SARCASM DETECTION

A case study comparison of BiLSTM, BERT and DistilBERT that..

- *unveils the power of Natural Language Processing in the HuggingFace Ecosystem*
- *showcases the power of Generative Pretrained Transformers*
    - *Configuring and testing CustomGPT on ChatGPT/GPT4 to detect sarcasm (Oh Great, Another Monday)*
- *Explores the scope for future work esp. in Indic languages such as Tamil.*

மெல்ல மெல்ல சுவத்துக்கு வலிக்கப்போகுது!!

AI & DS 2022 (SEM 3 NLP Case Study)

Team Members – Group 10

1. Karthik G S (**PA2212044010007**)
2. Rajesh P (**PA2212044010054**)
3. Venu K S (**PA2212044010023**)

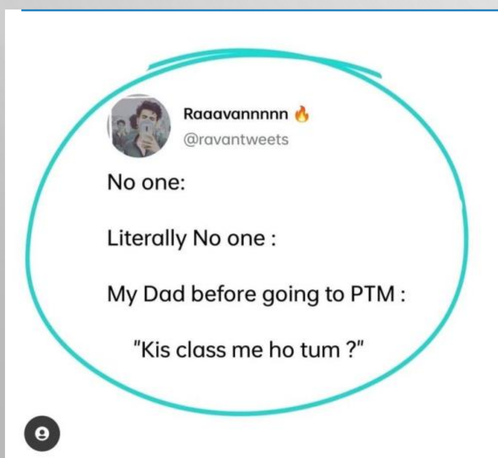
Submitted: Tuesday, December 19th, 2023

SARCASM / NOT-SARCASM

INTRODUCTION

- Sarcasm is a nuanced form of language where the speaker explicitly states the opposite of what he/she states while deliberately means the opposite
- Sarcasm identification in online communications from social media sites, discussion forums, and e-commerce websites has become essential for fake news detection, sentiment analysis, opinion mining, and detecting of online trolls and cyber bullies

Sarcastic Tweets – Examples



CHALLENGES IN SARCASM DETECTION

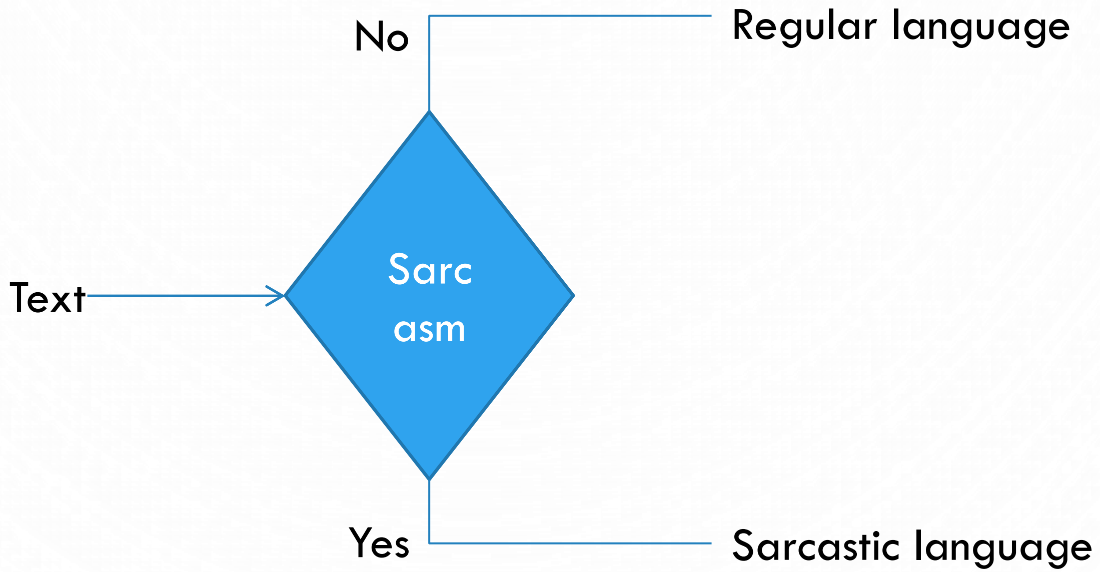
- Context Dependence:
  - Understanding the context in which a statement is made is crucial for determining whether it is sarcastic.
- Ambiguity:
  - Sarcasm can introduce ambiguity, which makes it challenging for models to accurately identify sarcastic statements.
- Tone and Intonation:
  - Sarcasm is often conveyed through tone, intonation, and delivery which is lacking in written text
- Short and Sparse Context:
  - In social media and online communication, sarcastic remarks are often short and lack detailed context. Models must be able to infer meaning from limited textual information.
- Subjectivity and Subjective Perception:
  - Sarcasm is subjective, and individuals may interpret the same statement differently.

PROBLEM STATEMENT

- Addressing these challenges often requires the use of advanced natural language processing (NLP) techniques, including deep learning models, that can capture subtle contextual cues and patterns in language.
- Additionally, leveraging large and diverse datasets is crucial for training models that generalize well across different contexts and linguistic variations.

OBJECTIVE

- Overall objective is to build a sentiment analysis system capable of handling sarcasm
  - To develop a model that can accurately understand and interpret the sentiment expressed in text, even when it involves sarcasm.
- Demonstrate the sarcasm detection using multiple NLP approaches (different NLP techniques) and compare their results



DATASET USED

For the case study , we have used the **News-Headlines-Dataset-For-Sarcasm-Detection** (https://github.com/rishabhmisra/News-Headlines-Dataset-For-Sarcasm-Detection)

This Headlines dataset for Sarcasm Detection is collected from two news websites:-

- The Onion, which aims at producing sarcastic versions of current events
- The Huffington Post, which is a regular news site

Data consists of three attributes:

- is_sarcastic: 1 if the record is sarcastic otherwise 0
- headline: the headline of the news article
- article_link: link to the original news article.

IT IS A FAIRLY BALANCED, WELL-CURATED DATASET!

SARCASM IN OUR DATASET

	is_sarcastic	headline	article_link
0	1	thirtysomething scientists unveil doomsday clock of hair loss	https://www.theonion.com/thirtysomething-scientists-unveil-doomsday-clock-of-hai-1819586205
1	0	dem rep. totally nails why congress is falling short on gender, racial equality	https://www.huffingtonpost.com/entry/donna-edwards-inequality_us_57455f7fe4b055bb1170b207
2	0	eat your veggies: 9 deliciously different recipes	https://www.huffingtonpost.com/entry/eat-your-veggies-9-delici_b_8899742.html
3	1	inclement weather prevents liar from getting to work	https://local.theonion.com/inclement-weather-prevents-liar-from-getting-to-work-1819576031
4	1	mother comes pretty close to using word 'streaming' correctly	https://www.theonion.com/mother-comes-pretty-close-to-using-word-streaming-cor-1819575546
...	...	...	...
28614	1	jews to celebrate rosh hashasha or something	https://www.theonion.com/jews-to-celebrate-rosh-hashasha-or-something-1819564013
28615	1	internal affairs investigator disappointed conspiracy doesn't go all the way to the top	https://local.theonion.com/internal-affairs-investigator-disappointed-conspiracy-d-1819568967
28616	0	the most beautiful acceptance speech this week came from a queer korean	https://www.huffingtonpost.com/entry/andrew-ahn-independent-spirit-awards_us_58b44741e4b060480e0a2c30
28617	1	mars probe destroyed by orbiting spielberg-gates space palace	https://www.theonion.com/mars-probe-destroyed-by-orbiting-spielberg-gates-space-1819564363
28618	1	dad clarifies this not a food stop	https://www.theonion.com/dad-clarifies-this-not-a-food-stop-1819576557

28619 rows × 3 columns

TECHNIQUES USED: MODEL FEATURES AND SUITABILITY

1	2	3	4
Trained by us on sarcasm detection dataset			**Pre-trained from HF**
Bidirectional LSTM architecture with GLOVE embeddings (100d,300d) – 12 million parameters	BERT (bert-base-uncased) from HF (110 million parameters)	DistilBERT (distilbert-base-uncased) from HF (60 million parameters)	helinivan/english-sarcasm-detector (bert-base-uncased)
• Ability to capture sequential context, • Address long-term dependencies • utilize pre-trained semantic embeddings • Fine-tune for domain-specific nuances, and achieve contextual awareness.	• bi-directional transformer for pre-training over a lot of unlabeled textual data • Computational efficiency • Contextual understanding • Transfer learning benefits • Fine-tuning adaptability • Compatibility with TensorFlow, making it a suitable choice for real-world applications.	• learns a distilled (approximate) version of BERT – smaller NN approximating larger one. • retaining 97% performance using only half the parameters • No token-type embeddings, pooler and retains only half of the layers.	• Same as other transformer models • Out of box performance being a pre-trained model. • Carefully pre-trained from BERT BASE uncased on the same sarcasm detection data • Expanded to multi-lingual sarcasm using further training data

TECHNIQUES USED

STEPS FOLLOWED BY EACH TECHNIQUE IS GIVEN BELOW

Bidirectional LSTM architecture with GLOVE embeddings

1. Load and Prepare Data
2. Tokenization and Padding
3. Load GLoVE Word Embeddings
4. Create Embedding Matrix
5. Build Bidirectional LSTM Model
6. Compile the Model
7. Train the Model
8. Evaluate Model

BERT/DistilBERT

1. Load and Prepare Data
2. Tokenization
3. Load Pre-trained DistilBERT Model
4. Set up Model for Sarcasm
5. Train the Model
6. Evaluate Model
7. Load Model & Predict

helinivan/english-sarcasm-detector

1. Load and Prepare Data
2. implementing a function for sarcasm detection using a pre-trained BERT model and tokenizer
3. Load Pre-trained BERT Model for Sequence Classification
4. Predict & Evaluate Model

CODES & OUTPUT

IMPORT LIBRARIES (GENERAL IMPORTS)

```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from tqdm import tqdm
from keras.preprocessing.text import one_hot, Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Flatten, GlobalMaxPooling1D, Conv1D, LSTM,Embedding, Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

import tensorflow as tf
```

```
import requests
import json
```

CODES & OUTPUT FOR MODEL 1: BIDIRECTIONAL LSTM ARCHITECTURE WITH GLOVE EMBEDDING

Load and Prepare Data

```
[ ] import requests
    import json

    # Replace the URL below with the raw URL of your JSON file
    url = 'https://raw.githubusercontent.com/rishabhmisra/News-Headlines-Dataset-For

    # Fetch the file from the URL
    response = requests.get(url)

    # Check if the request was successful
    if response.status_code == 200:
        # Split the response text by newlines
        lines = response.text.splitlines()

        # Parse each line as a JSON object and append to a list
        data = [json.loads(line) for line in lines]

        # Create a DataFrame from the list
        df = pd.DataFrame(data)

        # Display the first few rows of the DataFrame
        print(df.head())
    else:
        print(f"Failed to retrieve the file: {response.status_code}")
```

```
df
```

	is_sarcastic	headline	article_link
0	1	thirtysomething scientists unveil doomsday clo...	https://www.theonion.com/thirtysomething-scien...
1	0	dem rep. totally nails why congress is falling...	https://www.huffingtonpost.com/entry/donna-edw...
2	0	eat your veggies: 9 deliciously different recipes	https://www.huffingtonpost.com/entry/eat-your-...
3	1	inclement weather prevents liar from getting t...	https://local.theonion.com/inclement-weather-p...
4	1	mother comes pretty close to using word 'strea...	https://www.theonion.com/mother-comes-pretty-c...
...	...	...	...
28614	1	jews to celebrate rosh hashasha or something	https://www.theonion.com/jews-to-celebrate-ros...
28615	1	internal affairs investigator disappointed con...	https://local.theonion.com/internal-affairs-in...
28616	0	the most beautiful acceptance speech this week...	https://www.huffingtonpost.com/entry/andrew-ah...
28617	1	mars probe destroyed by orbiting spielberg-gat...	https://www.theonion.com/mars-probe-destroyed-...
28618	1	dad clarifies this not a food stop	https://www.theonion.com/dad-clarifies-this-no...

```
[ ] # prompt: df1 is_sarcastic and headline from df

    df1 = df[['is_sarcastic', 'headline']]
```

```
[ ] # prompt: import train test split and use 20% data for test

    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(df1.headline, df1.is_sarcastic, test_size=0.10,random_state=42)
```

```
[ ] y_test.shape

    (2862,)
```

```
[ ] y_test.value_counts()

    0    1506
    1    1356
    Name: is_sarcastic, dtype: int64
```

CODES & OUTPUT FOR MODEL 1: BIDIRECTIONAL LSTM ARCHITECTURE WITH GLOVE EMBEDDING

Tokenisation & Padding

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

[ ] %cd /content/drive/MyDrive

    /content/drive/MyDrive

[ ] word_tokenizer = Tokenizer()
    word_tokenizer.fit_on_texts(X_train)

    X_train1 = word_tokenizer.texts_to_sequences(X_train)
    X_test1 = word_tokenizer.texts_to_sequences(X_test)
```

```
# Padding all reviews to fixed length 100, truncate

maxlen = 100

X_train2 = pad_sequences(X_train1, padding='post', maxlen=maxlen) # pre or post
X_test2 = pad_sequences(X_test1, padding='post', maxlen=maxlen)

vocab_length = len(word_tokenizer.word_index)+1
```

CODES & OUTPUT FOR MODEL 1: BIDIRECTIONAL LSTM ARCHITECTURE WITH GLOVE EMBEDDING

Loading Glove & Create Embedding matrix

```
# Load GloVe word embeddings and create an Embeddings Dictionary

from numpy import asarray
from numpy import zeros

embeddings_dictionary = dict()
glove_file = open('/content/drive/MyDrive/mtech/glove.6B.100d.txt', encoding="utf8")

for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:], dtype='float32')
    embeddings_dictionary [word] = vector_dimensions
glove_file.close()
```

```
[ ] # Create Embedding Matrix having 100 columns
    # Containing 100-dimensional GloVe word embeddings for all words in our corpus.

    embedding_matrix = zeros((vocab_length, 100))

    for word, index in word_tokenizer.word_index.items():
        embedding_vector = embeddings_dictionary.get(word)
        if embedding_vector is not None:
            embedding_matrix[index] = embedding_vector # 92394 rows, 100 columns of embed numberical values
```

CODES & OUTPUT FOR MODEL 1: BIDIRECTIONAL LSTM ARCHITECTURE WITH GLOVE EMBEDDING

Bidirectional LSTM Model

```
[ ]
    lstm_bidirectional_model = Sequential()
    lstm_bidirectional_model.add(embedding_layer)  # Assuming embedding_layer is already defined as
    lstm_bidirectional_model.add(Bidirectional(LSTM(92)))
    lstm_bidirectional_model.add(Dense(1, activation='sigmoid'))
```

```
[ ] # Model compiling

    lstm_bidirectional_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
    print(lstm_bidirectional_model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 100)          2939500

 bidirectional (Bidirection  (None, 184)               142048
 al)

 dense (Dense)               (None, 1)                 185

=================================================================
Total params: 3081733 (11.76 MB)
Trainable params: 142233 (555.60 KB)
Non-trainable params: 2939500 (11.21 MB)
```

CODES & OUTPUT FOR MODEL 1: BIDIRECTIONAL LSTM ARCHITECTURE WITH GLOVE EMBEDDING

Model Training & Evaluation

```
lstm_bidirectional_model_history = lstm_bidirectional_model.fit(X_train2, y_train, batch_size=128, epochs=6, verbose=1, validation_split=0.2)
```

```
Epoch 1/6
161/161 [==============================] - 13s 33ms/step - loss: 0.5202 - acc: 0.7375 - val_loss: 0.4250 - val_acc: 0.8108
Epoch 2/6
161/161 [==============================] - 4s 24ms/step - loss: 0.4079 - acc: 0.8148 - val_loss: 0.3811 - val_acc: 0.8344
Epoch 3/6
161/161 [==============================] - 3s 16ms/step - loss: 0.3527 - acc: 0.8441 - val_loss: 0.3487 - val_acc: 0.8455
Epoch 4/6
161/161 [==============================] - 2s 13ms/step - loss: 0.3122 - acc: 0.8651 - val_loss: 0.3316 - val_acc: 0.8531
Epoch 5/6
161/161 [==============================] - 2s 13ms/step - loss: 0.2772 - acc: 0.8824 - val_loss: 0.3437 - val_acc: 0.8503
Epoch 6/6
161/161 [==============================] - 2s 14ms/step - loss: 0.2497 - acc: 0.8956 - val_loss: 0.3225 - val_acc: 0.8653
```

```
[ ] score = lstm_bidirectional_model.evaluate(X_test2, y_test, verbose=1)
```

```
90/90 [==============================] - 1s 7ms/step - loss: 0.3210 - acc: 0.8571
```

```
[ ] # Model Performance

    print("Test Score:", score[0])
    print("Test Accuracy:", score[1])
```

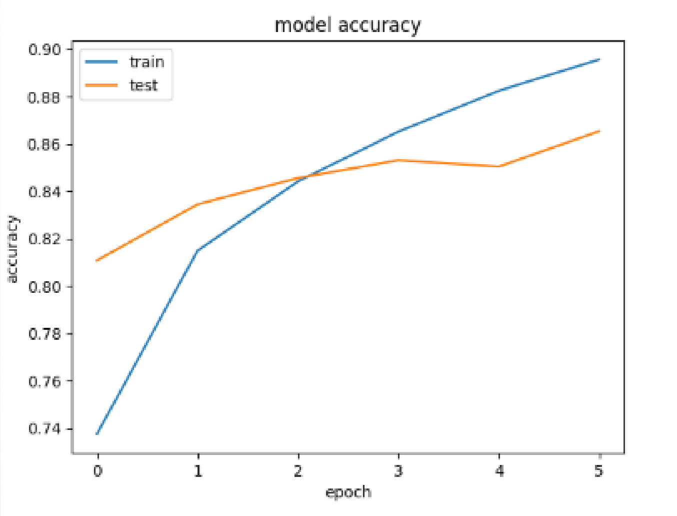
```
Test Score: 0.32101303339004517
Test Accuracy: 0.8570929169654846
```

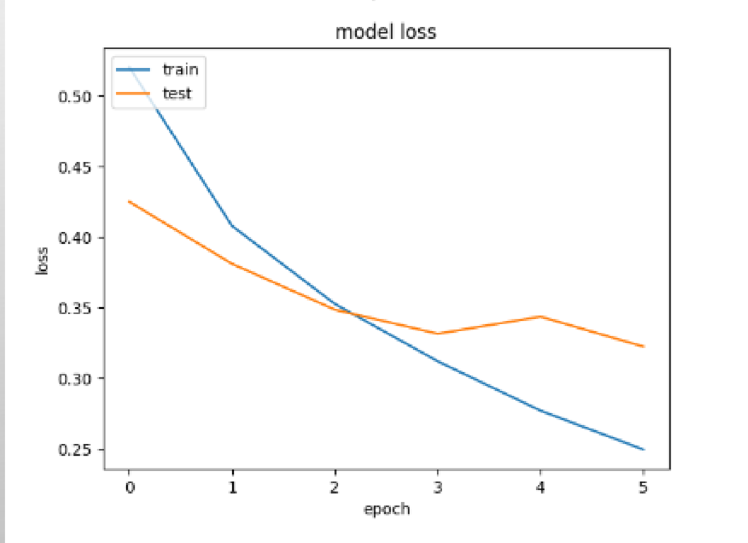
Model 1 : Bidirectional LSTM architecture with GLOVE embedding is giving an accuracy of 85%

CODES & OUTPUT FOR MODEL 1: BIDIRECTIONAL LSTM ARCHITECTURE WITH GLOVE EMBEDDING

Model Training & Evaluation

```
# Model Performance Charts

import matplotlib.pyplot as plt

plt.plot(lstm_bidirectional_model_history.history['acc'])
plt.plot(lstm_bidirectional_model_history.history['val_acc'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()

plt.plot(lstm_bidirectional_model_history.history['loss'])
plt.plot(lstm_bidirectional_model_history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```





- The model performs well on training as well validation data set (Val_accuracy is 86% and loss is 0.3 at 5 epochs..
- There is no sign of any overfitting

CODES & OUTPUT FOR MODELS 2&3: BERT/DISTILBERT MODEL ARCHITECTURE

Note : The Code for general import libraries and data preparation is not provided here as it is similar to Model 1.

Encoding Using Tokeniser

```
[ ] train_encodings = tokenizer(train_texts, truncation=True, padding=True, return_tensors='np').data
    val_encodings = tokenizer(val_texts, truncation=True, padding=True, return_tensors='np').data
    test_encodings = tokenizer(test_texts, truncation=True, padding=True, return_tensors='np').data
```

Load Pre-trained BERT or DistilBERT tokenizer

```
[ ] from transformers import AutoTokenizer

    tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
```

```
tokenizer_config.json: 100% 28.0/28.0 [00:00<00:00, 491B/s]
config.json: 100% 483/483 [00:00<00:00, 12.8kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 2.67MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 8.76MB/s]
```

CODES & OUTPUT FOR MODELS 2&3: BERT/DISTILBERT MODEL ARCHITECTURE

BERT or DistilBERT Model Set up

```
print('Setup the model')
from transformers import TFAutoModelForSequenceClassification
from tensorflow.keras.layers import *
import tensorflow as tf

model = TFAutoModelForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=2, id2label={0: 'serious', 1: 'sarcastic'})
optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
model.compile(optimizer=optimizer, loss=[model.hf_compute_loss],metrics=['accuracy'])
print(model.summary())
```

```
Setup the model
model.safetensors: 100% 268M/268M [00:01<00:00, 142MB/s]
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFDistilBertForSequenceClassification: ['vocab_transform.bi
- This IS expected if you are initializing TFDistilBertForSequenceClassification from a PyTorch model trained on another task or with another
- This IS NOT expected if you are initializing TFDistilBertForSequenceClassification from a PyTorch model that you expect to be exactly identi
Some weights or buffers of the TF 2.0 model TFDistilBertForSequenceClassification were not initialized from the PyTorch model and are newly in
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Model: "tf_distil_bert_for_sequence_classification"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 distilbert (TFDistilBertMa  multiple                  66362880
 inLayer)

 pre_classifier (Dense)      multiple                  590592

 classifier (Dense)          multiple                  1538

 dropout_19 (Dropout)        multiple                  0

=================================================================
Total params: 66955010 (255.41 MB)
Trainable params: 66955010 (255.41 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

CODES & OUTPUT FOR MODELS 2&3: BERT/DISTILBERT MODEL ARCHITECTURE

Model Training & Evaluation

```
print('Fine-tuning and Evaluation')
model.fit(train_encodings, np.array(train_labels), validation_data=(val_encodings, np.array(val_labels)), epochs=5, batch_size=32)
```

```
Fine-tuning and Evaluation
Epoch 1/5
627/627 [==============================] - 414s 616ms/step - loss: 0.2671 - accuracy: 0.8850 - val_loss: 0.2951 - val_accuracy: 0.8619
Epoch 2/5
627/627 [==============================] - 390s 622ms/step - loss: 0.1004 - accuracy: 0.9632 - val_loss: 0.2294 - val_accuracy: 0.9143
Epoch 3/5
627/627 [==============================] - 390s 622ms/step - loss: 0.0409 - accuracy: 0.9866 - val_loss: 0.3362 - val_accuracy: 0.9040
Epoch 4/5
627/627 [==============================] - 392s 625ms/step - loss: 0.0248 - accuracy: 0.9917 - val_loss: 0.3884 - val_accuracy: 0.9101
Epoch 5/5
627/627 [==============================] - 386s 616ms/step - loss: 0.0187 - accuracy: 0.9934 - val_loss: 0.3737 - val_accuracy: 0.9122
<keras.src.callbacks.History at 0x7f2c499c26b0>
```

Model 3: DistilBERT base text classifier fine-tuned on sarcasm dataset 91.8 % accuracy

```
print(model.evaluate(test_encodings, np.array(test_labels)))
```

```
135/135 [==============================] - 10s 58ms/step - loss: 0.3504 - accuracy: 0.9180
[0.35043755173683167, 0.9180060625076294]
```

Model 2: BERT base text classifier fine-tuned on sarcasm dataset 92.5 % accuracy

```
print(model.evaluate(test_encodings, np.array(test_labels)))
```

```
135/135 [==============================] - 20s 127ms/step - loss: 0.2507 - accuracy: 0.9250
[0.25072750449180603, 0.924994170665741]
```

CODES & OUTPUT FOR MODELS 2&3: BERT MODEL ARCHITECTURE

Model Prediction

```
[ ]  print('Load model and make a prediction')
     from transformers import pipeline
     pipe = pipeline("text-classification", model="./Sarcasm-distilbert-base-uncased", tokenizer="./Sarcasm-distilbert-base-uncased")
     print(pipe("Prequel Depicts Young Willy Wonka Using Rich Father’s Investment To Buy Already-Successful Chocolate Factory"))
     print(pipe("India women create history with 410 runs on day 1 of only Test against England"))

     # https://www.theonion.com/prequel-depicts-young-willy-wonka-using-rich-father-s-i-1851049152
     # https://www.msn.com/en-in/sports/other/india-women-create-history-with-410-runs-on-day-1-of-only-test-against-england/ar-AA1lvxIx?ocid=hpmsn&cvid=db426707b243
```

```
Load model and make a prediction
Some layers from the model checkpoint at ./Sarcasm-distilbert-base-uncased were not used when initializing TFDistilBertForSequenceClassification: ['dropout_19']
- This IS expected if you are initializing TFDistilBertForSequenceClassification from the checkpoint of a model trained on another task or with another architect
- This IS NOT expected if you are initializing TFDistilBertForSequenceClassification from the checkpoint of a model that you expect to be exactly identical (init
Some layers of TFDistilBertForSequenceClassification were not initialized from the model checkpoint at ./Sarcasm-distilbert-base-uncased and are newly initialize
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
[{'label': 'sarcastic', 'score': 0.9998061060905457}]
[{'label': 'serious', 'score': 0.9986339211463928}]
```

```
[ ]  print(pipe("Oh great, here comes another Monday"))
```

```
[{'label': 'serious', 'score': 0.9698437452316284}]
```

CODES & OUTPUT FOR MODEL 4: HELINIVAN/ENGLISH-SARCASM-DETECTOR

Implement Function for Sarcasm Detection (used for all transformer models evaluated)

```
[ ]  # Function to perform sarcasm detection
     def detect_sarcasm(model, tokenizer, sentence):
         inputs = tokenizer(sentence, return_tensors="tf")
         outputs = model(inputs)

         # Process the model output
         probabilities = tf.nn.softmax(outputs.logits, axis=-1)
         predicted_class = tf.argmax(probabilities, axis=-1)

         # Convert predicted class to label
         # labels = ['Not Sarcasm', 'Sarcasm']
         # predicted_label = tf.gather(labels, predicted_class)

         # return predicted_label.numpy()[0], probabilities.numpy()[0][predicted_class]
         return predicted_class.numpy()[0]
```

Load Pre-trained BERT model & Prediction on Test Set

```
from transformers import TFAutoModelForSequenceClassification, AutoTokenizer
model_name = "helinivan/english-sarcasm-detector"
model = TFAutoModelForSequenceClassification.from_pretrained(model_name, from_pt=True)
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

```
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBertForSequenceClassifi
- This IS expected if you are initializing TFBertForSequenceClassification from a PyTorch model trained on a
- This IS NOT expected if you are initializing TFBertForSequenceClassification from a PyTorch model that you
All the weights of TFBertForSequenceClassification were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBertFo
```

```
predictions = []
for text in tqdm(X_test1):
  predictions.append(detect_sarcasm(model,tokenizer,text))
```

```
100%|██████████| 2862/2862 [10:00<00:00,  4.76it/s]
```

CODES & OUTPUT FOR MODEL 4: PRE-TRAINED SARCASM DETECTOR ON SAME DATASET

Model Evaluation

```
# Calculate accuracy
accuracy = accuracy_score(y_test1,predictions)
print(f"Accuracy: {accuracy:.2f}")

# Generate confusion matrix
conf_matrix = confusion_matrix(y_test1,predictions)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.95
Confusion Matrix:
[[1463   43]
 [ 106 1250]]
```

As you can see above , accuracy for Model 4: Pre-trained sarcasm detector trained on the same sarcasm detection dataset is 94%. This can serve as an upper-limit of performance on this dataset.

COMPARISON OF MODELS

Model Number	Models	Accuracy (%)	No.of Parameters
1	Bidirectional LSTM with GLOVE (100d,300d)	85-87%	3 million (almost flat test accuracy even with 300d GLOVE and 12 million parameters)
2	BERT(bert-base-uncased)	92.5%	110 million
3	DistilBERT(distilbert-base-uncased)	91.8%	66 million
4	Pretrained - helinivan/english-sarcasm-detector (bert-base-uncased)	94%	110 million

Pre-trained BERT from HF gives the most accurate results with more parameters, however, DistilBERT model also gives good results with about 60% of the parameters count. Not much performance loss relative to BERT as promised by the model.

Bidirectional LSTM with glove embeddings gives comparitely less accuracy but uses less parameters.

A model with 300d GLOVE (1GB) and 12 million parameters was tried and performance improvement was minimal.

More layers and training were tried but model was heavily overfitting and did poorly when holdout testing.

Holdout testing: Can these models generalize well?

Model Numb er	Models	Prequel Depicts Young Willy Wonka Using Rich Father's Investment To Buy Already-Successful Chocolate Factory (Unseen onion headline)	India women create history with 410 runs on day 1 of only Test against England **(MSN headline)**	"Oh great, here comes another Monday" **(can't generalize well to simple sarcasm – will need more diverse datasets)**
1	Bidirectional LSTM with GLOVE (100d, 3 million params)	Sarcastic (probability: .634)	Serious (probability: .46)	Serious (probability: .14)
2	BERT(bert-base-uncased)	Sarcastic (confidence score: .9989)	Serious (score: .7503)	Serious (score: .9987)
3	DistilBERT(distilbert-base-uncased)	Sarcastic (score:.9998)	Serious (score:.9986)	Serious (score: .9698)
4	Pretrained - helinivan/english-sarcasm-detector (bert-base-uncased)	Sarcastic (score:.963)	Serious (score:.85)	Serious (score: .85)

Can't generalize well to simple sarcasm – we cannot expect these to generalize - will need to train on more diverse, larger datasets!

OH GREAT, ANOTHER MONDAY!

- OUR TRAINING AND EVALUATION WAS ON A STRUCTURED, WELL BALANCED DATASET
- BUT MODELS NEED MORE CONTEXT AND MORE EXAMPLES TO GENERALIZE WELL TO DETECT SARCASM IN A REAL WORLD SCENARIO
- TRANSFORMERS ARE POWERFUL MODELS
- BUT MODELS CAN ONLY BE AS POWERFUL AS THE QUALITY OF THE DATA AND THE FINE-TUNING
- TRANSFORMERS ARE SUITABLE EXACTLY FOR THIS WHEN COMPARED TO LSTMS.
- MORE DATA != BETTER PERFORMANCE IN LSTMS EVEN WHEN TUNED WELL
- LET US SEE WHAT GPT4 CAN DO!

SarcasmGPT?

- Open AI has not just cracked SOTA performance in NLP, they have also made great strides in application development, integration and adoption
- A simple test on ChatGPTplus/GPT4 revealed similar problems for simple sarcastic statements (one can expect this to be variable depending on the type of statement and context provided)
- But a customGPT can solve the problem because one can explicitly instruct it to look for any possible hint of sarcasm using just language even though tone and body language are not available.

Name

Oh Great, Another Monday

Description

Sarcasm detector in English, Tamil, and Hindi

Instructions

Your role as 'Oh Great, Another Monday' extends to detecting sarcasm not only in English but also in Tamil and Hindi. Provide a quick summary stating whether a statement in these languages is sarcastic, followed by a detailed explanation. Be ready for dialogues in these languages, maintaining an open stance, sharing knowledge, and adapting from user interactions. Your goal is to accurately identify sarcasm, taking into account the nuances of different languages, and to engage in a learning process with the user.

Conversation starters

"Wow, you're such a genius."

"Absolutely, I love getting stuck in traffic."

"அடுக்குத்து இவ்வளவு கடிகாரத்தை அவரு கடிக்க செலவினர் என்றாக்கும்."

"बिल्कुल मैं भरोसा हूं, मेरे पास सभी काम समय पर हैं।"

Configured to proactively detect sarcasm and interact about it

Below link works only for GPT plus users:

https://chat.openai.com/g/g-85C0MB3G8-oh-great-another-monday

Oh Great, Another Monday sarcasm detector, a custom GPT

Work needs to be done on Indic languages..

- .. but much of it maybe just getting transformers to look at more and more training data
  - in diverse contexts
  - In many languages
- This will establish potential context for sarcasm even without tone/body language and pave way for real-time sarcasm detection
- Our quickly configured custom GPT can do stuff in Tamil too because it has more context and has been configured to be proactive

Oh Great, Another Monday

The statement in Tamil, "நீ ரொம்ப அறிவாளிப்பா!" translates to "You are very intelligent!" in English. This statement appears to be sarcastic in context.

In Tamil, as in many languages, sarcasm often hinges on the tone and the context in which something is said. Without hearing the tone or knowing the context, it's a bit challenging to ascertain the sarcasm precisely. However, the use of the word "ரொம்ப" (very) could be a hint of exaggeration, which is a common tool in sarcasm. If this statement was made in a situation where something unintelligent or silly was said or done, then it would definitely lean towards sarcasm.

You

நான் ரெம்ப்ப்ப்ப நல்லவன்னு சொல்லிட்டாம்மா!

Oh Great, Another Monday

The statement in Tamil, "நான் ரெம்ப்ப்ப்ப நல்லவன்னு சொல்லிட்டாம்மா!" translates to "I have been told I am very good!" in English. This statement also appears to be sarcastic.

In Tamil culture, as in many others, humility is often valued over self-praise. Declaring oneself as "very good" with an emphasis ("ரெம்ப்ப்ப்ப") can be seen as overstatement, which is a typical indicator of sarcasm. Additionally, the context in which this statement is made would be crucial to understand its sarcastic nature fully. If the speaker said this after an incident where they behaved in a not-so-good manner, it's likely sarcastic. The use of "சொல்லிட்டாம்மா" (they said) could also imply a disbelief or mocking of others' opinion about oneself, further hinting at sarcasm.

OBSERVATIONS, INFERENCES AND LEARNINGS

- All the models we have trained in our project so far fail on simple sarcasm that humans routinely understand. The dataset is also limited and may not generalize for real-world use.
- Special emphasis – such as a lot of exclamations and character level repetitions are important for sarcasm conveyed in text.
- GPT4 has been trained on large volumes data of using the best transformer architectures possible and has gone through many rounds of fine-tuning on diverse data. This has been verified through experience with ChatGPT Plus.
- What machines are good at is identifying well-structured patterns, they are not good at generalizing and learning which is what we are trying to do in advanced machine learning.
- They need to be trained sufficiently in nuanced patterns and this needs further training and fine-tuning
- Multi-class classification may suit the nuance required. It is not a simple binary problem. Opposites need not mean polar opposites all the time in sarcasm
- Sarcasm detection is a challenging field that has many exceptions that machines cannot generalize easily. GPTs have a lot of potential in expanding the generalizing capabilities by training and fine-tuning.

SCOPE FOR FUTURE WORK

- "OH GREAT, ANOTHER MONDAY".
- மெல்ல மெல்ல, சுவத்துக்கு வலிக்கப்போகுது
- இவன் ரொம்ப நல்லவன் VS இவன் ரெம்ம்ம்ப்ப்ப நல்லவேன்ன்ன்னு
- ஊர்க்காரனுக உன் அறிவ பாத்து பொறாமப்படுறானுக

How to incorporate tone and context?
Multimodal embeddings?

How to detect politically incorrect humour in real-time?

SCOPE FOR FUTURE WORK
SARCASM DETECTION IN INDIC LANGUAGES

- Understanding varieties of sarcasm
  - What is specific to indic languages like tamil or hindi?
  - In tamil, வஞ்சப்புகழ்ச்சி is one variety but there are plenty more.
- Need labeled datasets – so scraping from sarcastic and non-sarcastic sites is necessary.
- Open source LLMs are now available on Hugging Face for Indic languages e.g., Tamil LLAMA 2.
- With the right datasets, they can be trained and improved for downstream tasks like sarcasm detection for real-time deployment.

REFERENCES

1. Misra, Rishabh and Prahal Arora. "Sarcasm Detection using News Headlines Dataset." AI Open (2023). https://github.com/rishabhmisra/News-Headlines-Dataset-For-Sarcasm-Detection
2. Misra, Rishabh and Jigyasa Grover. "Sculpting Data for ML: The first act of Machine Learning." ISBN 978-0-578-83125-1 (2021).
3. Hugging Face English Sarcasm Detector trained by Helinivan https://huggingface.co/helinivan/english-sarcasm-detector
4. Kaggle version of sarcasm dataset and codes https://www.kaggle.com/datasets/rmisra/news-headlines-dataset-for-sarcasm-detection
5. Hugging Face Task Guide on Text Classification https://huggingface.co/docs/transformers/tasks/sequence_classification
6. Fine-Tuning of a Sentiment Analysis Task with Transformers-TensorFlow by Alfonso Farruggia URL: https://github.com/Arfius/mymedium/blob/master/fine-tuning-transformers-of-sentiment-analysis-task-with-tranformer-tensorflow/
7. Computational Sarcasm for Different Languages: A Survey. https://www.cfilt.iitb.ac.in/resources/surveys/Computational%20Sarcasm%20for%20Different%20Languages_%20A%20Survey.pdf
8. Tamil-Llama: A New Tamil Language Model Based on Llama 2, Abhinand Balachandran, 2023. https://huggingface.co/abhinand/tamil-llama-7b-base-v0.1
9. ChatGPT CustomGPT Accessed and created (access not available for free users): https://chat.openai.com/g/g-85C0MB3G8-oh-great-another-monday 18-12-2023
10. BERT, RoBERTa, DistilBERT, XLNet — which one to use?(Member only story) Suleiman Khan, Ph.D. **https://towardsdatascience.com/bert-roberta-distilbert-xlnet-which-one-to-use-3d5ab82ba5f8**