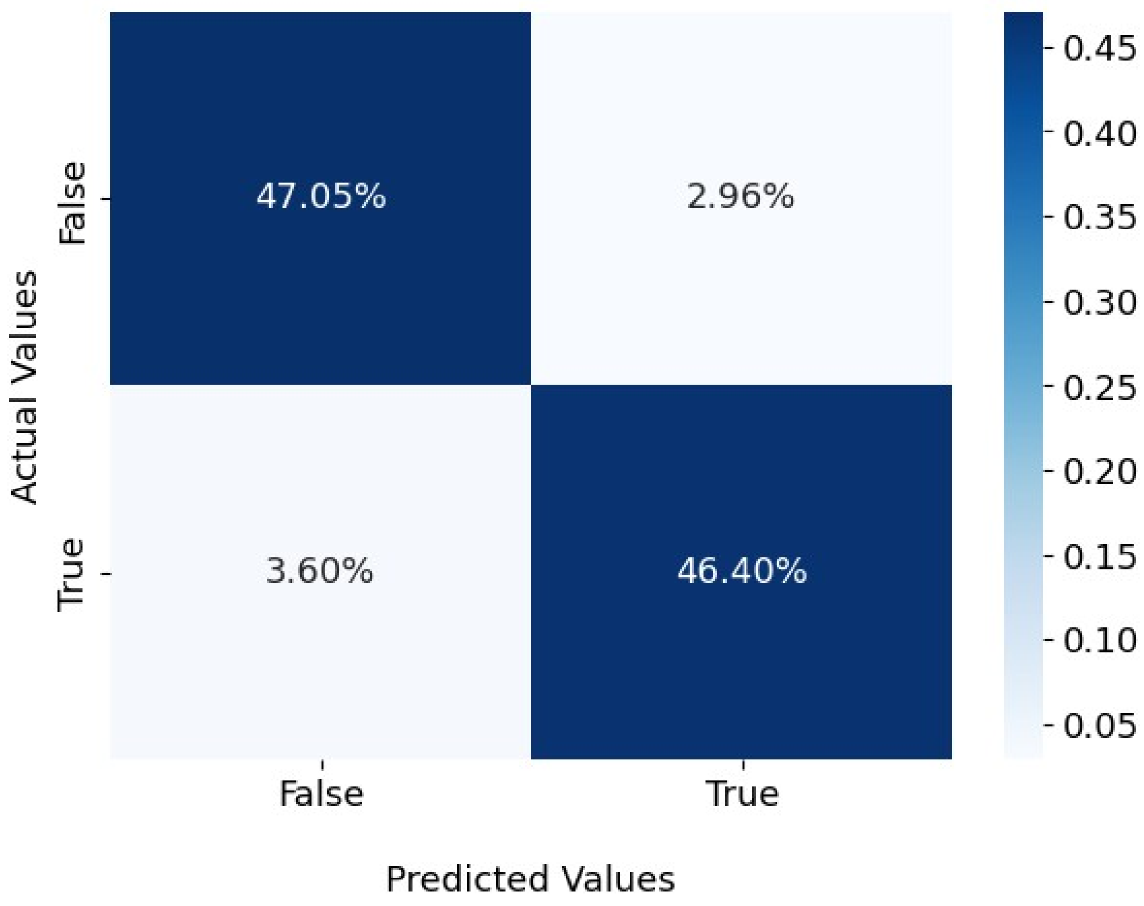
```
# Evaluate the model on the test set and display the precision,
recall, f1-score, and accuracy for each class (humorous and non-
humorous)

decode_label = {0: 'False', 1: 'True'}

y_pred = []
y_true = []

pred = model.predict(x_val)
pred = np.argmax(pred, axis=-1)

y_pred = [decode_label[int(i)] for i in pred]
y_true = [decode_label[int(i)] for i in y_val]

print(classification_report(y_true, y_pred, digits=3))
```

```
1250/1250 [==============================] - 14s 11ms/step
              precision    recall  f1-score   support

       False      0.929     0.941     0.935     20001
```

```
        True      0.940     0.928     0.934     19999

    accuracy                          0.934     40000
   macro avg      0.935     0.934     0.934     40000
weighted avg      0.935     0.934     0.934     40000
```

The evaluation of the model on the test set shows high precision, recall, and f1-score for both the humorous (True) and non-humorous (False) classes, with values around 0.92 to 0.94. The overall accuracy of the model is 93.4%, indicating that it performs well in classifying text as humorous or non-humorous. The macro and weighted averages of the precision, recall, and f1-score are also around 0.935, further confirming the model's good performance on the test data.

```
# Plot a confusion matrix heatmap for the model's predictions,
displaying the percentage of correctly and incorrectly classified
instances for humorous and non-humorous classes

cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(7, 5))

ax = sns.heatmap(cm / np.sum(cm), fmt='.2%', annot=True, cmap='Blues')

ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values')

ax.xaxis.set_ticklabels(['False', 'True'])
ax.yaxis.set_ticklabels(['False', 'True'])

plt.show()
```



The model correctly classified 47.05% of the non-humorous instances (True Negative, TN) and 46.40% of the humorous instances (True Positive, TP). The model misclassified 2.96% of the instances as humorous when they were actually non-humorous (False Positive, FP), and 3.60% of the instances as non-humorous when they were actually humorous (False Negative, FN). The confusion matrix shows that the model has a relatively low rate of misclassification, which is consistent with the high precision, recall, and f1-score values observed earlier.

```
# Define a function to preprocess input text, make predictions using
the trained model, and return the predicted humor label (True or
False)

def predict(text):
    text = tokenizer.texts_to_sequences([text])
    if len(text) > maxlen:
        text = text[:maxlen]
    text = pad_sequences(text, maxlen=maxlen, dtype='float',
padding='post', value=0.0)
    text = np.array(text)
    pred = model.predict(text)
    pred = np.argmax(pred, axis=-1)
    decode_label = {0: 'False', 1: 'True'}
    pred = decode_label[pred[0]]
    return pred

text = "If you live each day as if it was your last, some day you'll
most certainly be right." #Steve Jobs
pred = predict(text)
print("Text:",text)
print('Humor detected: ',pred)

1/1 [==============================] - 0s 155ms/step
Text: If you live each day as if it was your last, some day you'll
most certainly be right.
Humor detected:  True

text = "he cat slept peacefully on the warm windowsill, enjoying the
afternoon sun."
pred = predict(text)
print("Text:",text)
print('Humor detected: ',pred)

1/1 [==============================] - 0s 19ms/step
Text: he cat slept peacefully on the warm windowsill, enjoying the
afternoon sun.
Humor detected:  False

text = "The diligent gardener carefully pruned the rose bushes,
creating a beautiful and well-manicured garden."
pred = predict(text)
print("Text:",text)
print('Humor detected: ',pred)

1/1 [==============================] - 0s 27ms/step
Text: The diligent gardener carefully pruned the rose bushes, creating
a beautiful and well-manicured garden.
Humor detected:  False

text = "I told my wife she was drawing her eyebrows too high; she
looked surprised."
pred = predict(text)
print("Text:",text)
print('Humor detected: ',pred)

1/1 [==============================] - 0s 19ms/step
Text: I told my wife she was drawing her eyebrows too high; she looked
surprised.
Humor detected:  True

text = "Why don't some couples go to the gym? Because some
relationships don't work out!"
pred = predict(text)
```

```
print("Text:",text)
print('Humor detected: ',pred)

1/1 [==============================] - 0s 21ms/step
Text: Why don't some couples go to the gym? Because some relationships
don't work out!
Humor detected:  True

text = "Did you see the magnificent sunset tonight? The colors were
absolutely breathtaking!" #no-humor
pred = predict(text)
print("Text:",text)
print('Humor detected: ',pred)

1/1 [==============================] - 0s 38ms/step
Text: Did you see the magnificent sunset tonight. The colors were
absolutely breathtaking.
Humor detected:  True
```

It looks like you can fool the model with sentences that have the same structure as a joke (question and answer).

```
text = "Three elephants fell off a cliff, which was quite a surprise,
considering it was a calendar"
pred = predict(text)
print("Text:",text)
print('Humor detected: ',pred)

1/1 [==============================] - 0s 20ms/step
Text: Three elephants fell off a cliff, which was quite a surprise,
considering it was a calendar
Humor detected:  True
```

In this notebook, we have successfully implemented a deep learning model to detect humor in text. We preprocessed the text data, tokenized it, and created an embedding matrix for the words. We then built, trained, and evaluated a Bi-directional LSTM model, achieving high precision, recall, and f1-score values. Finally, we created a function to make predictions on new text inputs, allowing us to apply our trained model to real-world applications.

With the promising results obtained from this notebook, further improvements could be explored, such as fine-tuning the model's hyperparameters, incorporating additional features, or experimenting with different model architectures.