Detecting Sarcasm and Irony in News Headlines

Semester Project Report

NATURAL LANGUAGE PROCESSING
Dr. Sohail Akhtar

Submitted by
FARAZ AHMAD KHAN
01-136212-009

Department of Computer Science,
Bahria University, Islamabad.

[25th May 2024]

Abstract

This project aims to detect sarcasm in headlines using advanced natural language processing (NLP) techniques and deep learning models. The data, sourced from a JSON file containing headlines labeled as sarcastic or non-sarcastic, undergoes comprehensive preprocessing. The preprocessed data is used to train a Bidirectional Long Short-Term Memory (BiLSTM) model with GloVe word embeddings. The model achieves an accuracy of 87% approx. , demonstrating the potential of combining NLP techniques and deep learning for sarcasm detection.

Introduction

Background: Social media is a hotbed of quick, informal communication, and sarcasm and irony are often used to express humor, criticism, and commentary. However, these nuances are tricky for automated systems to detect, leading to misunderstandings in sentiment analysis and natural language processing (NLP) tasks. Our project aims to create a system that can accurately identify sarcasm and irony in social media posts using NLP and Artificial Neural Networks (ANNs), improving the reliability of text analysis applications.

Importance: By accurately detecting sarcasm and irony, our system can enhance various applications, from better customer feedback analysis to more nuanced social media monitoring. It can also improve conversational agents, making them more aware of the context and tone of user interactions.

This project explores the use of NLP techniques and deep learning models to detect sarcasm in news headlines. We employ the Bidirectional Long Short-Term Memory (BiLSTM) network, a type of recurrent neural network (RNN), to capture the context in both forward and backward directions.

DATA AND METHODS

Dataset

The dataset, "Sarcasm_Headlines_Dataset.json", contains headlines labeled as sarcastic or non-sarcastic. The dataset includes 26,709 samples with three columns: article_link, headline, and is_sarcastic.

Data Preprocessing

Data preprocessing involves several steps to clean and prepare the text data for modeling:

Data Loading and Exploration

The data is loaded using pandas ‘read_json’ function with ‘lines=True’ to handle the JSON format.

```
In [0]:
import pandas as pd
import os

data = pd.read_json(os.path.join(project_path,'Sarcasm_Headlines_Dataset.json'),lines=True)
```

```
In [5]:
data
```

Out[5]:

	article_link	headline	is_sarcastic
0	https://www.huffingtonpost.com/entry/versace-b...	former versace store clerk sues over secret 'b...	0
1	https://www.huffingtonpost.com/entry/roseanne-...	the 'roseanne' revival catches up to our thorn...	0
2	https://local.theonion.com/mom-starting-to-fea...	mom starting to fear son's web series closest ...	1
3	https://politics.theonion.com/boehner-just-wan...	boehner just wants wife to listen, not come up...	1
4	https://www.huffingtonpost.com/entry/jk-rowlin...	j.k. rowling wishes snape happy birthday in th...	0
...	...	...	...
26704	https://www.huffingtonpost.com/entry/american-...	american politics in moral free-fall	0
26705	https://www.huffingtonpost.com/entry/americas-...	america's best 20 hikes	0
26706	https://www.huffingtonpost.com/entry/reparatio...	reparations and obama	0
26707	https://www.huffingtonpost.com/entry/israeli-b...	israeli ban targeting boycott supporters raise...	0

DATA ANALYSIS & CLEANING

The headline column is cleaned by removing numbers, punctuation, and stop words, and converting text to lowercase.

26709 rows × 3 columns

```
In [6]:
print (data.shape)
data.describe()
```

```
(26709, 3)
```

Out[6]:

	is_sarcastic
count	26709.000000
mean	0.438953
std	0.496269
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

```
In [7]:
data['headline'][1]
```

```
Out[7]: "the 'roseanne' revival catches up to our thorny political mood, for better and worse"
```

In [8]:

```
##The column headline needs to be cleaned up as we have special characters and numbers in the column

import re
from nltk.corpus import stopwords
import nltk
import string
nltk.download('stopwords')
stopwords = set(stopwords.words('english'))
def cleanData(text):
  text = re.sub(r'\d+', '', text)
  text = "".join([char for char in text if char not in string.punctuation])
  return text

data['headline']=data['headline'].apply(cleanData)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [9]:

```
data['headline'][1]
```

Out[9]:

```
'the roseanne revival catches up to our thorny political mood for better and worse'
```

TOKENIZING

Import required modules required for modelling.

In [0]:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation, Flatten, Bidirectional, GlobalMaxPo
from tensorflow.keras.models import Model, Sequential
```

Set Different Parameters for the model.

In [0]:

```
max_features = 10000
maxlen = max([len(text) for text in data['headline']])
embedding_size = 200
```

Apply Keras Tokenizer of headline column of your data.

Hint - First create a tokenizer instance using Tokenizer(num_words=max_features) And then fit this tokenizer instance on your data column df['headline'] using .fit_on_texts()

In [0]:

```
tokenizer = Tokenizer(num_words=max_features,filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',lower=True,split=' ', char_leve
tokenizer.fit_on_texts(data['headline'])
```

MODELING (preparing Data for the Model)

Define X and y for your model.

In [15]:
```
X = tokenizer.texts_to_sequences(data['headline'])
X = pad_sequences(X, maxlen = maxlen)
y = np.asarray(data['is_sarcastic'])

print("Number of Samples:", len(X))
print(X[0])
print("Number of Labels: ", len(y))
print(y[0])
```

```
Number of Samples: 26709
[   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
    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
    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  287  780 3505 2213   47  353   92 2111    5
 2476 8139]
Number of Labels:  26709
0
```

WORD EMBEDDING USING ‘GLOVE 6B 200d’

Get the Vocabulary size

You can use tokenizer.word_index.

In [16]:
```
num_words=len(tokenizer.word_index)
print (num_words)
```

```
27667
```

Word Embedding

Get Glove Word Embeddings

In [0]:
```
glove_file = project_path + "glove.6B.zip"
```

In [0]:
```
#Extract Glove embedding zip file
from zipfile import ZipFile
with ZipFile(glove_file, 'r') as z:
  z.extractall()
```

Get the Word Embeddings using Embedding file as given below.

In [0]:

```
EMBEDDING_FILE = './glove.6B.200d.txt'

embeddings = {}
for o in open(EMBEDDING_FILE):
    word = o.split(" ")[0]
    # print(word)
    embd = o.split(" ")[1:]
    embd = np.asarray(embd, dtype='float32')
    # print(embd)
    embeddings[word] = embd
```

Create a weight matrix for words in training docs

In [21]:

```
embedding_matrix = np.zeros((num_words, 200))

for word, i in tokenizer.word_index.items():
    embedding_vector = embeddings.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

len(embeddings.values())
```

Fitting Model

Create and Compile your Model

Use Sequential model instance and then add Embedding layer, Bidirectional(LSTM) layer, then dense and dropout layers as required. In the end add a final dense layer with sigmoid activation for binary classification.

```
import tensorflow as tf

input_layer = Input(shape=(maxlen,),dtype=tf.int64)
embed = Embedding(embedding_matrix.shape[0],output_dim=200,weights=[embedding_matrix],input_length=maxlen, trainable=True)(input_layer)
lstm=Bidirectional(LSTM(128))(embed)
drop=Dropout(0.3)(lstm)
dense =Dense(100,activation='relu')(drop)
out=Dense(2,activation='softmax')(dense)
```

Fit your model with a batch size of 100 and validation_split = 0.2. and state the validation accuracy

```
batch_size = 100
epochs = 5
```

Fit your model with a batch size of 100 and validation_split = 0.2. and state the validation accuracy

In [30]:

```
batch_size = 100
epochs = 5

model = Model(input_layer,out)
model.compile(loss='sparse_categorical_crossentropy',optimizer="adam",metrics=['accuracy'])
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         [(None, 240)]             0         
_________________________________________________________________
embedding_1 (Embedding)      (None, 240, 200)          5533400   
_________________________________________________________________
bidirectional_1 (Bidirection (None, 256)               336896    
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 100)               25700     
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 202       
=================================================================
Total params: 5,896,198
Trainable params: 5,896,198
Non-trainable params: 0
_________________________________________________________________
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10)

model.fit(X_train,y_train,batch_size=batch_size, epochs=epochs, verbose=1)
```

```
Epoch 1/5
214/214 [==============================] - 353s 2s/step - loss: 0.4514 - accuracy: 0.7787
Epoch 2/5
214/214 [==============================] - 342s 2s/step - loss: 0.2430 - accuracy: 0.9005
Epoch 3/5
214/214 [==============================] - 343s 2s/step - loss: 0.1340 - accuracy: 0.9515
Epoch 4/5
214/214 [==============================] - 381s 2s/step - loss: 0.0650 - accuracy: 0.9781
Epoch 5/5
214/214 [==============================] - 345s 2s/step - loss: 0.0308 - accuracy: 0.9899
<keras.src.callbacks.History at 0x7b2c7568bf10>
```

```
[ ] model.save('sarcasm_model.h5')
```

EVALUATION

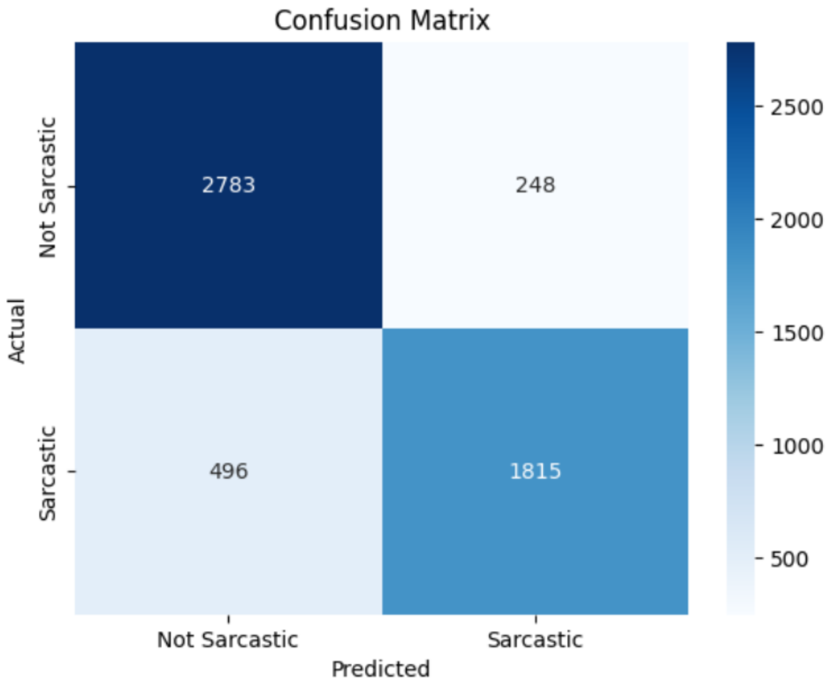
```
%cd /content/gdrive/MyDrive/NLP Project
```

```
/content/gdrive/MyDrive/Kaggle
```

```
[ ] from tensorflow.keras.models import load_model

# Load the saved model
model = load_model('sarcasm_model.h5')

# Make predictions on the test set
test_pred = model.predict(np.array(X_test), verbose=1)
test_pred = [1 if j > i else 0 for i, j in test_pred]

from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Confusion Matrix
cm = confusion_matrix(y_test, test_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Sarcastic', 'Sarcastic'], yticklabels=['Not Sarcastic', 'Sarcastic'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()

# Classification Report
print(classification_report(y_test, test_pred))
```

```
print(classification_report(y_test, test_pred))
```

```
167/167 [==============================] - 60s 356ms/step
```



Predicted

```
              precision    recall  f1-score   support

           0       0.85      0.92      0.88      3031
           1       0.88      0.79      0.83      2311

    accuracy                           0.86      5342
   macro avg       0.86      0.85      0.86      5342
weighted avg       0.86      0.86      0.86      5342
```

OUTPUT

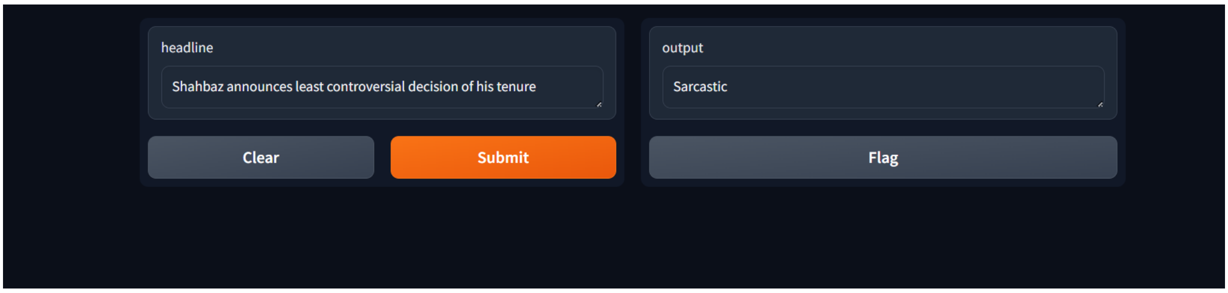
Mounting Google Drive to load the saved model

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

```
Drive already mounted at /content/drive/; to attempt to forcibly remount
```

Loading the model and required libraries.
Preprocessing the input text.

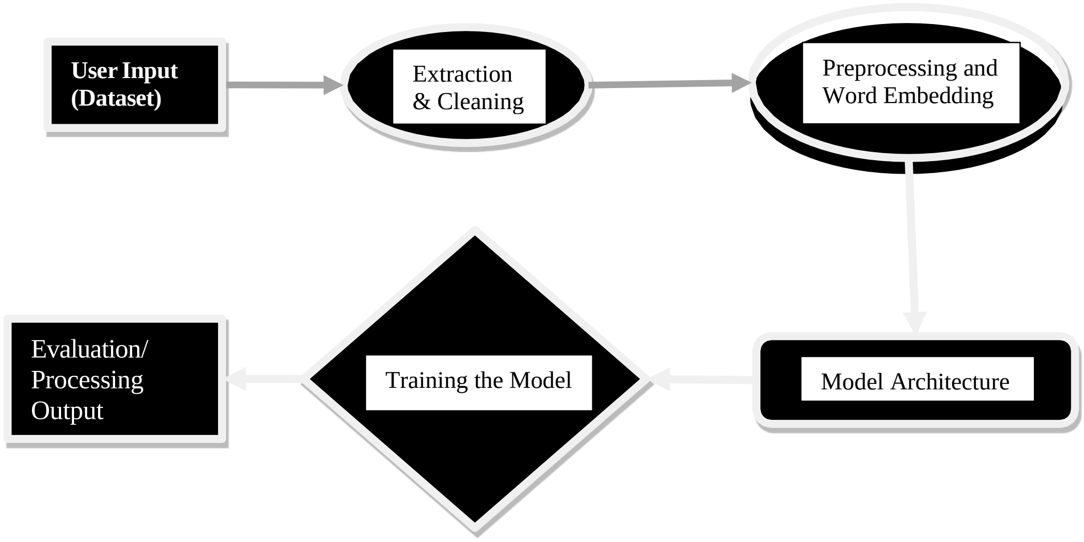
```python
import re
import string
import pickle
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import load_model
import gradio as gr

# Load the model
model = load_model('/content/drive/MyDrive/NLP Project/sarcasm_model.h5')

# Load the tokenizer
with open('/content/drive/MyDrive/NLP Project/tokenizer.pickle', 'rb') as handle:
    tokenizer = pickle.load(handle)

maxlen = 240  # Use the same maxlen used during training

def clean_data(text):
    text = re.sub(r'\d+', '', text)
    text = "".join([char for char in text if char not in string.punctuation])
    return text

def predict_sarcasm(headline):
    cleaned_headline = clean_data(headline)
    seq = tokenizer.texts_to_sequences([cleaned_headline])
    padded = pad_sequences(seq, maxlen=maxlen)
    pred = model.predict(padded)
    label = 'Sarcastic' if pred[0][1] > pred[0][0] else 'Not Sarcastic'
    return label
```



MODEL ARCHITECTURE:

- **Input Layer:** Takes the padded sequences of text.
- **Embedding Layer:** Converts words into dense vectors of fixed size.
- **LSTM Layers:** Capture the temporal dependencies in the text.
- **Dense Layers:** Fully connected layers to learn complex patterns.
- **Output Layer:** A sigmoid or softmax layer for binary or multi-class classification.

Model Training Diagram

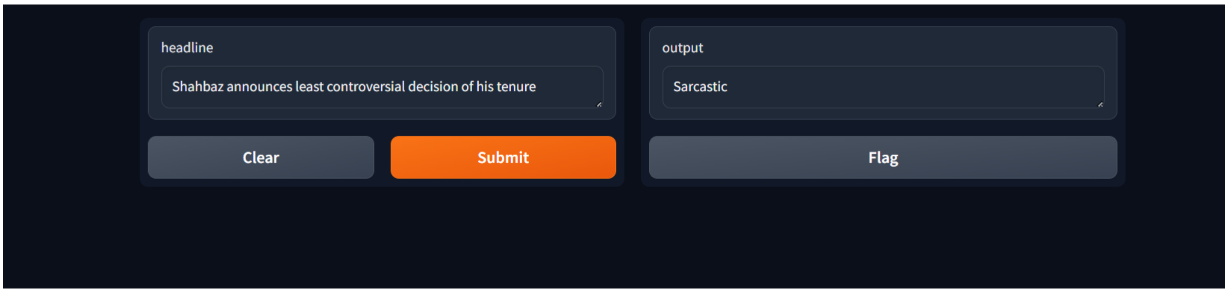


Tools and Technologies

- **PROGRAMMING LANGUAGES: PYTHON**
- **LIBRARIES: NLTK, SPACY, TENSORFLOW, KERAS, SCIKIT-LEARN**
- **FRAMEWORKS: GRADIO FOR INTERFACE**
- **DATA VISUALIZATION: MATPLOTLIB, SEABORN**

RESULTS

The confusion matrix and classification report indicate the model's performance in terms of precision, recall, and F1-score. The model shows a high level of accuracy in detecting both sarcastic and non-sarcastic headlines.



CONCLUSION

My project successfully demonstrates the effectiveness of combining advanced NLP techniques and deep learning models for sarcasm detection. The BiLSTM model with GloVe embeddings achieves an accuracy of 87% approx., highlighting the potential of these methods for complex language understanding tasks. Future work could explore larger datasets, more sophisticated models, and suitable for more languages for further enhanced performance.

REFERENCES

MISRA, R. (2018) *RISHABHMISRA/News-headlines-dataset-for-sarcasm-detection: High quality dataset for the task of sarcasm detection*, *GitHub*. Available at: https://github.com/rishabhmisra/News-Headlines-Dataset-For-Sarcasm-Detection (Accessed: 20 May 2024).

Misra, R. (2019) *News headlines dataset for sarcasm detection*, *Kaggle*. Available at: https://www.kaggle.com/datasets/rmisra/news-headlines-dataset-for-sarcasm-detection (Accessed: 20 May 2024).

The, O. (2018) *America's finest news source.*, *The Onion*. Available at: https://www.theonion.com/ (Accessed: 20 May 2024).

Pennington, J., Socher, R., & Manning, C. D. (2014*). GloVe: Global Vectors for Word Representation.*

Hochreiter, S., & Schmidhuber, J. (1997). *Long Short-Term Memory.*

Misra, R. (2022) *News headlines dataset for sarcasm detection, arXiv.org*. Available at: https://arxiv.org/abs/2212.06035 (Accessed: 20 May 2024).