CIS 668: NATURAL LANGUAGE PROCESSING

QUORA SINCERE AND INSINCERE QUESTIONS FILTERING

Syracuse University
Professor: Lu Xiao

Team Members :
Rishitha Vanamala
Vineela Gudipati
Nilesh Bhoi
Chinmay Maganur

Table of Contents

Abstract

Online social networks (OSNs), in specific, and the internet in general continue to play a key role in our lives as information is posted and distributed in vast volumes. Because of their prominence and relevance, these communication channels are frequently abused for a variety of reasons. Some individuals utilize OSNs to upload their deceptive or dishonest content because they are motivated by objectives like marketing and financial benefits.
In this context, we tested several filters and algorithms to remove spam and untruthful information using a real-world dataset that Quora uploaded on Kaggle.com. We evaluated various preprocessing and analytic models. We also looked into the possibility that users' cognitive activities during post-writing could improve prediction accuracy. We provided a list of the sincerity prediction models with the highest efficiency.

1. Introduction:

Online social networks (OSNs) include unrestricted information, allowing members to post almost whatever they wish in freeform language. Additionally, they might disseminate information that is wholly false or misleading. The capacity to enforce and verify the integrity of content on websites is still missing; for instance, the content may be false or erroneous.

On the internet, information credibility is a major issue. For instance, numerous publications suggested that internet goods may have fraudulent reviews that are fabricated and then placed to trick readers. Such reviews aim to either enhance the reputation of products by offering extremely positive ratings or harm it by providing extremely negative ones. When they disparage a good product that most other reviewers believe to be true, when they recommend a product of poor quality that most other reviewers believe to be true, or when they incorrectly praise or denigrate a product of average quality, these types of fabricated, altered reviews can be especially dangerous. In addition to posting false product reviews, individuals can also submit false news and inaccurate information as though they were true. Such material might become well-known and appear more prominently in search results than more accurate information. For instance, a student trying to research a location, an event, or a famous person online can come upon one of the widely read yet inaccurate pieces and consider it as the main source of accurate information. In other words, using the internet as your main information source can lead to a lot of issues because search engines prioritize popularity above information credibility or veracity.

When determining the accuracy of information, three main factors should be taken into account: the website that is hosting the post, the author or writer of the post, and the information of the post. These three things are interdependent on one another. For instance, a reliable website might only allow reliable authors or contents, or it might have filters that exclude unreliable authors and contents. In a similar vein, reliable writers typically publish reliable content on reliable websites.

Should websites have the authority to ban or remove harmless but sincere comments? Websites may prohibit such actions for a variety of contradictory reasons. They must strike a balance between growing their audience, emphasizing quantity, and offering their devoted users great content that has been vetted. Websites attempting to address this issue will encounter a variety of difficulties. These websites don't want to come across as "controlling what should be posted," as being against free speech, or as preventing their users from voicing their opinions.

Although still in their infancy, the mechanisms that automatically determine whether a newly created response is incredible may produce a large number of false positives or negatives. Instead, it takes a lot of time and effort to manually or by humans identify and remove amazing content. Some verifying the facts websites, like snopes.com, employ human experts to evaluate the veracity of claims and content. Instead of focusing on fraudulent reviews, these websites evaluate claims or fake news.

In this study, we utilize data analytics to evaluate the legitimacy of a query. Analyzing user-generated content in Online Social Networks (OSNs) has been shown to be a useful application of data analytics. Data analytics can assist in creating scalable methods to identify fraudulent and misleading content. To achieve our specific goals and objectives, we used a unique real-world data collection obtained from the Quora website. These goals include:

1. Investigate how text preprocessing and feature representation can aid in detecting fake news on social media platforms.
2. Evaluate the performance of various supervised machine learning algorithms, such as decision trees, linear SVC, logistic regression, and random forests, in detecting dishonest content when given various data representations.
3. Examine the cognitive processes involved in user-generated contributions and how it contributes to identifying fake content.

2. Systematic review of literature

2.1. Information Reliability

The assessment of information credibility on the internet is often lacking due to various factors, such as the absence of quality control mechanisms . Credibility is commonly associated with the accuracy, truthfulness, or factual basis of information. However, a significant portion of the content on online social networks (OSNs) consists of personal opinions, which may not necessarily have an objective basis. OSN users discuss various topics, such as news events, celebrities, politics, and fashion, among others.

Numerous studies have examined signs of deception in OSN posts. Unlike face-to-face encounters, eye contact, gaze aversion, and shrugs are inappropriate on OSNs. In order to combat this, researchers created a fresh set of indicators for spotting dishonesty in OSNs, which take into account elements including phrase length, complexity, mood, text informality, and emoticon usage. Their research revealed, among other things, that liars frequently employ brief sentences.

2.2. Fake reviews

In this section, we'll examine a few studies that investigated the issue of fake reviews. Both legal and illegal transactions might result in fake reviews. Vendors may pay for reviewers to purchase and evaluate their products, or they may incentivize reviewers to write positive reviews. Conversely, vendors may also attempt to post negative reviews on their competitors' products.

The prevalence of "duplicate or repeated reviews" is one factor associated with fraudulent reviews that has received a lot of attention. This method is frequently utilized as a crucial sign of online spam reviews in the literature. It is assumed that spammers will frequently write the same evaluations. Jindal and Liu used duplicate reviews as a positive training dataset to create a logistic regression model for identifying non-duplicate spam reviews that exhibit similar characteristics. Meta-features pertaining to the reviews and reviewers should also be taken into consideration to improve the detection's accuracy. The model's ability to predict non-duplicate reviews is next tested using outlier reviews, or reviews that dramatically vary from the average product rating.

3. Data and framework

We describe our data and experiment structure in this section with the aim of developing approaches to recognize dishonest materials in the Quora dataset.

3.1. Data:

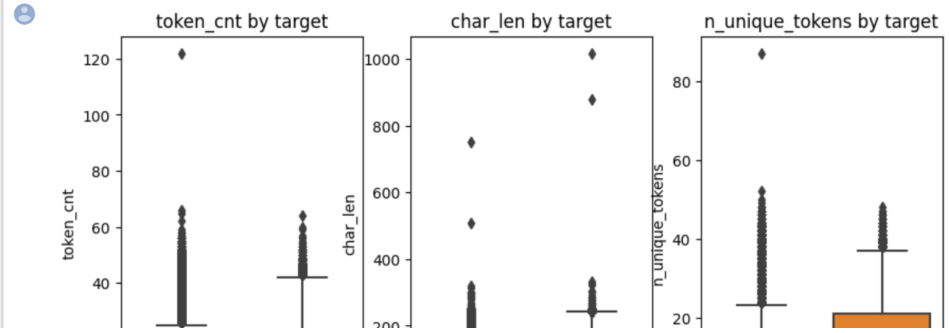
The data for this study came from the Quora website. (https://www.kaggle.com/c/quorainsincere-questions-classification/data). Each data entry contains the question that was posed as well as whether or not it was deemed to be sincere (label = 1) or not (label = 0).

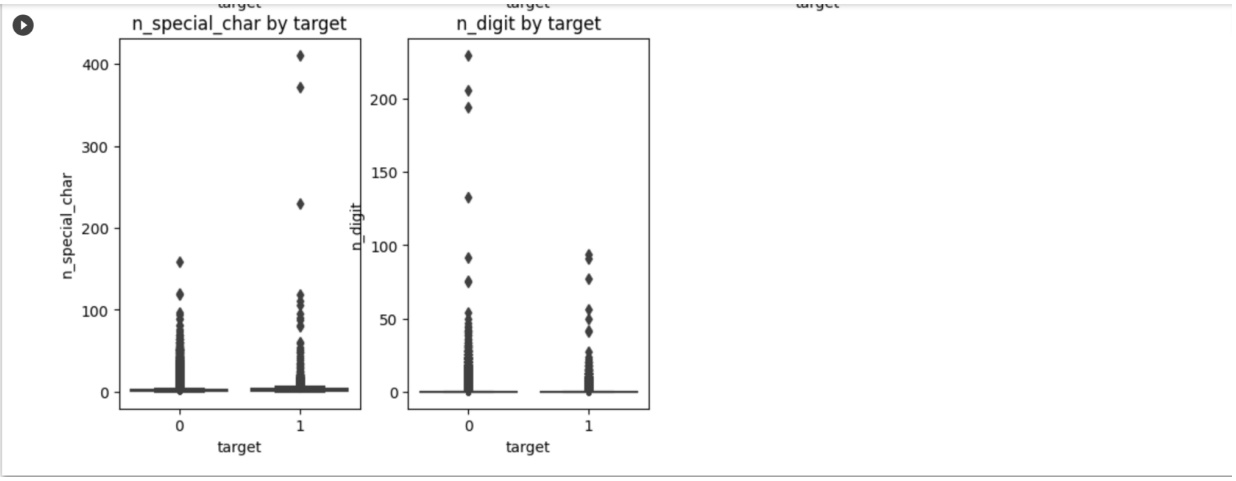
Question text	Label
"What a difference between religion and spirituality?"	(0, sincere)
"Why are religious folks not considered clinically insane?"	(1, insincere)

We conducted an analysis of the data, which involved various tasks such as determining the length of the longest token and character, identifying the data line containing this token, and calculating several metrics for each data item, including token count, character length, unique tokens, and the number of special characters and digits present.

Based on our analysis, it has been observed that insincere questions have a greater count of tokens and character length compared to sincere questions. Furthermore, insincere questions also exhibit a higher number of unique tokens when compared to sincere questions.

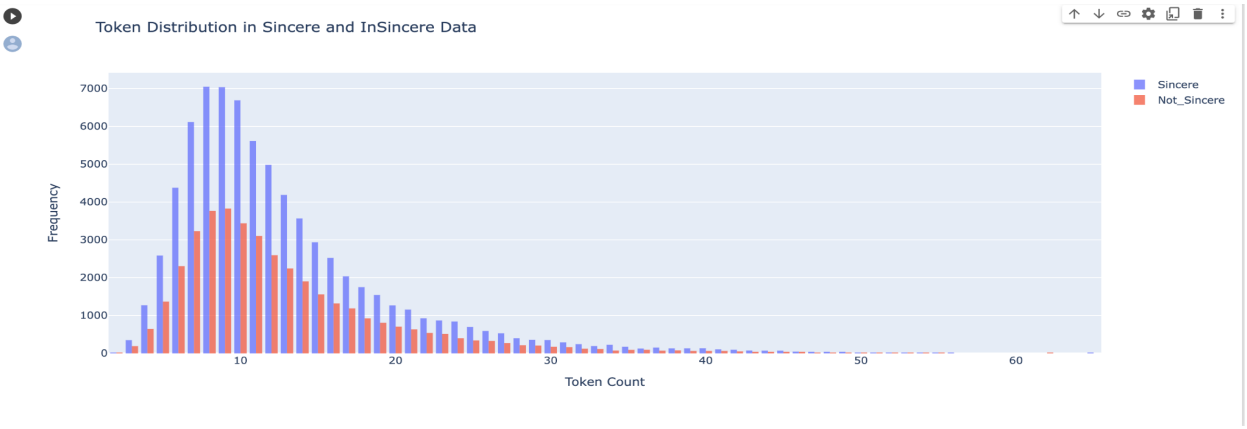
```
fig, ax = plt.subplots(nrows=2, ncols=3, figsize=(10, 10))
num_cols = ['token_cnt', 'char_len', 'n_unique_tokens', 'n_special_char', 'n_digit']

for col, axes in enumerate(ax.ravel()):
    if col < len(num_cols):
        sns.boxplot(x='target', y=num_cols[col], data=df, ax=axes)
        axes.set_title(f'{num_cols[col]} by target')
    else:
        axes.axis('off')
```





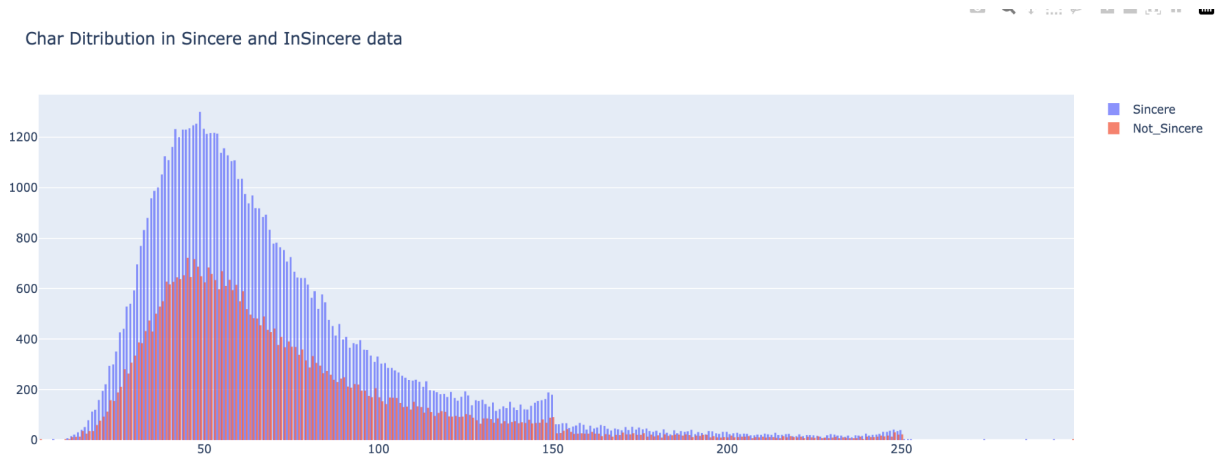
The graph illustrates the distribution of tokens in both sincere and insincere questions data.

```
%matplotlib inline
#
sin = df[df['target']==0]['token_cnt']
ins = df[df['target']==1]['token_cnt']

trace1 = go.Histogram(
        x=sin, name='Sincere',
        opacity=0.75
    )
trace2 = go.Histogram(
        x=ins, name = 'Not_Sincere',
        opacity=0.75
    )

data = [trace1, trace2]
layout = go.Layout(
    title='Token Distribution in Sincere and InSincere Data',
    xaxis=dict(title='Token Count'),
    yaxis=dict(title='Frequency')
)
fig = go.Figure(data=data, layout=layout)
fig.show(renderer="colab")
```



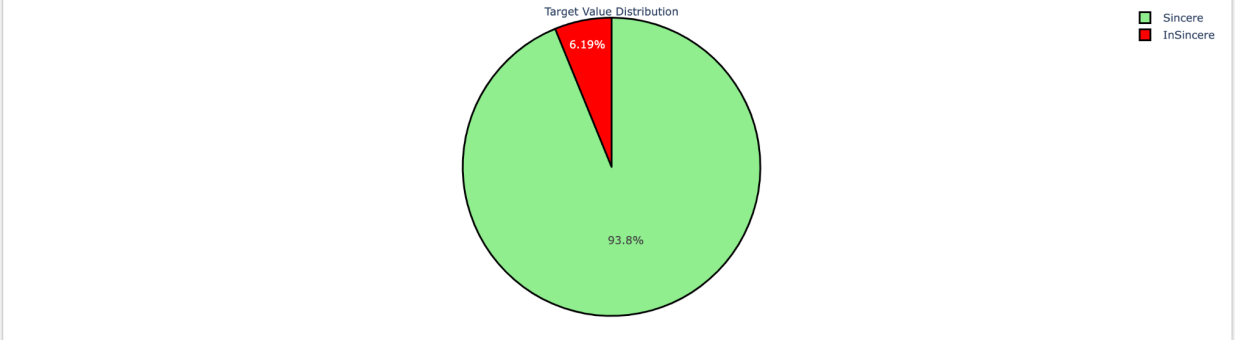
The graph represents the distribution of characters in both sincere and insincere questions data.

```
%matplotlib inline
#
sin = df1[df1['target']==0]['char_len']
ins = df1[df1['target']==1]['char_len']

trace1 = go.Histogram(
        x=sin, name='Sincere',
        opacity=0.75
    )
trace2 = go.Histogram(
        x=ins, name = 'Not_Sincere',
        opacity=0.75
    )

data = [trace1, trace2]
layout = go.Layout( title='Char Ditribution in Sincere and InSincere data')
fig = go.Figure(data=data, layout=layout)
fig.show(renderer="colab")
```



The graph depicts the distribution of target values for both sincere and insincere questions in the dataset.

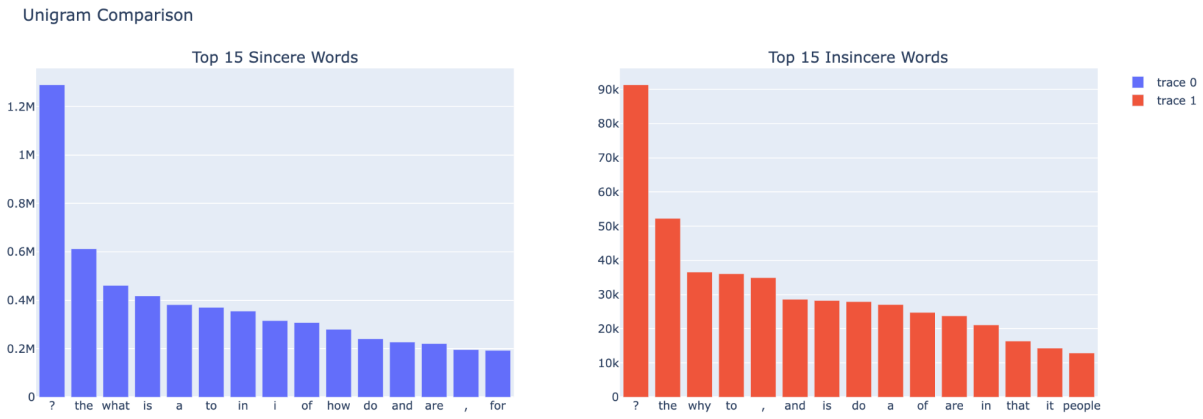
```
labels = ['Sincere','InSincere']
values = df['target'].value_counts().values
colors = ['lightgreen','red']

fig = go.Figure([go.Pie(labels = labels,values = values,title='Target Value Distribution')])
fig.update_traces(hoverinfo = "label+percent",marker=dict(colors=colors,line=dict(color='#000000', width=2)))
fig.show(renderer="colab")
```



The below graph showcases the top 15 most frequently occurring words for both sincere and insincere questions in the dataset

```
[ ] token_sin = nltk.word_tokenize(sin_corpus)
    token_ins = nltk.word_tokenize(ins_corpus)
```

```
unigram_sin = nltk.FreqDist(token_sin).most_common(15)
unigram_ins = nltk.FreqDist(token_ins).most_common(15)
key_sin =  [k for k,v in unigram_sin]
val_sin =  [v for k,v in unigram_sin]

key_ins =  [k for k,v in unigram_ins]
val_ins =  [v for k,v in unigram_ins]



# Create a figure with one row and two columns of subplots, each with a title
fig = make_subplots(rows=1, cols=2, subplot_titles=['Top 15 Sincere Words', 'Top 15 Insincere Words'])

# Add the bar chart for the sincere words to the first subplot
fig.add_trace(
    go.Bar(x=key_sin, y=val_sin),
    row=1, col=1
)

# Add the bar chart for the insincere words to the second subplot
fig.add_trace(
    go.Bar(x=key_ins, y=val_ins),
    row=1, col=2
)

# Set the title of the figure
fig.update_layout(title="Unigram Comparison")

# Show the figure using the default renderer
fig.show(renderer="colab")
```



3.2. Data Preprocessing:

To prepare the data for further processing, we initially duplicated the dataset and converted all text to lowercase. Next, we generated multiple versions of the lowercase dataset by applying various combinations of preprocessing techniques. The overall procedure comprised five distinct stages:

1.The process involves changing all the letters in the alphabet to lowercase format, for instance, changing the word "Word" to "word."
2. A pre-established dictionary of contractions is utilized to expand contractions, such as replacing "shouldn't" with "should not."
3. We will utilize a special-character dictionary map that has been predefined to eliminate non-English language characters from the text and to insert spaces around punctuation and special characters. For instance, "well, resumé" will be changed to "well , resume," and an exponent such as "2" will be converted to "2."
4. We will correct the 50 most frequent misspellings in the dataset by compiling a list of words that do not appear in the embeddings. As an example, "qoura" will be replaced with "quora."
5. We removed all emojis present in the dataset. Emojis were removed to ensure that the model trained on the dataset would not be biased towards the presence or absence of emojis. To achieve this, we used a regular expression to match all emojis and replaced them with an empty string.
6. We removed all punctuations from the text data. Punctuations were removed as they are typically not useful for text analysis and machine learning models. To remove punctuations, we used the string.punctuation method from the Python string library.
7. The last preprocessing task involved removing all mathematical equations present in the text data. The equations were removed as they do not provide any meaningful

information for text analysis. To remove the equations, we used a regular expression to match all text enclosed within [math] and [\math] tags and replaced it with an empty string.

```
#removing stopwrds may change the meaning of sentence for example below sentence went from negative to postive.

def clean_text(text):

  text = remove_maths_links(text)
  print(f"After removing Maths notations:  {text}")
  text = clean_contractions(text)
  print(f"After removing contractions:  {text}")

  text = clean_punct(text)
  print(f"After removing punctuations:  {text}")
  text = remove_stopwords(text)
  print(f"After removing stopwords: {text}")

  #print()
  text = lemmatize(text)
  print(f"After lemmatizing:  {text}")
  return text



clean_text("This is why i don't want icecream peace ✌. [math]a**2 = 4*4 [/math]")
```

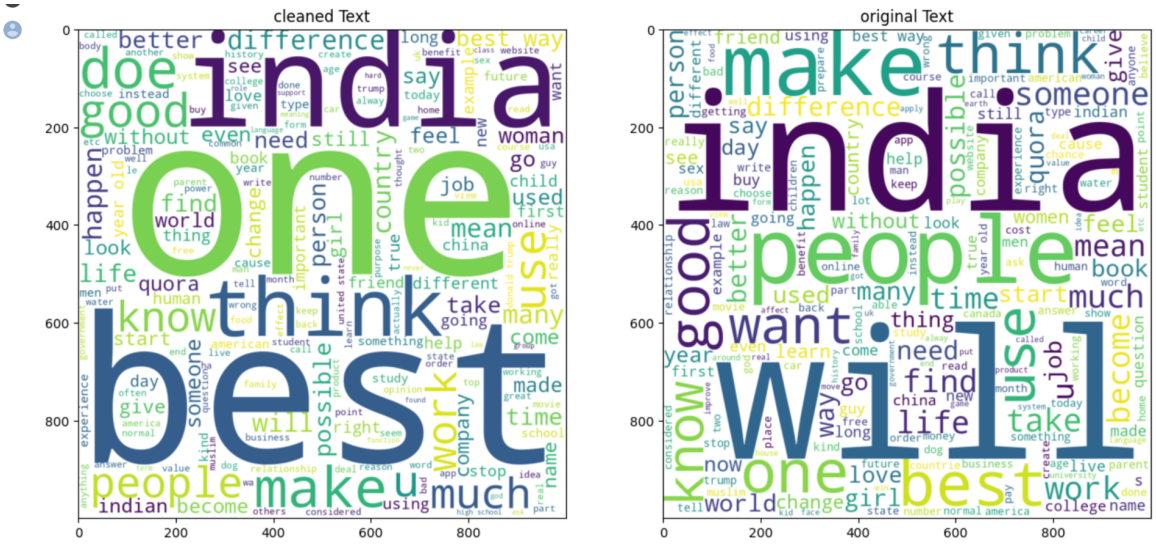
```
After removing Maths notations:  This is why i don't want icecream peace ✌. maths equation
After removing contractions:  This is why i do not want icecream peace ✌. maths equation
After removing punctuations:  This is why i do not want icecream peace  maths equation
After removing stopwords: This want icecream peace maths equation
After lemmatizing:  this want icecream peace math equation
'this want icecream peace math equation'
```

Word clouds displaying cleaned or pre-processed text and original text:

```
from wordcloud import WordCloud
```

```
fig ,axes = plt.subplots(1,2,figsize=(15,10))

texts = ' '.join(df['clean_text'])
wordcloud_stp_removed = WordCloud(width = 1000, height = 1000, background_color ='white', min_font_size = 10).generate(texts)
axes[0].imshow(wordcloud_stp_removed)
axes[0].set_title('cleaned Text')

texts_stp = ' '.join(text.lower() for text in df['question_text'])
wordcloud_stp = WordCloud(width = 1000, height = 1000, background_color ='white', min_font_size = 10).generate(texts_stp)
axes[1].imshow(wordcloud_stp)
axes[1].set_title('original Text')
```

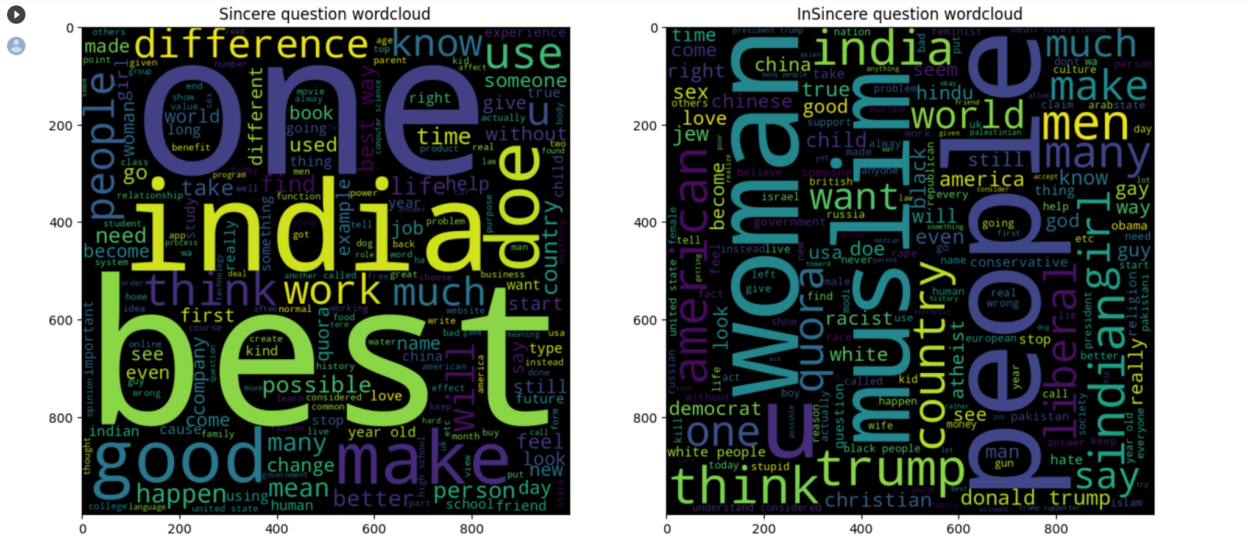
```
Text(0.5, 1.0, 'original Text')
```



Word clouds displaying Sincere questions and Insincere questions:

```
fig ,axes = plt.subplots(1,2,figsize=(15,10))

texts_sin = ' '.join(df[df['target']==0]['clean_text'])
wordcloud_sin = WordCloud(width = 1000, height = 1000,background_color ='black', min_font_size = 8).generate(texts_sin)
axes[0].imshow(wordcloud_sin)
axes[0].set_title('Sincere question wordcloud')

texts_ins = ' '.join(df[df['target']==1]['clean_text'])
wordcloud_ins = WordCloud(width = 1000, height = 1000,background_color ='black', min_font_size = 8).generate(texts_ins)
axes[1].imshow(wordcloud_ins)
axes[1].set_title('InSincere question wordcloud')
```

```
Text(0.5, 1.0, 'InSincere question wordcloud')
```



In conclusion, the preprocessing tasks performed on the dataset involved cleaning the text data by removing emojis, punctuations, mathematical equations, and substituting

contractions. These tasks aimed to ensure that the text data is consistent, free from unwanted characters, and suitable for machine learning models.Before starting the model training process, we perform preprocessing on all the question text present in the train and dev sets.

3.3. Data Modelling

3.3.1 TFIDF and Logistic regression:

TFIDF:

TFIDF is a widely used technique in natural language processing for text analysis and information retrieval. The model aims to assign a weight to each term in a document based on its frequency and rarity in the entire corpus.

TFIDF assigns weights to a token in such a way that the most frequent words are weighed less as compared to rare words. Most frequent words are ‘the’,’is’,’a’ called stop words.

Logistic regression:

Logistic Regression is a classification algorithm that uses sigmoid functions which caps the value between 0 and 1. In our model we have used TFIDF+Logistic Regression to predict whether a given question is Sincere or Not.

Steps:

We divide the data into train, val and test in a proportion of 60%,30% and 10% respectively.

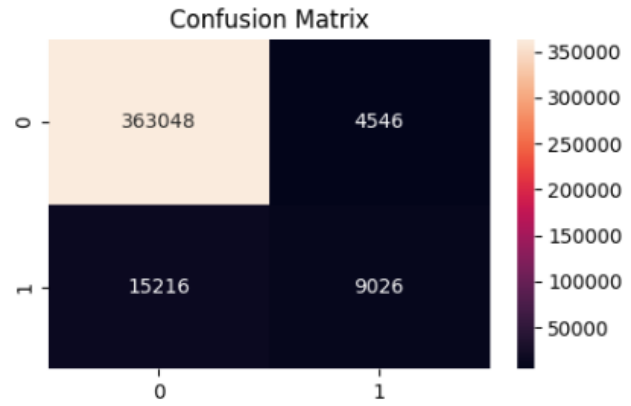
```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, stratify = y,random_state=22)
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,stratify = y_train, test_size=0.1, random_state=22)

print('X_train: ', X_train.shape, y_train.shape)
print('X_test: ',X_test.shape, y_test.shape)
print('X_val: ',X_val.shape, y_val.shape)
```

Num_of_features for TF IDF was set as 100000 and we had considered bigram as input to TFIDF. Following this we normalized all the numerical functions using Standard Scalar function.

```
#since we are using Logistic regression scaling data play imp role

tfidf = TfidfVectorizer(max_features =100000,stop_words='english',ngram_range=(1,2))
tf_fit = tfidf.fit(df['clean_text'])
tf_train = tfidf.transform(X_train['clean_text'].values)
tf_val = tfidf.transform(X_val['clean_text'].values)
tf_test = tfidf.transform(X_test['clean_text'].values)

sc = StandardScaler()
x_train_tc = sc.fit_transform(X_train['token_cnt'].values.reshape(-1, 1))
x_val_tc = sc.fit_transform(X_val['token_cnt'].values.reshape(-1, 1))
x_test_tc = sc.fit_transform(X_test['token_cnt'].values.reshape(-1, 1))

sc_char = StandardScaler()
x_train_ch = sc_char.fit_transform(X_train['char_len'].values.reshape(-1, 1))
x_val_ch = sc_char.fit_transform(X_val['char_len'].values.reshape(-1, 1))
x_test_ch = sc_char.fit_transform(X_test['char_len'].values.reshape(-1, 1))

sc_unique = StandardScaler()
x_train_uq = sc_unique.fit_transform(X_train['n_unique_tokens'].values.reshape(-1, 1))
x_val_uq= sc_unique.fit_transform(X_val['n_unique_tokens'].values.reshape(-1, 1))
x_test_uq = sc_unique.fit_transform(X_test['n_unique_tokens'].values.reshape(-1, 1))

sc_special = StandardScaler()
x_train_sp = sc_special.fit_transform(X_train['n_special_char'].values.reshape(-1, 1))
x_val_sp = sc_special.fit_transform(X_val['n_special_char'].values.reshape(-1, 1))
x_test_sp = sc_special.fit_transform(X_test['n_special_char'].values.reshape(-1, 1))


sc_digit = StandardScaler()
x_train_dg = sc_digit.fit_transform(X_train['n_digit'].values.reshape(-1, 1))
x_val_dg = sc_digit.fit_transform(X_val['n_digit'].values.reshape(-1, 1))
x_test_dg = sc_digit.fit_transform(X_test['n_digit'].values.reshape(-1, 1))
```

Our Model was able to predict 97% of data belonging to class 0 and 37% belonging to class 1.

```
def plot_confusion_matrix(y_test, y_pred):
  cm = confusion_matrix(y_test, y_pred)
  plt.figure(figsize=(5, 3))
  sns.heatmap(cm, annot=True, fmt='d')
  plt.title('Confusion Matrix')
  plt.show()

  print(f"Correctly classified sincere questions: {round(cm[0][0]/(cm[0][0] + cm[0][1])*100, 2)}%")
  print(f"Correctly classified insincere questions: {round(cm[1][1]/(cm[1][0] + cm[1][1])*100, 2)}%")

plot_confusion_matrix(y_val,pred)
```




```
Correctly classified sincere questions: 98.76%
Correctly classified insincere questions: 37.23%
```

```
metrics=pd.DataFrame({'Accuracy':accuracy,'Precision':precision,'Recall':recall,'f1_score':f1},index=algo)
metrics
```

	Accuracy	Precision	Recall	f1_score
LogisticRegression	0.949566	0.665046	0.372329	0.477389

3.3.2 BI-LSTM:

Bidirectional Long Short Term Memory is a type of RNN which can be used for Sequence-to-Sequence tasks. Advantage of Using Bi-LSTM over LSTM is that it considers previous as well as future tokens which makes the model more accurate.

We have used embedding dim as 64 with vocab_length the same as above . Additionally we have used Bi-LSTM of 256 units followed by a Dense layer of 32 units. This resulted in a f1-score of 71% which was more than Logistic Regression.

```
max_length=50
vocab_size=100000
embedding_dim=64
trunc_type="post"
oov_tok="<OOV>"
padding_type="post"
```

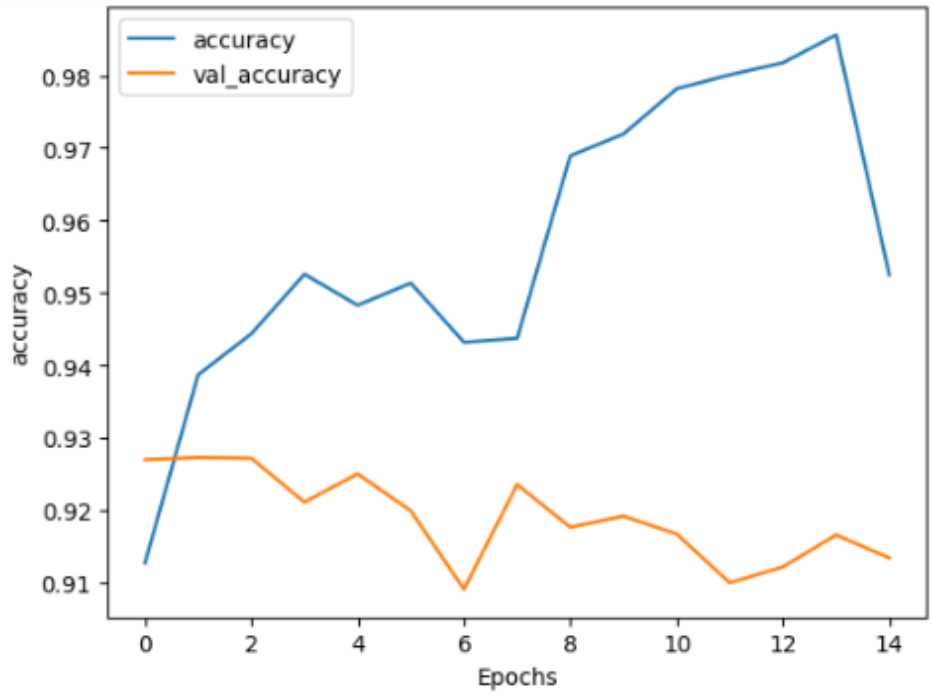
```
tokenizer = Tokenizer(num_words=vocab_size,oov_token=oov_tok)  #tokenize the sentences
tokenizer.fit_on_texts(df_sample['clean_text'])                #fit the tokens
word_index = tokenizer.word_index
x_train_seq = pad_sequences(tokenizer.texts_to_sequences(X_train), maxlen=max_length, padding=padding_type, truncating=trunc_type)  #padd the sentences if its less than max_length
x_test_seq = pad_sequences(tokenizer.texts_to_sequences(X_test), maxlen=max_length, padding=padding_type, truncating=trunc_type)    #padd the sentences if its less than max_length
```

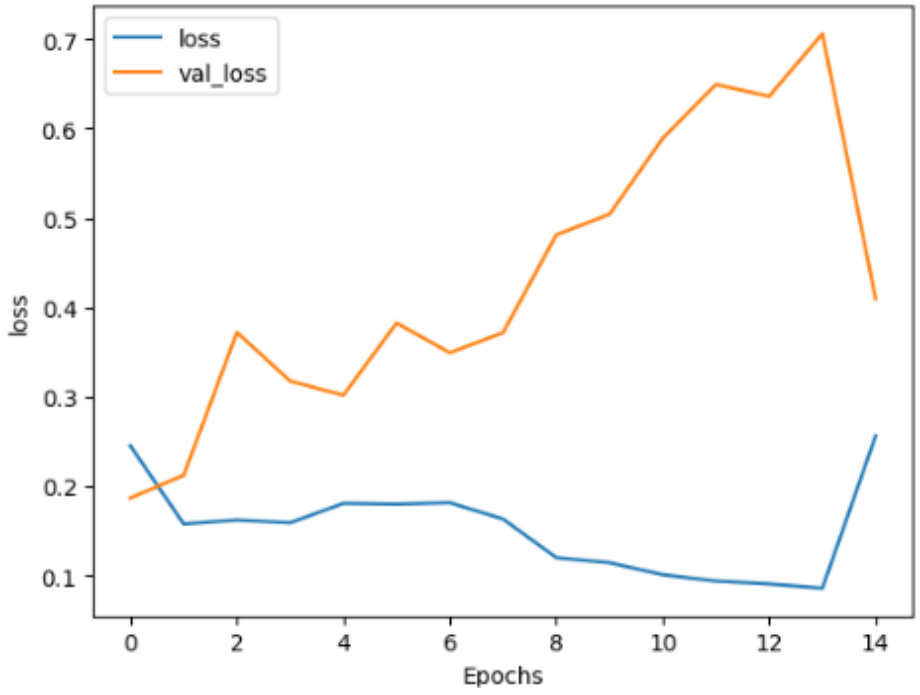
```
#from tensorflow.keras.layers import CuDNNLSTM

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=max_length))  #define embed layer
model.add(SpatialDropout1D(0.2))                                          #add dropout layer
model.add(Bidirectional(LSTM(256)))                                       #bi-LSTM of 256 units
model.add(Dense(32,activation='relu'))                                    #FN of 32 units
model.add(Dense(1))

print(model.summary())
```

Results:

```
[ ]
    threshold = 0.5
    pred_values = model.predict(x_test_seq)
    print(classification_report((pred_values>0).astype(int).flatten(),y_test))
```

```
938/938 [==============================] - 4s 4ms/step
              precision    recall  f1-score   support

           0       0.75      0.98      0.85     20726
           1       0.83      0.27      0.41      9274

    accuracy                           0.76     30000
   macro avg       0.79      0.62      0.63     30000
weighted avg       0.78      0.76      0.71     30000
```


accuracy
val_accuracy
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
accuracy
0
2
4
6
8
10
12
14
Epochs


loss
val_loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
loss
0
2
4
6
8
10
12
14
Epochs

3.3.3 BERT:

BERT is a Transformers based model that is trained on two tasks mainly Mask prediction and Next sentence Prediction. It was trained on billions of Data and can be fine tuned according to the required application.

Here we have imported a pre-trained BERT model and tokenizer from Hugging Face Transformers.

We have reduced the data from 1.3 M rows to 100K rows with class 0 and 1 in proportion 90% and 10% respectively. Due to limitations in Computing resources and memory.

Here we have tokenized the sentences with length 50 . It returns input_ids and attention_mask id.

```
X_tr = tokenizer(text=X_train.tolist(), add_special_tokens=True,
                 max_length=max_len, truncation=True,
                  padding='max_length', return_tensors='tf',
                  return_token_type_ids=False, return_attention_mask=True)

X_te = tokenizer(text=X_test.tolist(), add_special_tokens=True,
                 max_length=max_len, truncation=True,
                   padding='max_length', return_tensors='tf',
                 return_token_type_ids=False, return_attention_mask=True)
```

```
[9] X_tr      #shape=(70000, 50)
```

```
{'input_ids': <tf.Tensor: shape=(70000, 50), dtype=int32, numpy=
array([[  101,  1110,  1936, ...,     0,     0,     0],
       [  101,  1184,  1948, ...,     0,     0,     0],
       [  101,  1725, 15865, ...,     0,     0,     0],
       ...,
       [  101,  1150,  1207, ...,     0,     0,     0],
       [  101,  1184,  3719, ...,     0,     0,     0],
       [  101,  1110,  6782, ...,     0,     0,     0]], dtype=int32)>, 'attention_mask': <tf.Tensor: shape=(70000, 50), dtype=int32, numpy=
array([[1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       ...,
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0]], dtype=int32)>}
```

```
[ ] X_te      #shape=(30000, 50)
```

```
{'input_ids': <tf.Tensor: shape=(30000, 50), dtype=int32, numpy=
array([[ 101, 1293, 1510, ...,    0,    0,    0],
       [ 101,  178, 1185, ...,    0,    0,    0],
       [ 101, 1184, 3641, ...,    0,    0,    0],
       ...,
       [ 101, 1293, 1831, ...,    0,    0,    0],
       [ 101, 1184, 1341, ...,    0,    0,    0],
       [ 101, 1293,  178, ...,    0,    0,    0]], dtype=int32)>, 'attention_mask': <tf.Tensor: shape=(30000, 50), dtype=int32, numpy=
array([[1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       ...,
       [1, 1, 1, ..., 0, 0, 0],
```

```
[10] input_ids = Input(shape=(max_len,), dtype=tf.int32, name="input_ids") #create a layer of len 50 tokens
     input_mask = Input(shape=(max_len,), dtype=tf.int32, name="attention_mask")
```

```
embeddings = bert(input_ids, attention_mask = input_mask)[0] # 0 = last hidden state, 1 = poller_output
out = tf.keras.layers.GlobalMaxPool1D()(embeddings)         #captures importance features
out = Dense(128, activation='relu')(out)                    #FNN with 128 units and dropout of 0.1
out = tf.keras.layers.Dropout(0.1)(out)
out = Dense(32, activation='relu')(out)                     #FNN with 128 units

y = Dense(2, activation='softmax')(out)                     #results in two output

model = tf.keras.Model(inputs=[input_ids, input_mask], outputs=y)   #input_id and input_mask is passed as inputs
model.layers[2].trainable = True                                    #update weights of BertLayer

optimizer = Adam(learning_rate=5e-05, epsilon=1e-08, clipnorm=1.0 )

loss = CategoricalCrossentropy(from_logits=True)
metric = CategoricalAccuracy('balanced_accuracy')

model.compile(
    optimizer=optimizer,
    loss=loss,
    metrics=metric
)
```

```
[16] #epoch 5 and bacth_size=64
     history_bert = model.fit(x = {'input_ids':X_tr['input_ids'], 'attention_mask':X_tr['attention_mask']},y = to_categorical(y_train),
                              validation_data = ({'input_ids':X_te['input_ids'], 'attention_mask':X_te['attention_mask']},to_categorical(y_test)),
                              epochs=5,batch_size=64)
```

We have used just 5 epochs with a batch as 64 due to long training time for each step.

Bert achieved a precision , recall and F1-Score of 81% which was more than all previous models.

```
[17] from sklearn.metrics import classification_report

     pred_values = model.predict({'input_ids': X_te['input_ids'], 'attention_mask': X_te['attention_mask']})
     y_pred = np.argmax(pred_values, axis=1)
     print(classification_report(y_test, y_pred))
```

```
938/938 [==============================] - 129s 130ms/step
              precision    recall  f1-score   support

           0       0.96      0.96      0.96     27000
           1       0.65      0.68      0.66      3000

    accuracy                           0.93     30000
   macro avg       0.81      0.82      0.81     30000
weighted avg       0.93      0.93      0.93     30000
```

4. Results and Conclusion

4.1. Results

Model	Precision	Recall	Accuracy	F1_Score
Logistic Regression	0.66	0.37	0.94	**0.47**
Bi_LSTM	0.79	0.62	.76	**0.63**
BERT	0.81	0.82	0.93	**0.81**

Given the imbalance in our data, we have opted to use F1-Score as our primary metric. This is because accuracy alone can be misleading, as it does not provide information about how many instances of each class were correctly classified. By using BERT, we were able to significantly improve the F1-Score for correctly classifying class 1 (i.e., "Insincere") from 47% to 81%. This indicates that BERT performed better than other models in accurately identifying instances of class 1.

4.2. Conclusion

Our research using the Quora dataset revealed that reporting queries that are inappropriate for websites like Quora and StackOverflow might be quite valuable. We examined the distribution of genuine and phony inquiries, determined the most prevalent terms, and computed numerical values for token distribution, character distribution, and stop words. Then we built models with TF-IDF + Logistic Regression, Bi-LSTM, and BERT, with BERT outperforming the others.

5. Future Scope

To expand the range and efficiency of our model, we can integrate data from diverse fields, including healthcare (such as WebMD and MedHelp), technology (like Stack Overflow and Yahoo), and social media platforms like Twitter. By incorporating information from these sources, we can enrich the quality and quantity of our training data, thereby improving the performance and accuracy of our model.

In addition to incorporating new data sources, we can also optimize the hyperparameters of our final model by combining different word embeddings or using model ensembling techniques. This process involves tweaking the settings of our model to find the optimal configuration for achieving our desired results. By exploring different approaches, we can create a more robust and effective model.

Furthermore, we can also explore the implementation of cutting-edge techniques like GPT's and XLNET to improve the performance and accuracy of our system. These advanced algorithms have demonstrated remarkable success in natural language processing tasks and could potentially enhance the capabilities of our model.

Finally, our model can be applied in various domains beyond Quora, including social media analysis, customer feedback analysis, detecting hate speech, and spam detection of fraudulent activities. These applications can help businesses and organizations extract valuable insights from user-generated content and make data-driven decisions.

6. References

[1]https://towardsdatascience.com/fine-tuning-bert-for-text-classification-54e7df642894#6ba6

[2]https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/?utm_source=blog&utm_medium=comprehensive-guide-attention-mechanism-deep-learning

[3]https://towardsdatascience.com/multi-label-text-classification-using-bert-and-tensorflow-d2e88d8f488d

[4]https://arxiv.org/pdf/1810.04805.pdf

[5]https://pytorch.org/hub/huggingface_pytorch-transformers/

[6]https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270