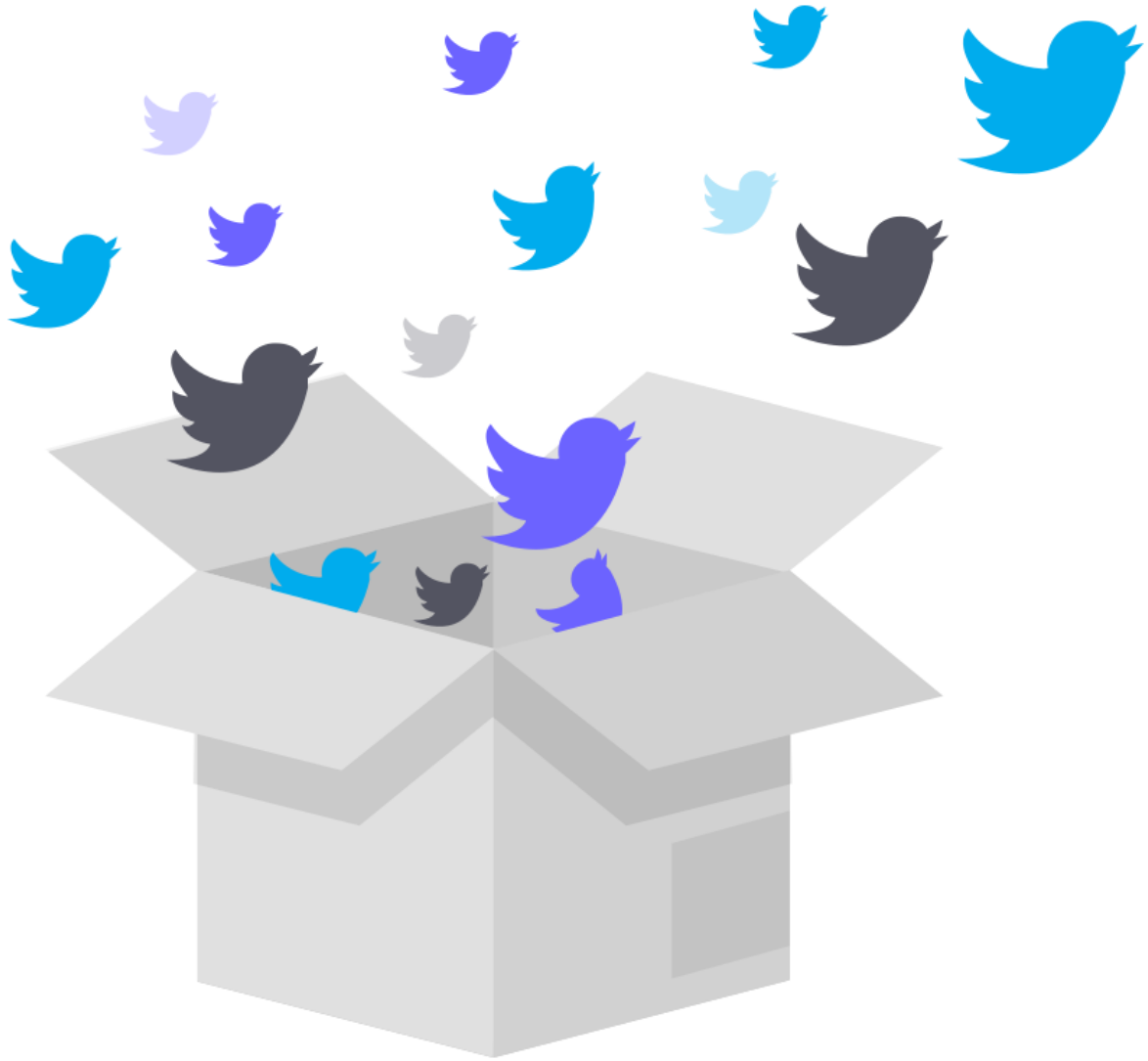


Text Classification
Twitter Sarcasm Detection

Group Name – Sarcasm Experts

Artsiom Strok
astrok2@illinois.edu

Peter Zukerman
peterz2@illinois.edu

Dheeraj Patta (Captain)
npatta2@Illinois.edu

The Problem Statement

Classify a given sequence of tweets (responses) as sarcastic or not sarcastic.

sar·casm
/ˈsärˌkazəm/

noun

the use of irony to mock or convey contempt.
"his voice, hardened by sarcasm, could not hide his resentment"

Similar: derision | mockery | ridicule | satire | irony | scorn | sneering

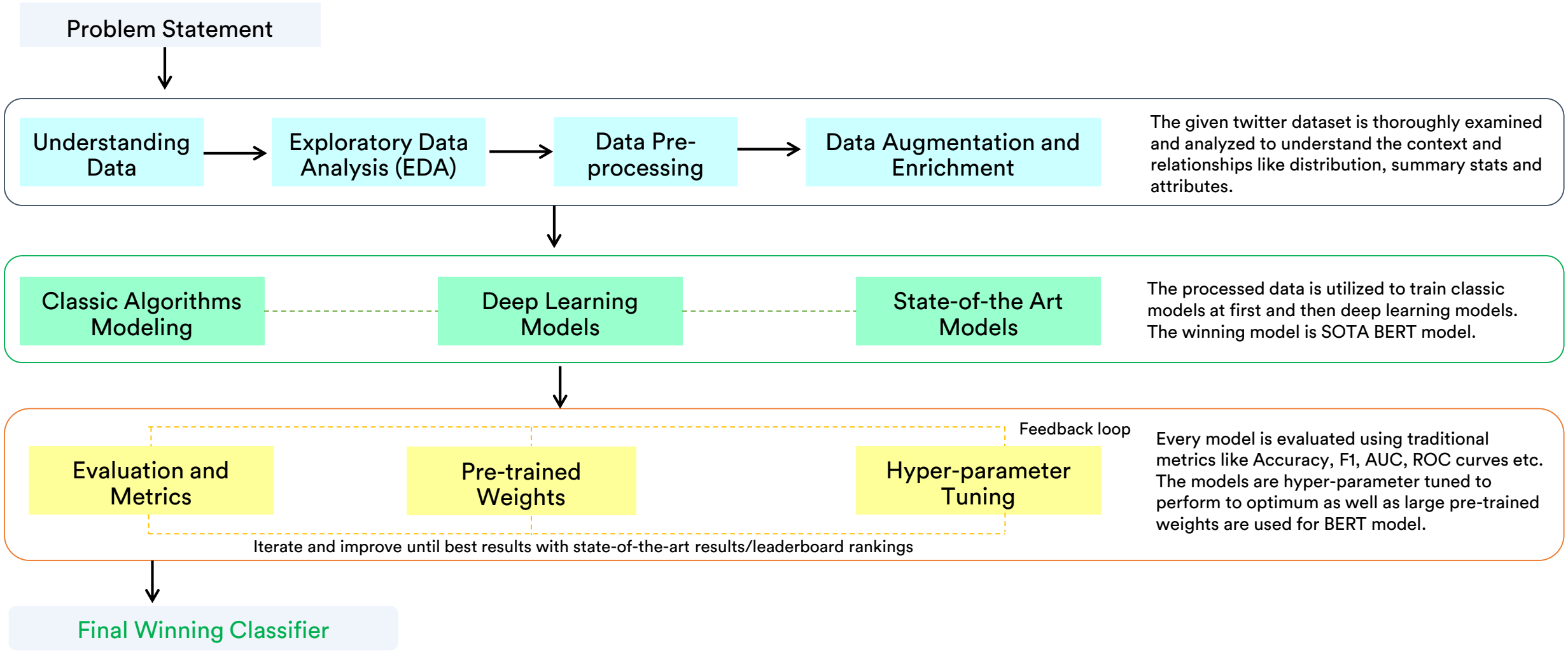
The objective of this competition is to predict the “label” of the response (tweets) using the given context (either immediate or full context)

The tweets are provided with conversation context which is an ordered list of dialogue. The responses are the tweets to be classified.

Why Sarcasm is a hard NLP problem

- Sarcasm is one of the most **complex** forms of figurative human expression—and therefore it's one of the hardest to teach AI systems.
- **Narrow** field of research in Natural Language Processing (NLP).
- Specific case of sentiment analysis where focus is on **sarcasm/mockery** instead of polarity or sentiment in the whole spectrum.
- Common challenges of Sentiment Analysis include Negations, Ambiguity, Multi-polarity and Sarcasm.
- Beyond Ambiguity, to be able to interpret **tone and meaning** of a statement in the context of other statements is hard.
- **Hard** for machines to understand the **contradiction** between literal and intended meaning.
- **Knowledge** about the external world, environment and **context** is often needed to understand sarcastic comments.

Our Approach



Understanding Data and Pre-processing

Basic Attributes

A. Dataset format

Each line contains a JSON object with the following fields:

- ***response*** : the Tweet to be classified
- ***context*** : the conversation context of the
- ***response*** - Note, the context is an ordered list of dialogue, i.e., if the context contains three elements, `c1`, `c2`, `c3`, in that order, then `c2` is a reply to `c1` and `c3` is a reply to `c2`. Further, the Tweet to be classified is a reply to `c3`.
- ***label*** : `SARCASM` or `NOT_SARCASM`

TABLE I
DATASET SIZE STATISTICS

Train	Test
5000	1800

Pre-processing

We've performed the following data pre-processing steps using *genism*,

- POS tagging,
- Tokenization
- Lemmatization
- Label Encodings
- Word Embeddings etc.

and utilized *ekphrasis* package to perform specialized tasks like

- Social Tokenization
- Text pre-processor
- Word Segmentation on Hashtags
- Unpack Contractions (can't to can not),
- Spell Correction and
- Emoticon Analysis

```
from ekphrasis.classes.preprocessor import TextPreProcessor
from ekphrasis.classes.tokenizer import SocialTokenizer
from ekphrasis.dicts.emoticons import emoticons

text_processor = TextPreProcessor(
    # terms that will be normalized
    normalize=['url', 'email', 'percent', 'money', 'phone', 'user',
        'time', 'date', 'number'],
    # terms that will be annotated
    annotate={"hashtag", "allcaps", "elongated", "repeated",
        'emphasis', 'censored'},
    fix_html=False,  # fix HTML tokens

    # corpus from which the word statistics are going to be used
    # for word segmentation
    segmenter="twitter",

    # corpus from which the word statistics are going to be used
    # for spell correction
    corrector="twitter",

    unpack_hashtags=True,  # perform word segmentation on hashtags
    unpack_contractions=True,  # Unpack contractions (can't -> can not)
    spell_correct_elong=False,  # spell correction for elongated words

    # select a tokenizer. You can use SocialTokenizer, or pass your own
    # the tokenizer, should take as input a string and return a list of tokens
    tokenizer=SocialTokenizer(lowercase=False).tokenize,

    # list of dictionaries, for replacing tokens extracted from the text,
    # with other expressions. You can pass more than one dictionaries.
    dicts=[emoticons]
)
```

```
Word statistics files not found!
Downloading... done!
Unpacking... done!
Reading twitter - 1grams ...
generating cache file for faster loading...
reading ngrams /root/.ekphrasis/stats/twitter/counts_1grams.txt
Reading twitter - 2grams ...
generating cache file for faster loading...
reading ngrams /root/.ekphrasis/stats/twitter/counts_2grams.txt
Reading twitter - 1grams ...
```

** Code snippet from final BERT model*

Initial Modeling (Classic)

Below is the list of classic models developed using Python's scikit-learn and their performance metrics.

Classic Algorithms developed –

- Count Vectorizer
- TD-IDF Transformer
- Multinomial Naïve Bayes
- Stochastic Gradient Descent (SGD)

Boosting Algorithms developed –

- Gradient Boosting (GB)
- XG Boosting (XGB)

Validation Mechanism -

- K-fold stratified cross validation

TABLE II
CLASSICAL ML ALGORITHMS PERFORMANCE

Model	Precision	Recall	F1
Multinomial Naïve Bayes	0.7349	0.7808	0.7570
XGBoost	0.6548	0.7744	0.7095

TABLE III
AUTO SCIKIT LEARN RESULTS

Model	Accuracy
LibSVM (SVC)	0.7770
Random Forest	0.7624
Bernoulli NB	0.7545
SGD	0.7509
Extra Trees	0.7491
Passive Aggressive	0.7412
LibLinear (SVC)	0.7388
AdaBoost	0.7176
KNN	0.6921
DT	0.6321

Source Code
https://github.com/dheerajpatta/CourseProject/blob/main/models/sarcasm_classification.ipynb

Modeling (Deep Learning Models)

We've developed deep learning model developed using TensorFlow and high-level abstraction framework **Keras.**

NLP steps -

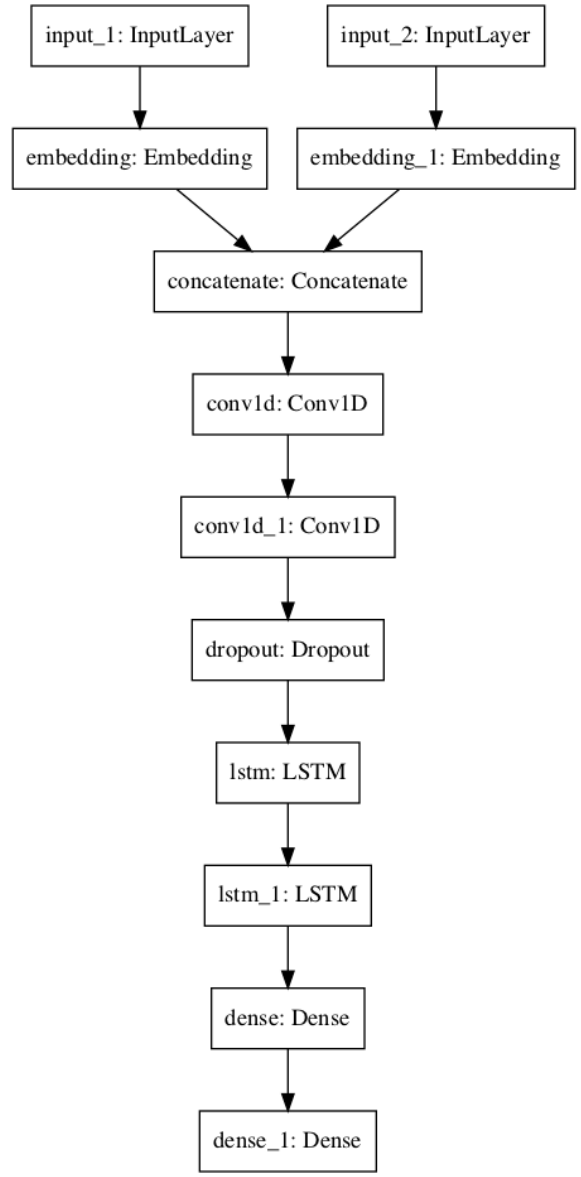
- Word Embeddings - Word2Vec, Glove (Genism)
- Keyed Vectors (Genism)
- Label Encodings

Deep Learning Models -

- Conv 1D (CNN)
- Long Short Term Memory (LSTM)
- Dropouts
- Dense/Hidden Layers

Evaluation Metric -

- Categorical Cross-entropy



Source Code
https://github.com/dheerajpatta/CourseProject/blob/main/models/sarcasm_classification_deep_learning.ipynb

State-of-the-art (SOTA) Modeling - BERT

What is BERT?

BERT stands for Bi-directional Encoder Representations from Transformers. BERT is a general purpose pre-trained model that can be finetuned on smaller task-specific datasets like sentiment analysis, question & answer system etc.

Why BERT?

- **BERT** was revolutionary when introduced by Google. It is bidirectionally trained which means it has deeper sense of language context and flow compared to Uni-directional models.
- Instead of predicting the next word in a sequence, BERT makes use of a novel technique called **Masked LM** (MLM): it randomly masks words in the sentence and then it tries to predict them and uses the full context of the sentence, both left and right surroundings, in order to predict the masked word. Unlike the previous language models, it takes both the previous and next tokens into account at the **same time.**

How BERT works?

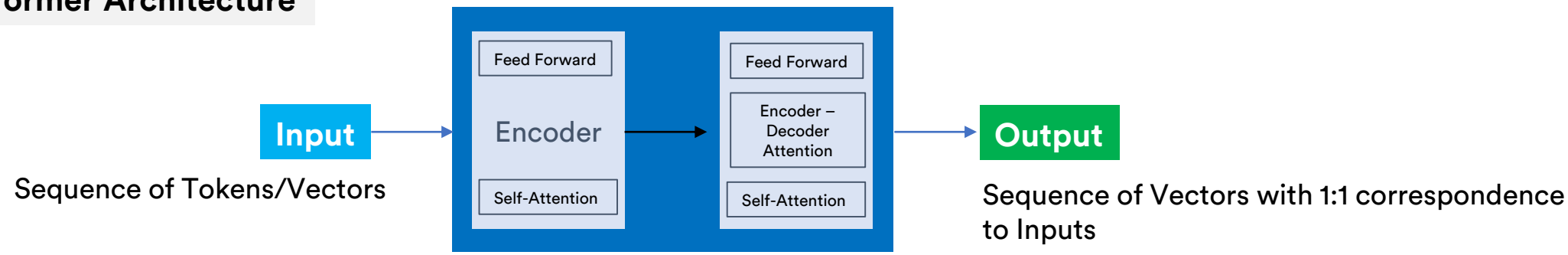
BERT relies on Transformers (the attention mechanism that learns contextual relationships between words in a given text)

A basic Transformer consists of an **encoder** to read the text input and a **decoder** to produce a prediction for the task. The input to the encoder for BERT is a sequence of tokens, which are first converted into vectors and then processed in the neural network.

https://github.com/google-research/bert

BERT (continued)

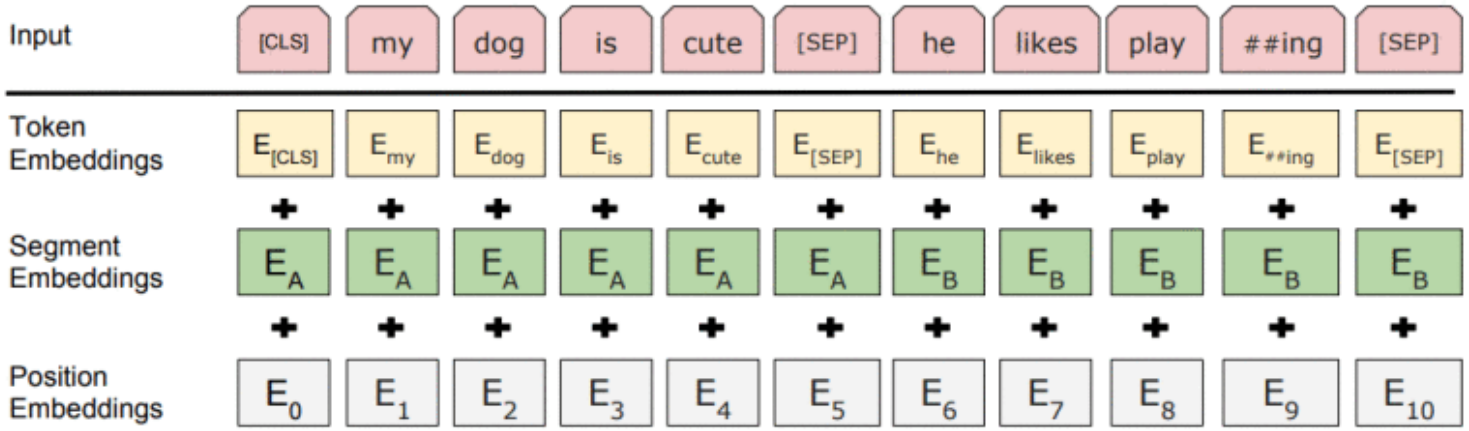
Transformer Architecture



BERT needs the input to be massaged and decorated with some extra metadata:

- **Token embeddings**: A [CLS] token is added to the input word tokens at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
- **Segment embeddings**: A marker indicating Sentence A or Sentence B is added to each token. This allows the encoder to distinguish between sentences.
- **Positional embeddings**: A positional embedding is added to each token to indicate its position in the sentence.

Input Embeddings = Token + Segment + Position Embeddings



Our State-of-the-art (SOTA) Modeling (BERT Base)

We've developed our BERT model using TensorFlow and high-level abstraction framework **Keras.**
Google Colab is used to develop, train, deploy and evaluate BERT model.

BERT Model Architecture

- Google's Pre-trained BERT-Base Model
- BERT-Base, Uncased
- 12-layers, 768-hidden-nodes, 12-attention-heads
- Trained Parameters - 110M parameters
- Stock Weights / Pretrained Weights.
- Trained using Google Cloud NVDIA GPUs (Google Colab)

BERT Model Evaluation Metrics

- Confusion Matrix
- Classification Report
- Accuracy, Precision, Recall, F1 score

Our BERT Model Implementation Explained

Mounting drive for saving all activity

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

```
Mounted at /content/drive
```

Pandas for reading and writing to files. Read the given JSON train file

```
import pandas as pd

with open('/content/drive/My Drive/data/train.jsonl') as f:
    train_data = pd.read_json(f, lines=True)
with open('/content/drive/My Drive/data/validation.jsonl') as f:
    test_data = pd.read_json(f, lines=True)
```

Data prep/cleanup – Remove non-contextual/meaningless redundant words like @USER, <URL>

```
train_data.response = train_data.response.str.replace('@USER', '', regex=False)
train_data.response = train_data.response.str.replace('<URL>', '', regex=False)

test_data.response = test_data.response.str.replace('@USER', '', regex=False)
test_data.response = test_data.response.str.replace('<URL>', '', regex=False)
```

Data pre-processing for specialized NLP tasks like Social Tokenizer, Emotion Understanding etc.

```
!pip install ekphrasis
```

Our BERT Model Implementation Explained

Using ekphrasis to pre-process and enrich data

```
from ekphrasis.classes.preprocessor import TextPreProcessor
from ekphrasis.classes.tokenizer import SocialTokenizer
from ekphrasis.dicts.emoticons import emoticons

text_processor = TextPreProcessor(
    # terms that will be normalized
    normalize=['url', 'email', 'percent', 'money', 'phone', 'user',
        'time', 'date', 'number'],
    # terms that will be annotated
    annotate={"hashtag", "allcaps", "elongated", "repeated",
        'emphasis', 'censored'},
    fix_html=False,  # fix HTML tokens

    # corpus from which the word statistics are going to be used
    # for word segmentation
    segmenter="twitter",

    # corpus from which the word statistics are going to be used
    # for spell correction
    corrector="twitter",

    unpack_hashtags=True,  # perform word segmentation on hashtags
    unpack_contractions=True,  # Unpack contractions (can't -> can not)
    spell_correct_elong=False,  # spell correction for elongated words

    # select a tokenizer. You can use SocialTokenizer, or pass your own
    # the tokenizer, should take as input a string and return a list of tokens
    tokenizer=SocialTokenizer(lowercase=False).tokenize,

    # list of dictionaries, for replacing tokens extracted from the text,
    # with other expressions. You can pass more than one dictionaries.
    dicts=[emoticons]
)
```

Rich data ready for Training

```
train_data.context
```

```
0       [If, your, child, is, not, named, Barron, ., <...
1       [having, to, make, up, excuses, of, why, your,...
2       [I, ’, ll, remember, to, not, support, you, at...
3       [But, not, half, as, stupid, as, Schiff, looks...
4       [They, already, did, ., Obama, said, many, tim...
                              ...
4995    [We, will, work, on, this, and, reach, out, at...
4996    [Yup, she, went, there, ., Nah, not, at, all, ...
4997    [", some, look, at, silence, as, boredom, or, ...
4998    [Lots, of, people, believed, –, and, still, fe...
4999    [Atiku, ', s, campaign, is, coming, wit, Arabi...
Name: context, Length: 5000, dtype: object
```

Using ekphrasis text-processor methods for response and context

```
train_data.response = train_data.response.apply(text_processor.pre_process_doc)
train_data.context = train_data.context.apply(lambda x: [text_processor.pre_process_doc(s) for s in x])

test_data.response = test_data.response.apply(text_processor.pre_process_doc)
test_data.context = test_data.context.apply(lambda x: [text_processor.pre_process_doc(s) for s in x])
```

Our BERT Model Implementation Explained

Utilizing GPUs for training

```
!nvidia-smi
```

```
Tue Nov 17 18:45:54 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 455.38       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   53C    P8    10W /  70W |      0MiB / 15079MiB |      0%      Default |
|                               |                      |                 ERR! |
+-------------------------------+----------------------+----------------------+
```

Install all necessary libraries and frameworks

```
!pip install tensorflow-gpu >> /dev/null
!pip install --upgrade grpcio >> /dev/null
!pip install tqdm  >> /dev/null
!pip install bert-for-tf2 >> /dev/null
```

```
import os
import math
import datetime

from tqdm import tqdm

import pandas as pd
import numpy as np

import tensorflow as tf
from tensorflow import keras

import bert
from bert import BertModelLayer
from bert.loader import StockBertConfig, map_stock_config_to_params, load_stock_weights
from bert.tokenization.bert_tokenization import FullTokenizer

from sklearn.metrics import confusion_matrix, classification_report
from sklearn import preprocessing
```

Our BERT Model Implementation Explained

Download pre-trained BERT-Base model weights

```
!wget https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-12_H-768_A-12.zip
!unzip uncased_L-12_H-768_A-12.zip
```

```
--2020-11-17 18:46:43--  https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-12_H-768_A-12.zip
Resolving storage.googleapis.com (storage.googleapis.com)... 172.217.164.176, 172.217.2.112, 172.217.164.144, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|172.217.164.176|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 407727028 (389M) [application/zip]
Saving to: ‘uncased_L-12_H-768_A-12.zip’

uncased_L-12_H-768_ 100%[===================>] 388.84M   190MB/s    in 2.0s

2020-11-17 18:46:45 (190 MB/s) - ‘uncased_L-12_H-768_A-12.zip’ saved [407727028/407727028]

Archive:  uncased_L-12_H-768_A-12.zip
   creating: uncased_L-12_H-768_A-12/
  inflating: uncased_L-12_H-768_A-12/bert_model.ckpt.meta
  inflating: uncased_L-12_H-768_A-12/bert_model.ckpt.data-00000-of-00001
  inflating: uncased_L-12_H-768_A-12/vocab.txt
  inflating: uncased_L-12_H-768_A-12/bert_model.ckpt.index
  inflating: uncased_L-12_H-768_A-12/bert_config.json
```

```
bert_model_name="uncased_L-12_H-768_A-12"

bert_ckpt_dir = os.path.join("model/", bert_model_name)
bert_ckpt_file = os.path.join(bert_ckpt_dir, "bert_model.ckpt")
bert_config_file = os.path.join(bert_ckpt_dir, "bert_config.json")
```

Our BERT Model Implementation Explained

Preparing the data as required by BERT (CLS, PAD, SEP)

```
class SarcasmDetectionData:

  def __init__(self, train, test, tokenizer: FullTokenizer, classes, max_seq_len=72):
    self.tokenizer = tokenizer
    self.max_seq_len = 0
    self.classes = classes

    ((self.train_x, self.train_y), (self.test_x, self.test_y)) = map(self._prepare, [train, test])

    print("max seq_len", self.max_seq_len)
    self.max_seq_len = min(self.max_seq_len, max_seq_len)
    self.train_x, self.test_x = map(self._pad, [self.train_x, self.test_x])

  def _prepare(self, df):
    x, y = [], []

    for _, row in tqdm(df.iterrows()):
      text, context, label = ' '.join(row['response']),' '.join(row['context']), row['label']
      tokens = ["[CLS]"]
      tokens.extend(self.tokenizer.tokenize(text))
      tokens.append("[PAD]")
      tokens.extend(self.tokenizer.tokenize(context))
      tokens.append("[SEP]")
      token_ids = self.tokenizer.convert_tokens_to_ids(tokens)
      self.max_seq_len = max(self.max_seq_len, len(token_ids))
      x.append(token_ids)
      y.append(self.classes.index(label))

    return np.array(x), np.array(y)

  def _pad(self, ids):
    x = []
    for input_ids in ids:
      input_ids = input_ids[:min(len(input_ids), self.max_seq_len - 2)]
      input_ids = input_ids + [0] * (self.max_seq_len - len(input_ids))
      x.append(np.array(input_ids))
    return np.array(x)
```

Initiating the Tokenizer as well as sequence length and classes.

The prepare() is used to create the input sequence of tokens/vectors in the way required by BERT.

Input Embeddings = Token + Segment + Position Embeddings (CLS, PAD, SEP)

Utilize the Full Tokenizer provided by BERT

```
tokenizer = FullTokenizer(vocab_file=os.path.join(bert_ckpt_dir, "vocab.txt"))
```

Our BERT Model Implementation Explained

Create BERT Model using input sequences and stock BERT config files. Create Deep NN with dropouts, hidden/dense layers.
We make use of activation functions of tanh and softmax for dense layers.

```
def create_model(max_seq_len, bert_ckpt_file):

  with tf.io.gfile.GFile(bert_config_file, "r") as reader:
      bc = StockBertConfig.from_json_string(reader.read())
      bert_params = map_stock_config_to_params(bc)
      bert_params.adapter_size = None
      bert = BertModelLayer.from_params(bert_params, name="bert")

  input_ids = keras.layers.Input(shape=(max_seq_len, ), dtype='int32', name="input_ids")
  bert_output = bert(input_ids)
  cls_out = keras.layers.Lambda(lambda seq: seq[:, 0, :])(bert_output)
  cls_out = keras.layers.Dropout(0.3)(cls_out)
  logits = keras.layers.Dense(units=512, activation="tanh")(cls_out)
  logits = keras.layers.Dropout(0.3)(logits)
  logits = keras.layers.Dense(units=len(classes), activation="softmax")(logits)

  model = keras.Model(inputs=input_ids, outputs=logits)
  model.build(input_shape=(None, max_seq_len))

  load_stock_weights(bert, bert_ckpt_file)

  return model
```

```
classes = train_data.label.unique().tolist()

data = SarcasmDetectionData(train_data, test_data, tokenizer, classes, max_seq_len=256)
```

Our BERT Model Implementation Explained

Create Model using BERT Weights from downloaded pre-trained model

```
model = create_model(data.max_seq_len, bert_ckpt_file)
```

```
Done loading 196 BERT weights from: model/uncased_L-12_H-768_A-12/bert_model.ckpt into <bert.model.BertModelLayer object at 0x7f7bee1ab0b
8> (prefix:bert). Count of weights not found in the checkpoint was: [0]. Count of weights with mismatched shape: [0]
Unused weights from checkpoint:
        bert/embeddings/token_type_embeddings
        bert/pooler/dense/bias
        bert/pooler/dense/kernel
        cls/predictions/output_bias
        cls/predictions/transform/LayerNorm/beta
        cls/predictions/transform/LayerNorm/gamma
        cls/predictions/transform/dense/bias
        cls/predictions/transform/dense/kernel
        cls/seq_relationship/output_bias
        cls/seq_relationship/output_weights
```

```
model.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_ids (InputLayer)       [(None, 256)]             0
_________________________________________________________________
bert (BertModelLayer)        (None, 256, 768)          108890112
_________________________________________________________________
lambda (Lambda)              (None, 768)               0
_________________________________________________________________
dropout (Dropout)            (None, 768)               0
_________________________________________________________________
dense (Dense)                (None, 512)               393728
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 1026
=================================================================
Total params: 109,284,866
Trainable params: 109,284,866
Non-trainable params: 0
```

This shows the summary of BERT Model with all the layers. As shown here, the Model contains Input layer, BERT layer, lambda function, dropouts and dense/hidden layers.

Total parameters and trainable parameters are also shown here.

Our BERT Model Implementation Explained

Compiling the Model with “Accuracy” as Evaluation Metrics.

```
model.compile(
  optimizer=keras.optimizers.Adam(1e-5),
  loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=[keras.metrics.SparseCategoricalAccuracy(name="acc")]
)
```

Saving the checkpoints to TensorFlow recommended hdf5 format for later use (we can load these checkpoints instead of re-training)

```
modelCheckpoint = keras.callbacks.ModelCheckpoint('/content/drive/My Drive/best_sarcasm_model14.hdf5', save_best_only = True)
```

Fitting/Training the BERT Model with a batch size of 24 and epochs of 15 and callbacks/checkpoints to save progress.

```
history = model.fit(
  x=data.train_x,
  y=data.train_y,
  validation_data=(data.test_x, data.test_y),
  batch_size=24,
  shuffle=True,
  epochs=15,
  callbacks=[modelCheckpoint]
)
```

```
Epoch 1/15
209/209 [==============================] - ETA: 0s - loss: 0.5966 - acc: 0.6930WARNING:tensorflow:Callbacks method `on_test_batch_end` is
slow compared to the batch time (batch time: 0.0096s vs `on_test_batch_end` time: 0.4280s). Check your callbacks.
209/209 [==============================] - 364s 2s/step - loss: 0.5966 - acc: 0.6930 - val_loss: 0.5996 - val_acc: 0.7000
Epoch 2/15
209/209 [==============================] - 364s 2s/step - loss: 0.5303 - acc: 0.7722 - val_loss: 0.5821 - val_acc: 0.7122
...
Epoch 14/15
209/209 [==============================] - 363s 2s/step - loss: 0.3729 - acc: 0.9404 - val_loss: 0.5758 - val_acc: 0.7350
Epoch 15/15
209/209 [==============================] - 357s 2s/step - loss: 0.3717 - acc: 0.9404 - val_loss: 0.5760 - val_acc: 0.7278
```

Our BERT Model Implementation Explained

Load our Training Weights specific to task of classifying Tweets whether Sarcastic or not.

```
model.load_weights('/content/drive/My Drive/best_sarcasm_model8.hdf5')
```

All that is left is to now “predict” the labels of test/validation dataset using the above loaded weights

```
y_pred = model.predict(data.test_x).argmax(axis=-1)
```

```
print(classification_report(data.test_y, y_pred, target_names=classes))
```

Create a data frame of ID, Result from the Prediction

```
rows = [['twitter_' + str(i+1), 'SARCASM' if r == 0 else 'NOT_SARCASM'] for i, r in enumerate(y_pred)]
results = pd.DataFrame(rows, columns=["ID", "RESULT"])
```

```
results.head()
```

	ID	RESULT
0	twitter_1	NOT_SARCASM
1	twitter_2	SARCASM
2	twitter_3	SARCASM
3	twitter_4	NOT_SARCASM
4	twitter_5	SARCASM

Write the results to answer.txt

```
results.to_csv('/content/drive/My Drive/answer.txt', header = False, index = False)
```

System Requirements (Installation)

1. All our models are in .ipynb format and can be easily replicated by using Jupyter / Google Colab or any other notebook environment.

2. We recommend **anaconda** distribution to create virtual environments for Python and recommend Google Colab for TensorFlow (TF)/Keras Implementations.

3. If you want to use BERT with Colab, you can get started with the notebook "BERT FineTuning with Cloud TPUs".

4. To replicate our BERT model, we highly recommend downloading and using pre-trained stock weights.
https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-12_H-768_A-12.zip

5. We've limited our training to 15 epochs only and the model can be trained for more epochs or perform hyper-parameter tuning to achieve even better results.

6. All about setting up Google Colab https://medium.com/@robertbracco1/configuring-google-colab-like-a-pro-d61c253f7573

7. You need to install the following to execute our BERT code -

- Python 3 or above
- Google Colab / Google Drive (for Mounting)
- Ekphrasis (for pre-processing of data)
- tensorflow-gpu
- tqdm
- bert-for-tf2

Project Team Contributions

As a team, we had very clear goals from the initiation of this project/competition. Below are key highlights and contributions of team members -

Task	Team Member	Highlights
Understanding and Documenting the Problem statement	Dheeraj Patta	Clear and concise documentation with clear goal setting.
Research Data Pre-processing mechanisms and linguistics	Peter Zukerman	New package ekphrasis was introduced and utilized in our models.
Research current SOTA Models and GLUE Metrics + Papers	Dheeraj Patta + Artsiom Strok	We've researched several cutting edge papers related to sentiment and sarcasm analysis.
Implementation of Classic Models and Evaluation Metrics	Artsiom Strok	Quick implementation of classic models led us to understand the depth of the problem which required sophisticated language models than simpler models.
Exploratory Data Analysis (EDA) and Context Understanding	Dheeraj Patta	Visual and Exploration of given twitter dataset.
Implementation of Boosting Algorithms and Evaluation	Dheeraj Patta	XGBoost, CATBoost and Gradient Boosting on top of Classic Algorithms.
Data Augmentation and Enrichment of Data	Peter Zukerman	Social Tokenizers, Hashtag Word Embeddings, Emoticon understanding, unpacking contractions, spell corrections etc.
Implementing a Deep Learning CNN+LSTM Model	Artsiom Strok	DL Model to include a CNN+LSTM with Dropouts and Dense/hidden layers. Failed to perform due to lack of data.
Project Proposal and Progress Report	Dheeraj Patta + Artsiom Strok + Peter Zukerman	Full scores from Reviewers.
Implementation of BERT Model	Artsiom Strok + Peter Zukerman	Our Winning Model – Using pretrained BERT Model. This is our implementation of classic BERT model.
Evaluation, Training and Metrics	Artsiom Strok + Dheeraj Patta	All evaluation tests were performed, documented and recorded for finetuning of all our models.
Implementing Hugging Face Transformers*	Dheeraj Patta	In Progress. Would like to explore the Transformers and see if there is any improvement in our winning model.
Presentation and Final Submission	Dheeraj Patta + Peter Zukerman	Final Submission of documentation with comprehensive walk-through of Project.

Leaderboard Rankings

Model	Precision	Recall	F1	Beat the Baseline	Rank
Classic	0.681	0.684	0.683	No	34
BERT	0.691	0.845	0.761	Yes	2
BERT Sentiment	0.692	0.845	0.761	Yes	5
BERT	0.692	0.845	0.761	Yes	10

References

REFERENCES

[1] Weitzel, Leila & Prati, Ronaldo & Aguiar, Raul. (2016). "The Comprehension of Figurative Language: What Is the Influence of Irony and Sarcasm on NLP Techniques¿' 10.1007/978-3-319-30319-2_3.

[2] Lee, H., Yu, Y., and Kim, G., "Augmenting Data for Sarcasm Detection with Unlabeled Conversation Context", *arXiv e-prints*, 2020.

[3] Dong, X., Li, C., and Choi, J. D., "Transformer-based Context-aware Sarcasm Detection in Conversation Threads from Social Media", *arXiv e-prints*, 2020.

[4] Pant, K. and Dadu, T., "Sarcasm Detection using Context Separators in Online Discourse", *arXiv e-prints*, 2020.

[5] González-Ibáñez, R.I., Muresan, S., & Wacholder, N. (2011). Identifying Sarcasm in Twitter: A Closer Look. ACL.

[6] SoTA Sarcasm Detection Benchmarks https://paperswithcode.com/task/sarcasm-detection/codeless

https://super.gluebenchmark.com/leaderboard/

https://arxiv.org/pdf/2006.06259v1.pdf

https://arxiv.org/pdf/2005.11424.pdf

https://arxiv.org/pdf/2006.00850.pdf

https://www.aclweb.org/anthology/P11-2102.pdf

https://paperswithcode.com/task/sarcasm-detection/codeless