PROJECT REPORT SARCASM DETECTION:

By Anah Veronica Immanuel

1. INTRODUCTION:

Sarcasm Detection is known as the Achilles Heal of NLP sentiment Analysis Problems. **Sarcasm**, which is both positively funny and negatively nasty, plays an important part in human social interaction. But sometimes it is difficult to detect whether someone is making fun of us with some irony. So, to make it easy we built something which helps you in detecting sarcastic text, but before getting into much more detail, let's define sarcasm. Sarcasm detection is a very narrow research field in NLP, a specific case of sentiment analysis where instead of detecting a sentiment in the whole spectrum, the focus is on sarcasm. Therefore, the task of this field is to detect if a given text is sarcastic or not.

2. BACKGROUND AND EXISTING SOLUTIONS (Business Understanding):

Over the years, researchers have come up with techniques to detect sarcasm by studying the properties to sarcasm. To understand this better we need to first define clearly what sarcasm is. Sarcasm can be defined **as the use of irony to mock or convey contempt**. It is a unique play of words where the person says one thing to mean the opposite to highlight absurdity. A method to convey contempt in a funny yet mocking way. This is called **linguistic incongruity.**

- **Lexical Incongruity:** As explained in the *paper 'sarcasm detection in tweets' by Ashwin Bhat and Kalpesh Patil.* The linguistic theory of Context Incongruity suggests that the common form of sarcasm expression consists of a positive sentiment which is

contrasted with a negative situation. Well defined and extensively used statistical models can benefit from the use of features generated on the basis of well-established linguistic theories. Number of times a positive word is followed by a negative word and Co-existence of positive and negative. Other features highlighted by *sarcasm detection in tweets by Ashwin Bhat and Kalpesh Patil,* include Lexical Polarity and Lexical Subjectivity.

- **Lexical Polarity:** Lexical Polarity is the overall polarity of the sentence. It has been observed that sarcastic sentences tend to have a more positive polarity. This is intuitive since sarcasm is used to convey contempt or mockery.

- **Lexical Subjectivity:** This is a measure of the overall subjectivity of the sentence. Sarcastic sentences tend to be more subjective than objective. A person is very likely to be sarcastic about things that they are more likely affected by. Also, a person shares their though process and how they perceive through sarcasm. All this highlights that sarcastic statements are highly subjective.

- **Pattern Based approach:** In their paper "A pattern-based approach for sarcasm detection 0n Twitter" Mondher Bouazizi and Tomoaki Otsuki discuss using a pattern-based matching for sarcasm detection. The proposed method in the paper first POS (Part of speech) tags the sentences. The POS tags of highly emotional sentences are observed and analysed. The sarcastic patterns extracted by POS tagging is then used as a contextual feature to identify sarcasm.

- **Word Similarity:** In their Paper, Word similarity score as augmented feature in sarcasm detection using deep learning Joseph Tarigan and Ganda Grisang discuss using the similarity scores of the words in the sentence. This idea is based on incongruence. But its implementation is different. The approach is to use word vectors embedding to first convert each of the word in the sentence to a number and then measure the similarity scores between each word. This is intuitive because non sarcastic words tend to have overall similar scores, however sarcastic sentences vary in their similarity. This is because of the incongruence of the sentence.

3. IMPLEMENTATION OF OUR SOLUTION

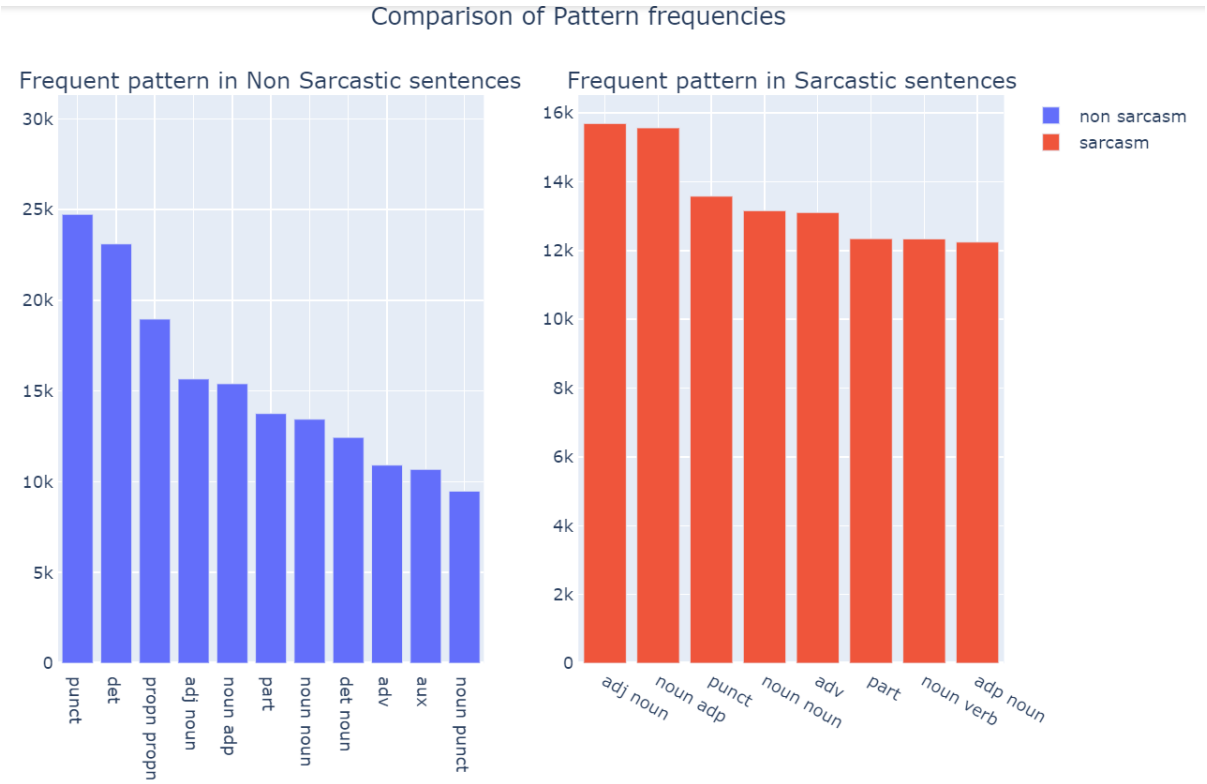
In The implementation of this solution, I will use the approaches used by the research papers explored to feature engineer our data and then feed it to the model. Once the data is pre-processed and ready, I will first try out Logistic Regression, SVC and XGBoost to perform training and validation. We will then use deep learning models to improve the performance.

- **DATA:** The dataset is provided by Kaggle website and can be accessed using this link: https://www.kaggle.com/rmisra/news-headlines-dataset-for-sarcasm-detection. The dataset consists of news headlines that are labelled as either sarcastic or non-sarcastic.
  The data set has 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. Useful in collecting supplementary data

- **DATA EXPLORATION AND ANALYSIS:**
  The Dataset consists of about **55328 rows and 3 columns** and does contains **no null values**. Building word clouds showed that non sarcastic news headlines were more about politics and sarcastic news headline were more varied in their topic of coverage. The data has almost equal number of non-sarcastic and sarcastic headline so I am going to use accuracy as the metric to validate.

- **DATA PRE-PROCESSING:**
  I am going to Pre-process the data and perform feature engineering based on the features that I learnt about during the research phase. This is going to create **contextual Lexical features** that we will feed our model.

  - **Common Patterns in sarcastic sentences:** To Identify common patterns, I first POS Tagged each sentence in the headline and created a POS sequence by joining the POS tags together. I created a function **frequent_pattern_finder** that broke the POS tag sequences to n-grams of 1 to 3 length sequences to identify unique patterns and count the number of times a pattern appeared overall. Then I pulled the most common sequences identified in sarcastic and non-sarcastic headline.



Non sarcastic headlines had **'det', 'det noun', 'punct' and 'punct noun'** patterns significantly higher than sarcastic headlines. I used this character to create four columns 'det_count', 'det_noun_count', 'punct_count' and 'punct noun_count' that counts the number of times each pattern occurs in the headline.

- **Incongruity Measure:** I create a function called **'incongruity measure'** which first tokenized each work in a sentence, found the polarity score of each word in the sentence using TextBlob. I round the polarity values to 1 for positive, -1 for negative and 0 for neutral. I then used the entropy function to the find the measure of disorder in the polarity values. I used the entropy value and the incongruity measure.
- **Word Similarity:** To find similarity scores, I used glove model pre trained vectors, glove-wiki-gigaword-50 that I downloaded using Genism downloader api. I tokenized, lemmatized then found the word vector of each word. Using the word vectors, I created the similarity matrix of the words

in the sentence. I then wrote a function **'get_feature'** that will use the similarity matrix to get four column features, namely **max similarity word pair score**, **min similarity word pair score, max dissimilarity word pair score and min dissimilarity word pair score.** Using these columns, I created **'difference in similarity' and 'difference in dissimilarity'** by subtraction and max and min of similarity and dissimilarity scores.

- **MODELING:**
  Split the data into train, test and validation files and upload to s3 buckets so our model will be able to use it. I am going to use Logistic Regression, SVC, XGBoost and LSTM to model the data and see which performs the best. Logistic Regression is my benchmark.

  - **LOGISTIC REGRESSION**: Logistic regression is a binary classifier. The logistic model is used to model the probability of a certain class or event. It tries to classify the data using a regression line and then maps it to 0 or 1 class by using the sigmoid function. Fit and train the model with X_train and Y_train.

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Validation:

```
Accuracy score of logistic regression model 0.6629607316939592
F1 score of logistic regression model 0.653490990990991
Recall score of logistic regression model 0.617866269165247
Precision score of logistic regression model 0.6934751434034416
```

Our model gave us an accuracy of 66% percent. We will see if we get a better performance from SVC.

- **SVC:** A **Support Vector Machine** (**SVM**) is a discriminative **classifier** formally defined by a separating hyperplane. In other words, given labelled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. Fit and train the model with X_train and Y_train.

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Validation:

```
Accuracy score of svc model 0.66723259762309
F1 score of svc model 0.6426723123970829
Recall score of svc model 0.6327003242241779
Precision score of svc model 0.652963671128107
```

Our SVC Model performed very similarly to the logistic regression model. Now Let's try XGBoost algorithm and analyse its performance.

- **XGBoost:** XGBoost is a tree-based model that's a boosted algorithm of decision tree that's uses the mechanism of bootstrapping and bagging. Let's see how well this model performs in our dataset.

I built this model using sagemaker.

```
xgb = sagemaker.estimator.Estimator(container, #Build model
                                    role,
                                    train_instance_count=1,
                                    train_instance_type='ml.m4.xlarge',
                                    output_path='s3://{}/{}/output'.format(session.default_bucket(), prefix),

                                    sagemaker_session=session)
```

Validation:

```
Accuracy score of svc model 0.66723259762309
F1 score of svc model 0.6426723123970829
Recall score of svc model 0.6327003242241779
Precision score of svc model 0.652963671128107
```

XGBoost performed very poorly with our dataset using lexical features. I am now going to try and see is LSTM is able to capture the features of our dataset and perform well.

- **LSTM Model:**

  I am only going to word embed the sentences and feed it into and LSTM and it's will be able to analyse the features of the sentences using the embedded format that we provided it. According to Wikipedia, **Long short-term memory** (**LSTM**) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition[3][4] and anomaly detection in network traffic or IDSs (intrusion detection systems). This makes this model vey ideal for our use case.

  GloVe: GloVe stands for global vectors for word representation. It is an unsupervised learning algorithm developed by Stanford for generating word **embeddings** by aggregating global word-word co-occurrence matrix from a corpus. The resulting **embeddings** show interesting linear substructures of the word in vector space.
  After Embedding our word vectors, we will then build our Keras model.

```
model = Sequential()
model.add(embedding_layer)
model.add(LSTM(64, dropout=0.2, recurrent_dropout=0.25))
model.add(Dense(1, activation='sigmoid'))

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

```
Summary of the built model...
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 50, 50)            1537250
_________________________________________________________________
lstm (LSTM)                  (None, 64)                29440
_________________________________________________________________
dense (Dense)                (None, 1)                 65
=================================================================
Total params: 1,566,755
Trainable params: 29,505
Non-trainable params: 1,537,250
_________________________________________________________________
None
```

Fit the model and set metric as accuracy.

Validation:

```
Epoch 1/3
462/462 [==============================] - 30s 66ms/step - loss: 0.3333 - acc: 0.8525 - val_loss: 0.3494 - val_acc: 0.8444
Epoch 2/3
462/462 [==============================] - 30s 64ms/step - loss: 0.3277 - acc: 0.8544 - val_loss: 0.3537 - val_acc: 0.8416
Epoch 3/3
462/462 [==============================] - 29s 64ms/step - loss: 0.3235 - acc: 0.8575 - val_loss: 0.3399 - val_acc: 0.8506
```

As you can see that we have a validation accuracy of 85% which beat our base model of 66%. Also, the validation accuracy is not way below the train accuracy which is a good sign that this model is not overfitting. I am going to deploy this model in AWS.

4. DEPLOYMENT:

I used “Deploy trained keras or tensorflow models in AWS” from AWS documentation to deploy this model. We can now connect using an api to connect out model to external applications.

```
predictor.endpoint
```

```
'sagemaker-tensorflow-2020-06-28-22-16-12-898'
```

5. **Future Improvements:** As a part of the future development of this project, Including the sentiment difference in the title and the body of the article can be feature engineered. If given more context about the situation in question positive comments about negative situation and negative comments about positive situation can we flagged as potential sarcastic comments and then analysed further to check for sarcastic features.