

Assignment 3: Natural Language Processing

Nainil Rajendra Maladkar
NUID: 002780019 maladkar.n@northeastern.edu

Simran Nagpurkar
NUID: 002922747 nagpurkar.s@northeastern.edu

TOPIC: Text Classification : Quora Questions Classification

PROBLEM DEFINITION:

The **Quora Sincere/Insincere Questions Classification** is a Python NLP project that aims to segregate the given dataset of questions posted on Quora into relevant Sincere or Insincere buckets for optimized text classification.

The problem at hand is to develop a model that can effectively classify questions posted on the Quora platform as either "sincere" or "insincere." In the context of this classification task, the following definitions are relevant:

Sincere Questions : These are questions that are genuinely seeking helpful answers, contributing to the knowledge-sharing ethos of Quora. Sincere questions are founded on a desire for information, insights, and solutions.

Insincere Questions : Insincere questions are problematic as they are typically not genuine inquiries for information. They may be based on false premises, intended to make statements, or can be offensive, divisive, or inappropriate in nature. These questions often violate Quora's "Be Nice, Be Respectful" policy.

The training and testing data set is obtained from Quora Insincere Questions Classification

(<https://www.kaggle.com/c/quora-insincere-questions-classification>) Kaggle Dataset

DATA LOADING AND PREPARATION:

This project makes use of *train.csv* as **raw_train_df** and *test.csv* as **raw_test_df** files from the Kaggle competition dataset.

The **raw_train_df** contains 3 columns :

'qid' for the id of each question

'question_text' for the content of the input question

'target' =0 for sincere question and 'target'= 1 for insincere questions

The **raw_train_df** contains 3 columns :

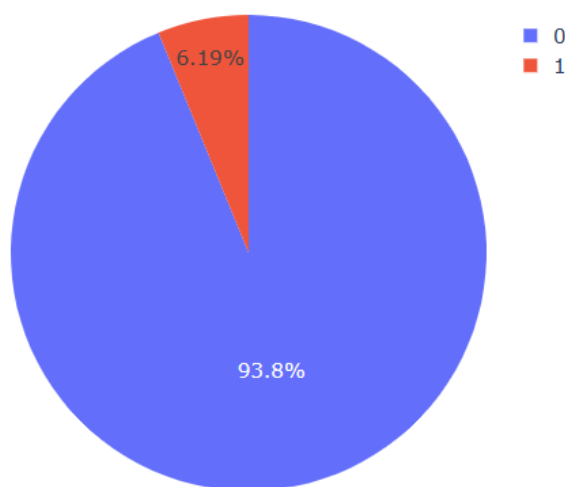
'qid' for the id of each question

'question_text' for the content of the input question

An insincere question is defined as a question intended to make a statement rather than look for helpful answers.

Some characteristics that can signify that a question is insincere:

- Has a non-neutral tone
- Has an exaggerated tone to underscore a point about a group of people
- Is rhetorical and meant to imply a statement about a group of people
- Is disparaging or inflammatory
- Suggests an idea against a protected class of people, or seeks confirmation of a stereotype
- Makes disparaging attacks/insults against a specific person or group of people
- Based on an outlandish premise about a group of people
- Disparages against a characteristic that is not fixable and not measurable
- Based on false information, or contains absurd assumptions
- Uses sexual content (incest, bestiality, pedophilia) and not to seek genuine answers



So about 6% of the training data are insincere questions (target=1) and rest of them are sincere.

Get TRAIN and TEST dataset

```
#Get Dataset
raw_train_df = pd.read_csv('train.csv',engine='python')
raw_test_df = pd.read_csv('test.csv',engine='python')
#sample_df = pd.read_csv('sample_submission.csv',engine='python')
```

```
raw_train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1306122 entries, 0 to 1306121
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  -
0   qid          1306122 non-null object
1   question_text 1306122 non-null object
2   target       1306122 non-null int64
dtypes: int64(1), object(2)
memory usage: 29.9+ MB
```

```
: raw_test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 375806 entries, 0 to 375805
Data columns (total 2 columns):
#   Column      Non-Null Count  Dtype
---  -
0   qid          375806 non-null object
1   question_text 375806 non-null object
dtypes: object(2)
memory usage: 5.7+ MB
```

We have implemented 'Bag of Words' for stemming, removal of stop words as text preprocessing and vectorization using TF-IDF

Further baseline models like Linear regression, Naïve Bayes and advanced model like CNN is implemented to evaluate the success of the model using appropriate machine learning metrics, such as accuracy, precision, recall, F1 score.

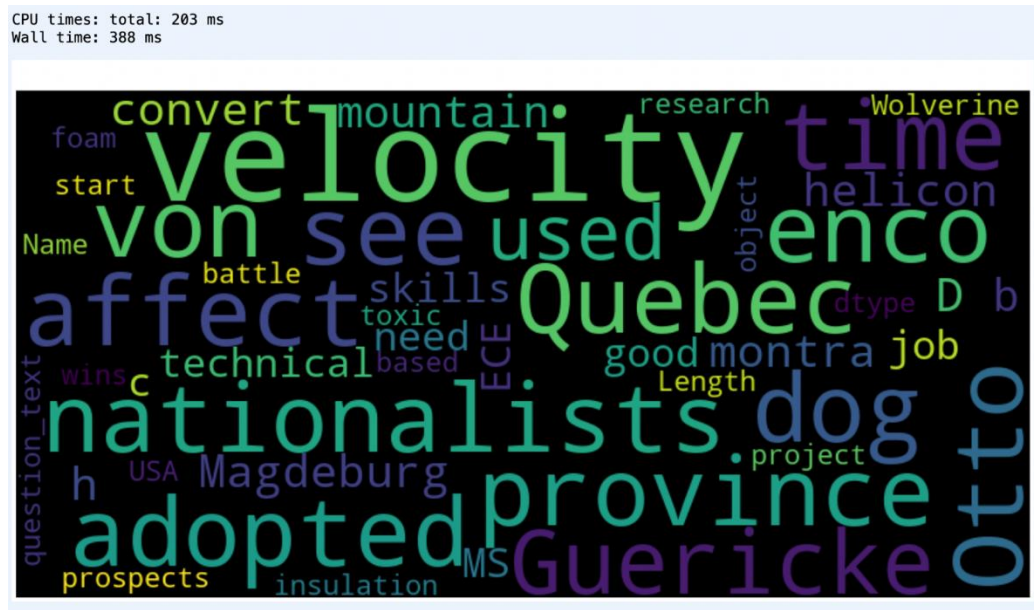
We have used utilizes libraries Pandas, Numpy, Matplotlib, Seaborn, Sklearn, NLTK, Torch, Transformers, Keras to perform exploratory data analysis and gather insights for machine learning.

EXPLORATORY DATA ANALYSIS:

1. Word Cloud Visualization:

The code starts by generating a word cloud for the "*question_text*" column in the dataset.

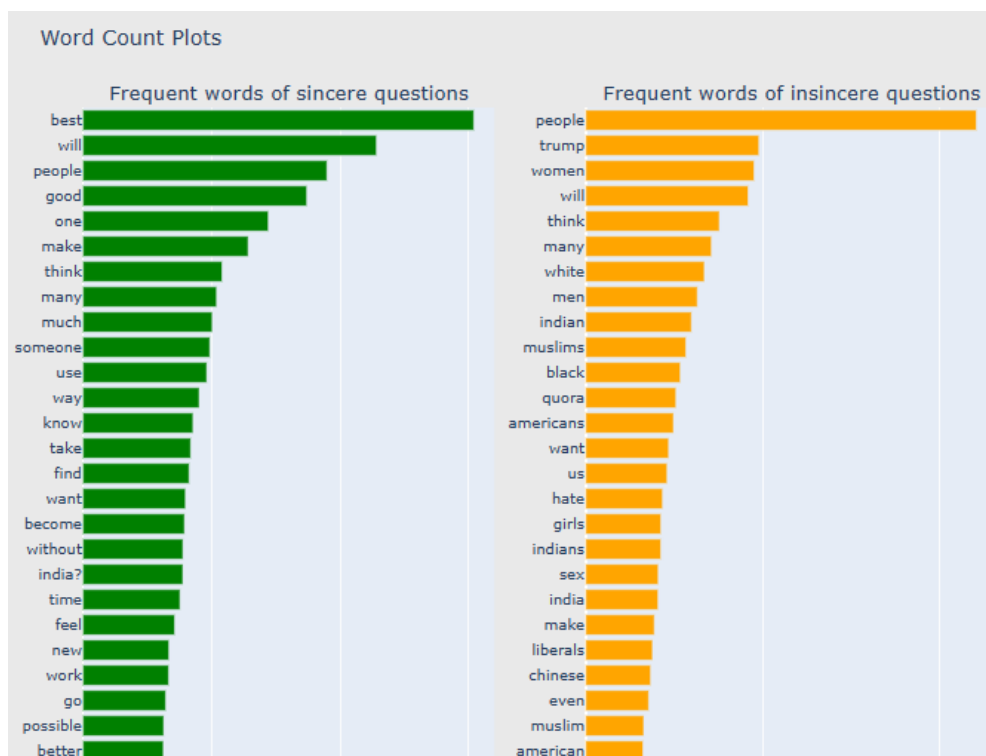
A word cloud is a visual representation of the most frequently occurring words in the text data.



2. Word Frequency Analysis:

The code then performs word frequency analysis for both sincere and insincere questions, focusing on the most frequently used words in each class.

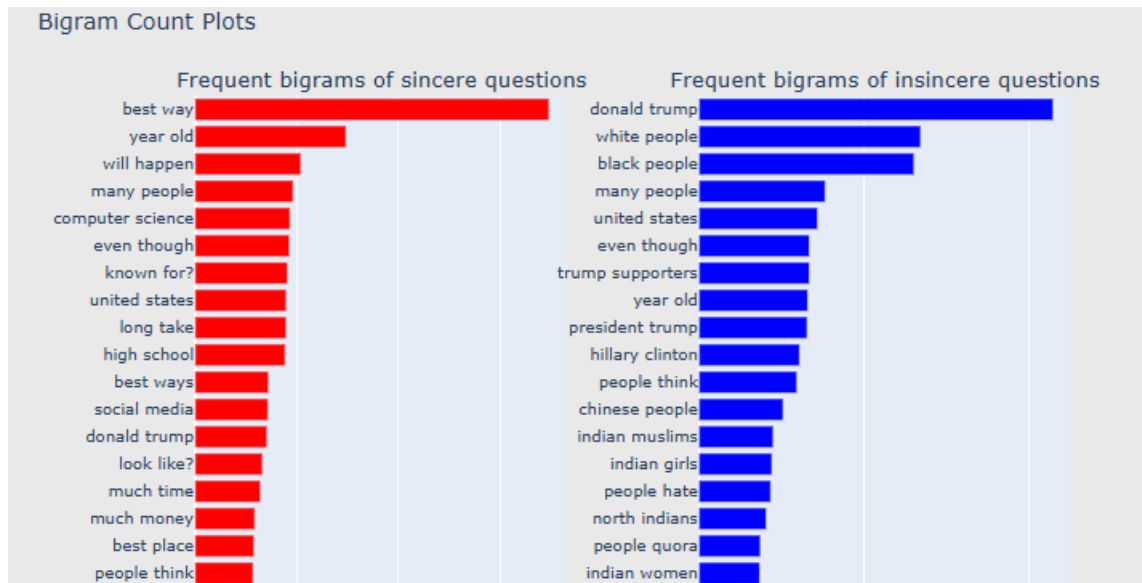
- For **Sincere** Questions: It identifies the most common words and visualizes their frequencies. The top words include terms like "people," "will," and "think."
- For **Insincere** Questions: It does the same analysis for insincere questions, highlighting the most frequent words specific to this class, such as "trump," "women," and "white."



3. Bigram Frequency Analysis:

Bigram (two-word combinations) frequencies for both sincere and insincere questions.

- For **Sincere** Questions: It identifies the most common bigrams in sincere questions.
- For **Insincere** Questions: It does the same for insincere questions.



Observations:

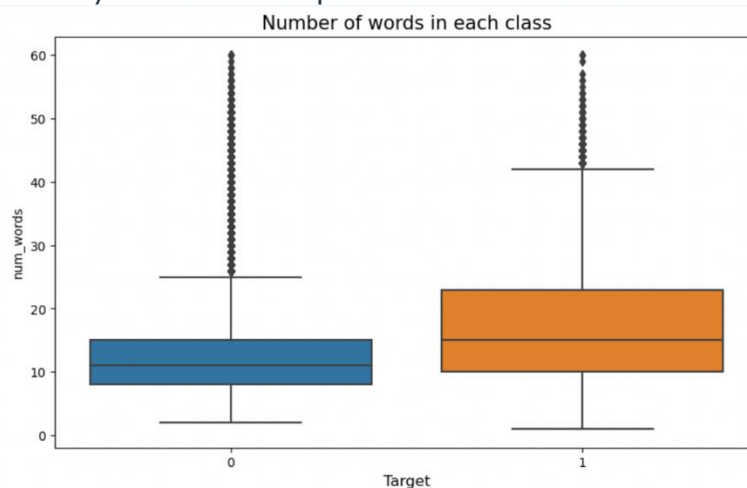
- The other top words in sincere questions after excluding the common ones at the very top are 'best', 'year', 'people' etc
- The other top words in insincere questions after excluding the common ones are 'trump', 'women', 'white' and other political words etc

FEATURE EXTRACTION:

In text classification, it's essential to understand the characteristics of the text data. The code below analyzes various aspects of the text to gain insights into how they may relate to the classification task.

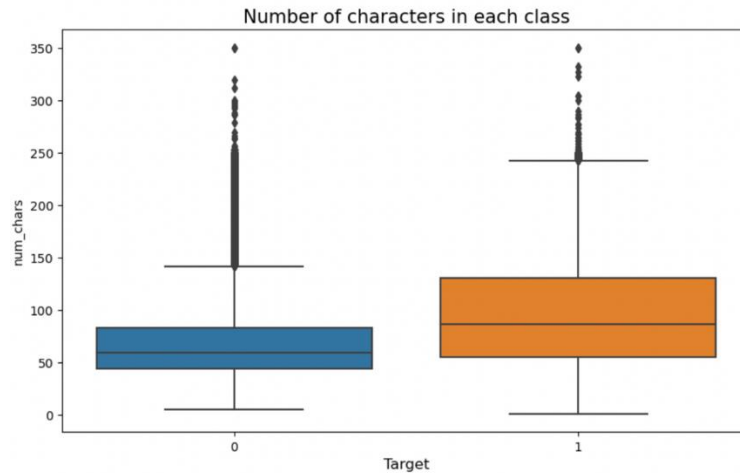
- Word Count and Unique Words Count:

num_words and num_unique_words represent the number of words and unique words in each text. Understanding the vocabulary size words in questions is crucial for feature engineering.



- **Character Count:**

`num_chars` counts the number of characters in each text, which is important for understanding the length of questions. Examining character count helps identify potential variations in question.

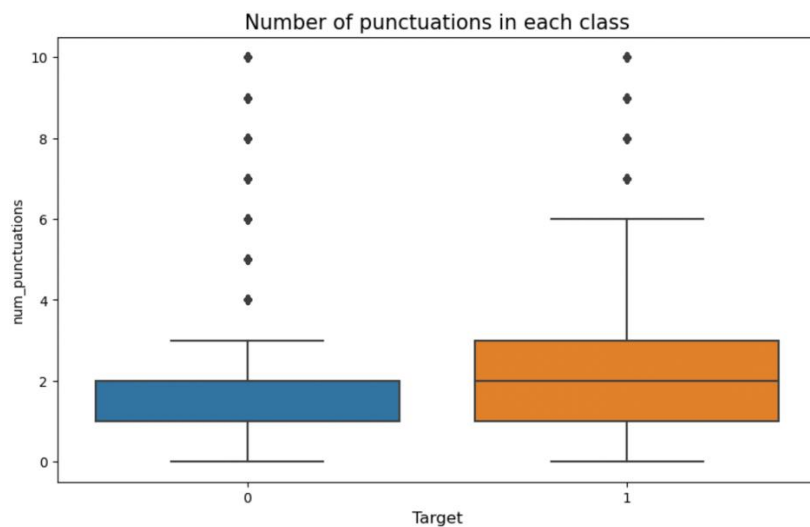


- **Stopwords Count:**

`num_stopwords` quantifies the number of stopwords (common words like "the," "is," "and," etc.) in each text. Identifying stopwords can common language the questions contain.

- **Punctuation Count:**

`num_punctuations` measures the number of punctuation marks in each text. Analyzing punctuation helps understand the usage of symbols and their potential impact on classification.



- **Title and Uppercase Words Count:**

`num_words_upper` and `num_words_title` calculate the number of words in uppercase and title case, respectively. Detecting uppercase and title case words can provide insights into question style.

- **Average Word Length:**

`mean_word_len` computes the average length of words in each text. Understanding the average word length may be relevant for differentiating question types.

- **Number of Words in Each Class:**

The first box plot shows how the number of words varies in questions of different classes. It helps assess whether the length of the questions is indicative of their sincerity.

1. Remove Numbers

Example: Which is best powerbank for iPhone 7 in India? -> Which is best powerbank for iPhone in India?

`removeNumbers(text)` :

This is a function that takes a string (text) as input and removes all the numeric digits from it using a list comprehension. It joins all the non-digit characters back together and returns the modified text.

```
def removeNumbers(text):
    text = ''.join([i for i in text if not i.isdigit()])
    return text

text_removeNumbers = pd.DataFrame(columns=['TextBefore', 'TextAfter', 'Changed'])
text_removeNumbers['TextBefore'] = text.copy()

%%time
for index, row in text_removeNumbers.iterrows():
    row['TextAfter'] = removeNumbers(row['TextBefore'])

text_removeNumbers['Changed'] = np.where(text_removeNumbers['TextBefore']!=text_removeNumbers['TextAfter'], 'no', 'yes')
print("{} of {} {:.4f}% questions have been changed.".format(len(text_removeNumbers[text_removeNumbers['Changed']=='yes']), len
1306122 of 1306122 (100.0000%) questions have been changed.
```

2. Replace Repetitions of Punctuation

This technique:

1. replaces repetitions of exclamation marks with the tag "multiExclamation"
2. replaces repetitions of question marks with the tag "multiQuestion"
3. replaces repetitions of stop marks with the tag "multiStop"

Example: How do I overcome the fear of facing an interview? It's killing me inside..what should I do? -> How do I overcome the fear of facing an interview? It's killing me inside multiStop what should I do?

```
def replaceMultiExclamationMark(text):
    text = re.sub(r"(!)\1+", ' multiExclamation ', text)
    return text

def replaceMultiQuestionMark(text):
    text = re.sub(r"(?)\1+", ' multiQuestion ', text)
    return text

def replaceMultiStopMark(text):
    text = re.sub(r"(\.)\1+", ' multiStop ', text)
    return text

text_replaceRepOfPunct = pd.DataFrame(columns=['TextBefore', 'TextAfter', 'Changed'])
text_replaceRepOfPunct['TextBefore'] = text.copy()

for index, row in text_replaceRepOfPunct[text_replaceRepOfPunct['Changed']=='yes'].head().iterrows():
    print(row['TextBefore'], '->', row['TextAfter'])

How did Quebec nationalists see their province as a nation in the 1960s? -> nan
Do you have an adopted dog, how would you encourage people to adopt and not shop? -> nan
Why does velocity affect time? Does velocity affect space geometry? -> nan
How did Otto von Guericke used the Magdeburg hemispheres? -> nan
Can I convert montra helicon D to a mountain bike by just changing the tyres? -> nan
```

3. Remove Punctuation

Create an empty DataFrame called `text_removePunctuation` with three columns: 'TextBefore,' 'TextAfter,' and 'Changed.'

This DataFrame will be used to store the original text, the modified text after removing punctuation, and a flag indicating whether any changes were made.

Populate the 'TextBefore' column of `text_removePunctuation` with the values from a variable named text, assuming that text is a list or array of text data.

Example: Why haven't two democracies never ever went for a full fledged war? What stops them? -> Why havent two democracies never ever went for a full fledged war What stops them

```
import string
translator = str.maketrans('', '', string.punctuation)
text_removePunctuation = pd.DataFrame(columns=['TextBefore', 'TextAfter', 'Changed'])
text_removePunctuation['TextBefore'] = text.copy()

for index, row in text_removePunctuation.iterrows():
    row['TextAfter'] = row['TextBefore'].translate(translator)

n(text_removePunctuation), 100*len(text_removePunctuation[text_removePunctuation['Changed']=='yes']/len(text_removePunctuation))

1306122 of 1306122 (100.0000%) questions have been changed.

for index, row in text_removePunctuation[text_removePunctuation['Changed']=='yes'].head().iterrows():
    print(row['TextBefore'], '->', row['TextAfter'])

How did Quebec nationalists see their province as a nation in the 1960s? -> nan
Do you have an adopted dog, how would you encourage people to adopt and not shop? -> nan
Why does velocity affect time? Does velocity affect space geometry? -> nan
How did Otto von Guericke used the Magdeburg hemispheres? -> nan
Can I convert montra helicon D to a mountain bike by just changing the tyres? -> nan
```


4. Replace Contractions

This technique replaces contractions to their equivalents.

`contraction_patterns` : This is a list of tuples, where each tuple consists of two elements. The first element in each tuple is a regular expression pattern that matches a contraction, and the second element is the replacement for that contraction. For example, (r'won't', 'will not') matches the contraction "won't" and replaces it with "will not".

`replaceContraction(text)` : This is a function that takes a text input and expands contractions within that text.

Inside the function, it creates a list of patterns and replacements using the `contraction_patterns`. It does this by iterating through the `contraction_patterns` list and compiling each regular expression pattern with `re.compile()`. The compiled pattern and its replacement are stored as a tuple in the `patterns` list.

Example: What's the scariest thing that ever happened to anyone? -> What is the scariest thing that ever happened to anyone?

```
contraction_patterns = [ (r'won\'t', 'will not'), (r'can\'t', 'cannot'), (r'i\'m', 'i am'), (r'ain\'t', 'is not'), (r'(\w+)\\'ll',
                        (r'(\w+)\\'ve', '\g<1> have'), (r'(\w+)\\'s', '\g<1> is'), (r'(\w+)\\'re', '\g<1> are'), (r'(\w+)\\'d', '\g<
def replaceContraction(text):
    patterns = [(re.compile(regex), repl) for (regex, repl) in contraction_patterns]
    for (pattern, repl) in patterns:
        (text, count) = re.subn(pattern, repl, text)
    return text

text_replaceContractions = pd.DataFrame(columns=['TextBefore', 'TextAfter', 'Changed'])
text_replaceContractions['TextBefore'] = text.copy()
```

```
for index, row in text_replaceContractions.iterrows():
    row['TextAfter'] = replaceContraction(row['TextBefore'])
```

```
text_replaceContractions['Changed'] = np.where(text_replaceContractions['TextBefore']!=text_replaceContractions['TextAfter'],
print("{} of {} ({:.4f}%) questions have been changed.".format(len(text_replaceContractions[text_replaceContractions['TextAfter']!=text_replaceContractions['TextBefore']],
1306122 of 1306122 (100.0000%) questions have been changed.
```

```
for index, row in text_replaceContractions[text_replaceContractions['Changed']=='yes'].head().iterrows():
    print(row['TextBefore'], '->', row['TextAfter'])
```

How did Quebec nationalists see their province as a nation in the 1960s? -> nan
Do you have an adopted dog, how would you encourage people to adopt and not shop? -> nan
Why does velocity affect time? Does velocity affect space geometry? -> nan
How did Otto von Guericke used the Magdeburg hemispheres? -> nan
Can I convert montra helicon D to a mountain bike by just changing the tyres? -> nan

TAKING A SAMPLE OF 100K DATA:

It randomly selects 100,000 rows from the `raw_train_df` training dataset, creating a smaller sample called `sample_df` for efficient analysis or modelling. The `random_state` parameter ensures reproducibility of the random selection.

Create a working sample

```
SAMPLE_SIZE = 100_000
sample_df = raw_train_df.sample(SAMPLE_SIZE, random_state=42)
```

`sample_df`

	qid	question_text	target
443046	56d324bb1e2c29f43b12	What is the most effective classroom managemen...	0
947549	b9ad893dc78c577f8a63	Can I study abroad after 10th class from Bangl...	0
523769	6689ebaeeb65b209a412	How can I make friends as a college junior?	0
949821	ba1e2c4a0fef09671516	How do I download free APK Minecraft: Pocket E...	0
1030397	c9ea2b69bf0d74626f46	Like Kuvera, is "Groww" also a free online inv...	0
...
998930	c3c03a307a29c69971b4	How do I research list of reliable charcoal im...	0
66641	0d119aba95ee6684f506	What are petroleum products, and what is petro...	0
90024	11a46cd148a104b271cf	What are some services that will let you quick...	0
130113	1973e6e2111a0c93193a	What credit card processors do online marketpl...	0
1137	0037ed037520d82393c0	On which number system does a computer work?	0

100000 rows x 3 columns

TEXT PREPROCESSING:

Outline:

1. Understand the bag of words model
2. Tokenization
3. Stop word removal
4. Stemming

Bag of Words Intuition

- Create a list of all the words across all the text documents.
- Convert each document into a vector of word counts.

Limitations:

- The dataset may contain too many words.
- Some words may occur too frequently.
- Some words may occur very rarely or only once.
- A single word may have many forms (e.g., go, gone, going, or bird vs. birds).

TOKENIZATION:

- The code processes two sample questions, represented by the variables `q0` and `q1`. These questions are part of a larger dataset.
- The `word_tokenize` function from the `nlTK` library is applied to each question. This function splits each question into individual words or tokens. For instance, the question "How are you today?" is tokenized into a list of words: ["How", "are", "you", "today", "?"].
- The code stores the tokenized versions of the questions in variables `q0_tok` and `q1_tok`. These variables now hold lists of words, which are more manageable for further NLP analysis.

```
q1
'Has the United States become the largest dictatorship in the world?'
```

```
word_tokenize(q1)
```

```
['Has',
 'the',
 'United',
 'States',
 'become',
 'the',
 'largest',
 'dictatorship',
 'in',
 'the',
 'world',
 '?']
```

```
q0_tok = word_tokenize(q0)
q1_tok = word_tokenize(q1)
```

```
q1_tok
```

```
['Has',
 'the',
 'United',
 'States',
 'become',
 'the',
 'largest',
 'dictatorship',
 'in',
 'the',
 'world',
 '?']
```


STOPWORD REMOVAL FUNCTION:

The code defines a function called `remove_stopwords(tokens)`. This function takes a list of tokens (words or phrases) as input.

- Tokenization:
Two example text strings, **q0** and **q1**, are tokenized into lists of words using a tokenization process. Tokenization splits the text into individual words or tokens.
- Stopword Removal:
The `remove_stopwords` function is applied to the tokenized lists, **q0_tok** and **q1_tok**. This function iterates through the tokens and removes any words that match the English stopwords list. It retains only the words that are not in the stopwords list.
- Filtered Text:
The code stores the results in variables **q0_stp** and **q1_stp**. These variables now contain the text with stopwords removed. For example, if "are" and "you" were stopwords, the list ["How", "are", "you", "today", "?"] would become ["How", "today", "?"] after stopwords removal.

```
# print stopwords
", ".join(english_stopwords)

'i, me, my, myself, we, our, ours, ourselves, you, you're, you've, you'll, you'd, your, yours, yourself, yourselfe
s, he, him, his, himself, she, she's, her, hers, herself, it, it's, its, itself, they, them, their, theirs, themsel
ves, what, which, who, whom, this, that, that'll, these, those, am, is, are, was, were, be, been, being, have, has,
had, having, do, does, did, doing, a, an, the, and, but, if, or, because, as, until, while, of, at, by, for, with,
about, against, between, into, through, during, before, after, above, below, to, from, up, down, in, out, on, off,
over, under, again, further, then, once, here, there, when, where, why, how, all, any, both, each, few, more, most,
other, some, such, no, nor, not, only, own, same, so, than, too, very, s, t, can, will, just, don, don't, should, s
hould've, now, d, ll, m, o, re, ve, y, ain, aren, aren't, couldn, couldn't, didn, didn't, doesn, doesn't, hadn, had
n't, hasn, hasn't, haven, haven't, isn, isn't, ma, mightn, mightn't, mustn, mustn't, needn, needn't, shan, shan't,
shouldn, shouldn't, wasn, wasn't, weren, weren't, won, won't, wouldn, wouldn't"

def remove_stopwords(tokens):
    return [word for word in tokens if word.lower() not in english_stopwords]

q0_tok

['Do',
'you',
'have',
'an',
'adopted',
'dog',
',',
',',
'how',
'would',
'you',
'encourage',
'people',
'to',
'adopt',
'and',
'not',
'shop',
'?']
```

STEMMING

Stemming is a common text preprocessing step in natural language processing (NLP) projects. Its primary goal is to reduce words to their root or base form. For example, it can transform variations of a word, such as "go," "gone," and "going," into the common root form, "go."

The code then applies stemming to two lists of words, **q0_stp** and **q1_stp**. These lists contain text that has already undergone stop word removal.

The stemming process iterates through each word in the lists and reduces them to their root forms, effectively simplifying the words.

```

from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer(language='english')

stemmer.stem('going')
'go'

stemmer.stem('supposedly')
'suppos'

q0_stm = [stemmer.stem(word) for word in q0_stp]

q0_stp
['adopted', 'dog', ',', 'would', 'encourage', 'people', 'adopt', 'shop', '?']

q0_stm
['adopt', 'dog', ',', 'would', 'encourag', 'peopl', 'adopt', 'shop', '?']

q1_stm = [stemmer.stem(word) for word in q1_stp]

q1_stp
['United', 'States', 'become', 'largest', 'dictatorship', 'world', '?']

q1_stm
['unit', 'state', 'becom', 'largest', 'dictatorship', 'world', '?']

```

Resulting Stemmed Text:

The resulting stemmed words are stored in variables `q0_stm` and `q1_stm`. These variables now contain the text with words reduced to their root forms, making it easier to analyse text data.

IMPLEMENT BAG OF WORDS

The Bag of Words (BoW) model is a popular technique in natural language processing (NLP) for converting text data into a numerical format that can be used for machine learning tasks. This code segment explains how to create a BoW model using the Count Vectorizer.

To create a BoW representation, we follow these steps:

- Vocabulary Creation: Using the `CountVectorizer` from the `scikit-learn` library, we first establish a vocabulary. This vocabulary is a set of all unique words found in the dataset, excluding common English stopwords that add little meaning to the analysis.
- Text Transformation: Each text entry is transformed into a numerical format. This is where the `CountVectorizer` comes into play, converting the text into a vector where each index corresponds to a word from our vocabulary, and the value is the frequency of that word in the text.
- Text Preprocessing: Before transforming the text, we can configure the `CountVectorizer` to include text preprocessing to standardize the data. This might involve lowercasing the text, removing punctuation, and other similar tasks to ensure consistency.

```

from sklearn.feature_extraction.text import CountVectorizer

small_vect = CountVectorizer()

small_vect.fit(small_df.question_text)

▼ CountVectorizer
CountVectorizer()

small_vect.vocabulary_

{'what': 49,
 'is': 31,
 'the': 47,
 'most': 39,
 'effective': 16,
 'classroom': 9,
 'management': 37,
 'skill': 44,
 'technique': 46,

```

The vectorizer is used to transform the text documents in `sample_df` into numerical vectors. The resulting vectors represent the word counts for each document. The code showcases the transformation for the first document. The code then applies this configured vectorizer to both the sample data (`sample_df`) and the test data (`raw_test_df`).

Transform documents into Vectors

```
In [80]: vectors = small_vect.transform(small_df.question_text)
```

```
In [81]: vectors
```

```
Out[81]: <5x51 sparse matrix of type '<class 'numpy.int64'>'
         with 56 stored elements in Compressed Sparse Row format>
```

```
In [82]: vectors.shape
```

```
Out[82]: (5, 51)
```

```
small_df.question_text.values[0]
```

```
'What is the most effective classroom management skill/technique to create a good learning environment?'
```

```
vectors[0].toarray()
```

```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
        0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
        1, 0, 1, 1, 1, 1, 0]], dtype=int64)
```

```
vectors.toarray()
```

```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
        0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
        1, 0, 1, 1, 1, 1, 0],
       [1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
        0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0,
        0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
        0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0,
        0, 0, 0, 0, 0, 1]], dtype=int64)
```

Configure Count Vectorizer Parameters

```
stemmer = SnowballStemmer(language='english')
```

```
def tokenize(text):
    return [stemmer.stem(word) for word in word_tokenize(text)]
```

```
tokenize('What is the really (dealing) here?')
```

```
['what', 'is', 'the', 'realli', '(', 'deal', ')', 'here', '?']
```

```
vectorizer = CountVectorizer(lowercase=True,
                             tokenizer=tokenize,
                             stop_words=english_stopwords,
                             max_features=1000)
```

The result is a set of inputs (vectors) for machine learning models.

Implementing a BoW with `CountVectorizer` is a crucial step in text analysis. This technique helps in transforming textual data into a numerical format that machine learning models can easily interpret, thereby facilitating a variety of natural language processing tasks.

TFIDF TOKENIZATION

TF-IDF, which stands for Term Frequency-Inverse Document Frequency. This method refines the process of turning text into numbers, enabling computers to understand and process language with a subtlety that goes beyond mere word counting.

The Custom Tokenizer: The function `tokenize_and_stem(text)` is crafted to tokenize and stem the given text. It combines text preprocessing steps like converting to lowercase, removing stopwords (common words that are typically excluded in text analysis), and stemming.

Vectorization: With `TfidfVectorizer`, we configure it to use our custom tokenizer function. Then, it is fit on our data, represented here as `raw_train_df['question_text']`, turning the text into a matrix.

Transformation: Finally, `model_inputs` is created by transforming our raw text data into the TF-IDF encoded format, ready for use in machine learning models.

```
%time

import nltk
from nltk.stem import SnowballStemmer
stemmer = SnowballStemmer(language="english")

en_stopwords = nltk.corpus.stopwords.words('english')

from sklearn.feature_extraction.text import TfidfVectorizer
def tokenize_and_stem(text):
    return [stemmer.stem(token) for token in word_tokenize(text)]
# creating count vectorizer object with configuration to incorporate preprocessing techniques together
vectorizer = TfidfVectorizer(lowercase=True,
                             stop_words=en_stopwords, #removing stopwords
                             tokenizer=tokenize_and_stem #tokenization and stemming at the same time
                             )

vectorizer.fit(raw_train_df['question_text'])
```

```
len(vectorizer.vocabulary_)
vectorizer.get_feature_names_out()[100]
```

```
array(['\x02t\x7f\xe\x1aaùø\x8d¶rwiliñö', '\x10i', '\x10new', '\x10æø',
      '\x17', '!', '#', '$', '%', '&', "'", '"', '+', ',', '-', '.', ':', ';', '\\',
      '\n', '\r', 'd', 'm', 'n', 'q', 's', 't', '~', '_-', '...', '(', ')', '*', '+',
      '++', '+++', '++\\sqrt', '++a/a', '++compil', '++i', '++j', '++p',
      '+-', '+-100', '+60', '+-ln', '+.1mg', '+/-', '+-/1', '+/-15v',
      '+-2', '+720', '+0', '+0.00206x', '+0.12929=0', '+0.50', '+0o',
      '+1', '+1,2,3,4', '+1.00', '+1.0pccharg', '+1.25', '+1.25d',
      '+1.5', '+1.50', '+1/', '+1/100', '+1/2', '+1/3+1/4+', '+1/6',
      '+1/e^x-1', '+1/q+1/r=', '+1/x', '+10', '+100', '+1000', '+100k',
      '+10=', '+10^', '+10k', '+11', '+11x-6=0', '+12', '+125mhz',
      '+12'', '+13', '+13x^2', '+1400', '+14x3-2x2+7x-8', '+15', '+150',
      '+160', '+16=0', '+175', '+18', '+1=0', '+1\\', '+1k', '+2',
      '+2.0', '+2/3', '+20', '+2000', '+2002', '+21', '+212', '+23'],
      dtype=object)
```

```
model_inputs.shape
model_inputs
raw_train_df['question_text'].values[0]
model_inputs[0].toarray()
array([[0., 0., 0., ..., 0., 0., 0.]])
```

Advantages of TF-IDF Over Count Vectorizer

While the Bag of Words model counts how often each word occurs, it treats every word as equally significant. TF-IDF is a step up from the basic Count Vectorizer by bringing in the element of a word's relevance, not just its occurrence. TF-IDF provides a more nuanced approach. It reduces the weight of terms that occur very frequently in the dataset (thus are less informative) and increases the weight of terms that occur rarely, which often adds more significance.

BASELINE MODEL IMPLEMENTATION:

1. LOGISTIC REGRESSION

Logistic Regression is a classification algorithm, not a regression algorithm. It's employed in binary outcomes - where there are only two possible classes. In the case of the Quora dataset, the two classes are 'sincere' and 'insincere'.

When dealing with text, inputs are typically converted into vectorized formats, such as TF-IDF or Count Vectorizer outputs, where each dimension represents a specific word or n-gram. Despite the high dimensionality, Logistic Regression can efficiently handle such sparse matrices, making it suitable for text classification tasks.

- Data Splitting: The journey begins with splitting the data into training and validation sets using `train_test_split` from `sklearn.model_selection`. This segregation is vital as it allows us to train the model on a subset of the data (70% in this case) and validate its performance on the remaining 30%.
- Model Selection: `LogisticRegression` from `sklearn.linear_model` is the chosen model. It is capable of handling high-dimensional datasets, text data transformed into numerical vectors.
- Configuring the Model: We specify `MAX_ITER` as 1000 and the solver as 'sag', standing for Stochastic Average Gradient descent, an optimization method well-suited for large datasets.

```
from sklearn.model_selection import train_test_split

train_inputs, val_inputs, train_targets, val_targets = train_test_split(inputs, sample_df.target,
                                                                    test_size=0.3, random_state=42)

train_inputs.shape
(70000, 1000)

train_targets.shape
(70000,)

val_inputs.shape
(30000, 1000)

val_targets.shape
(30000,)
```

```
train_preds
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)

pd.Series(train_preds).value_counts()
0    67957
1     2843
Name: count, dtype: int64
```

Training the Model: With `model.fit(train_inputs, train_targets)`, the Logistic Regression model learns to associate the input features with the target classes.

Predictions: Using `model.predict(train_inputs)`, we obtain predictions for the training set. These predictions can be compared with `train_targets` to evaluate how well the model is performing.

ERROR CALCULATION METRICS

Accuracy: Accuracy is the most intuitive performance measure. It is the ratio of correctly predicted observations to the total observations. It's the metric that tells us, overall, how often the model is correct.

Precision: Precision reveals the proportion of positive identifications that were actually correct. It is a crucial metric when the costs of False Positives are high.

Recall: Also known as Sensitivity or True Positive Rate, Recall calculates how many of the actual positives our model capture through labeling it as positive. High Recall is vital when the cost of a False Negative is severe.

F1 Score: The F1 Score is the harmonic mean of Precision and Recall, and it is useful when you need to strike a balance between Precision and Recall.

Interpreting the Model's Performance

```
from sklearn.metrics import precision_score, recall_score

# Calculate predictions on the validation set
val_preds = model.predict(val_inputs)

# Calculate accuracy
accuracy = accuracy_score(val_targets, val_preds)
print(f'Accuracy: {accuracy:.2f}')

# Calculate precision
precision = precision_score(val_targets, val_preds)
print(f'Precision: {precision:.2f}')

# Calculate recall
recall = recall_score(val_targets, val_preds)
print(f'Recall: {recall:.2f}')

# Calculate F1 score
f1 = f1_score(val_targets, val_preds)
print(f'F1 Score: {f1:.2f}')

Accuracy: 0.95
Precision: 0.63
Recall: 0.30
F1 Score: 0.41
```

With an Accuracy of 0.95, the model shows high-level correctness across all predictions.

However, Precision stands at 0.63, indicating that when the model predicts an insincere question, it is correct around 63% of the time.

Recall hits at 0.30, suggesting that the model identifies 30% of all actual insincere questions.

Finally, the F1 Score of 0.41 indicates there is room for improvement, particularly in achieving a balance between Precision and Recall, which is vital for this dataset where both false positives and false negatives carry significant implications.

2. NAIVE BAYES

Naive Bayes classifiers operate on the principle of feature independence; they assume that the presence of a particular feature in a dataset is unrelated to the presence of any other feature. For text classification, this could translate to understanding that the occurrence of a certain word in a text does not influence the occurrence of another.

- Model Initialization: `model_nb = MultinomialNB()`

A Multinomial Naive Bayes model is instantiated, suitable for classification with discrete features (e.g., word counts for text classification).

- Model Training: `model_nb.fit(train_inputs, train_targets)`

The model is trained using the training data, which consists of the inputs (`train_inputs`) and their corresponding targets (`train_targets`).

- Prediction: `val_preds = model_nb.predict(val_inputs)`

Predictions are made on the validation set (`val_inputs`), where the model has not seen the data during training. This helps in evaluating the model's generalization capabilities

```
from sklearn.naive_bayes import MultinomialNB

# Train Naive Bayes model
model_nb = MultinomialNB()
model_nb.fit(train_inputs, train_targets)

# Predict on the validation set
val_preds_nb = model_nb.predict(val_inputs)
```

Calculate Precision , Recall , F1 score , Accuracy

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
# Calculate and print accuracy
accuracy_nb = accuracy_score(val_targets, val_preds_nb)
print(f'Accuracy (Naive Bayes): {accuracy_nb:.2f}')

# Calculate and print precision
precision_nb = precision_score(val_targets, val_preds_nb)
print(f'Precision (Naive Bayes): {precision_nb:.2f}')

# Calculate and print recall
recall_nb = recall_score(val_targets, val_preds_nb)
print(f'Recall (Naive Bayes): {recall_nb:.2f}')

# Calculate and print F1 score
f1_nb = f1_score(val_targets, val_preds_nb)
print(f'F1 Score (Naive Bayes): {f1_nb:.2f}')
```

```
Accuracy (Naive Bayes): 0.94
Precision (Naive Bayes): 0.48
Recall (Naive Bayes): 0.52
F1 Score (Naive Bayes): 0.50
```

ERROR CALCULATION METRICS

Naive Bayes showed **94% accuracy**, slightly lower than Logistic Regression's 95%, indicating a high rate of correct predictions across the dataset.

The Naive Bayes model has a **precision of 48%**, suggesting that when it predicts a question as insincere, the prediction is correct about half the time.

At 52%, the Naive Bayes model captures more than half of the actual insincere questions but still misses a significant portion.

An F1 score of 0.50 for Naive Bayes suggests a middle ground between precision and recall, reflecting a balanced trade-off, although not as strong as Logistic Regression in this case.

The Naive Bayes model exhibits reasonable effectiveness in classification with a balanced approach to false positives and false negatives. Its moderate precision and recall indicate that while it can reliably identify many insincere questions, there is a notable chance of misclassification.

Metric	Logistic Regression	Naive Bayes
Accuracy	0.95	0.94
Precision	0.63	0.48
Recall	0.30	0.52
F1 Score	0.41	0.50

ADVANCE MODEL IMPLEMENTATION:

CONVOLUTIONAL NEURAL NETWORK

- **Embedding Layer:** Maps each word to a high-dimensional vector space and captures semantic meaning.
- **Convolutional Layer:** Applies filters to the embedded vectors to extract local features.
- **Dense Layer:** Fully connected layer that interprets the features extracted by the convolutional layers.

Tokenize & Preprocess: Text data is converted to lowercase, punctuation is removed, and words are stemmed. This standard preprocessing step ensures uniformity and reduces noise in the data.

Train/Test Split: The preprocessed dataset is divided into training test sets to enable evaluation.

Build the CNN Model: The Keras library is used to construct the CNN model, comprising an embedding layer, convolutional layers with ReLU activation, and a softmax layer for classification.

Build the CNN model

The 128 parameter specifies the number of filters in the Conv1D layer .

The 5 parameter specifies the size of the filters .

The relu parameter specifies the activation function used in the Conv1D layer.

The 1 parameter specifies the number of output units in the Dense layer .

The sigmoid parameter specifies the activation function used in the Dense layer.

```
model = Sequential()
model.add(Embedding(input_dim=max_words, output_dim=100, input_length=max_sequence_length))
model.add(Conv1D(128, 5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=1e-4), metrics=['accuracy'])
```

Train model and make prediction

```
# Train the model
model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=3, batch_size=128, verbose=1)

# Tokenize and preprocess the test data
test_sequences = tokenizer.texts_to_sequences(nlp_test_df['question_text'])
X_test = pad_sequences(test_sequences, maxlen=max_sequence_length)

# Make predictions on the test data
test_preds = model.predict(X_test)
test_preds = (test_preds > 0.5).astype(int)
```

ERROR CALCULATION METRICS

An accuracy of 95% suggests that the model is making correct predictions for the vast majority of the data. This is a high level of overall correctness, indicating the model's ability to distinguish between sincere and insincere questions

A precision of 66% is a notable level of accuracy when it comes to predicting insincere questions. It implies that out of all the questions the model predicts as insincere, 66% are correct. This high precision is particularly valuable if you want to minimize false positives.

A recall of 45% indicates that the model is capturing almost half of the actual insincere questions. While not extremely high, it's still a respectable level of recall. This suggests that the model is effective at identifying insincere questions and minimizes false negatives.

F1 Score of 0.53 indicates that the model strikes a good balance between minimizing false positives and false negatives.

These metrics collectively suggest that your CNN model is performing well in classifying questions as sincere or insincere, and it's achieving a good trade-off between precision and recall.

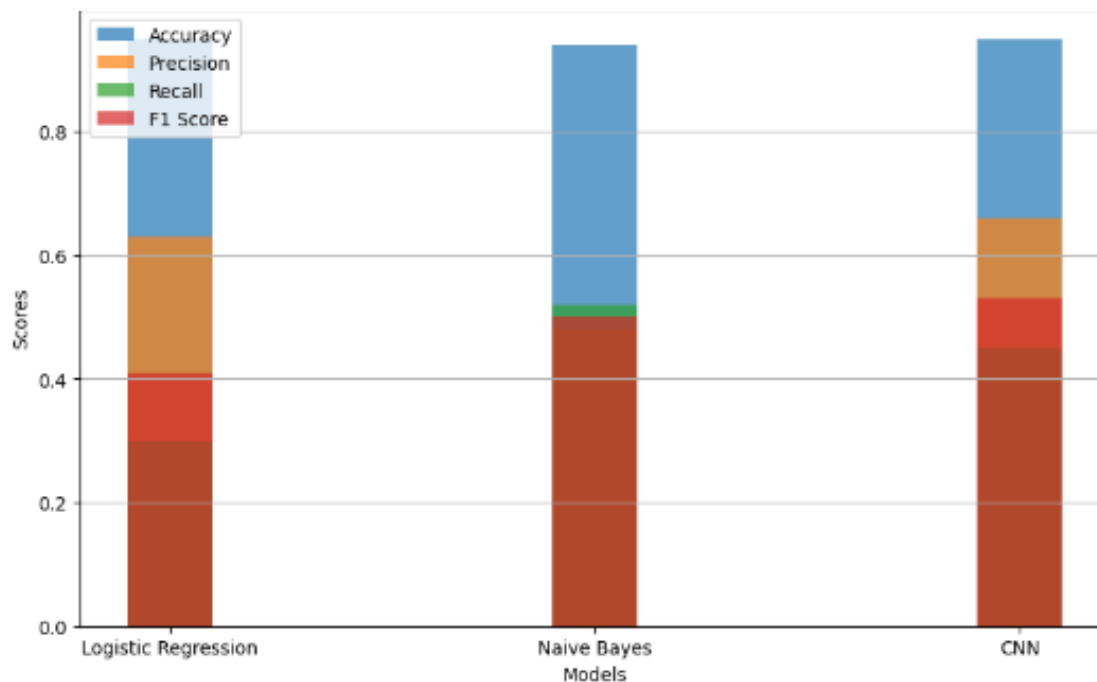
Calculate Precision , Recall , F1 score , Accuracy

```
] : # Calculate performance metrics
accuracy = accuracy_score(y_val, model.predict(X_val) > 0.5)
precision = precision_score(y_val, model.predict(X_val) > 0.5)
recall = recall_score(y_val, model.predict(X_val) > 0.5)
f1 = f1_score(y_val, model.predict(X_val) > 0.5)

print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')

12245/12245 [=====] - 118s 9ms/step
12245/12245 [=====] - 113s 9ms/step
12245/12245 [=====] - 114s 9ms/step
12245/12245 [=====] - 121s 9ms/step
Accuracy: 0.95
Precision: 0.66
Recall: 0.45
F1 Score: 0.53
```

RESULT



Interpretation of Model Performance

Logistic Regression

- **Accuracy (Logistic Regression): 0.95**
 - Logistic Regression achieved a high level of accuracy, indicating that it correctly classified a significant portion of the questions.
- **Precision (Logistic Regression): 0.63**
 - The precision score suggests that when the model predicts a question as insincere, it is correct about 63% of the time. This precision score is moderate but not exceptionally high.
- **Recall (Logistic Regression): 0.30**
 - Logistic Regression's recall score is relatively low, indicating that it misses a significant number of insincere questions. This leads to a substantial number of false negatives.
- **F1 Score (Logistic Regression): 0.41**
 - The F1 score is a harmonic mean of precision and recall. In the case of Logistic Regression, the F1 score suggests a moderate balance between precision and recall.

Naive Bayes (MultinomialNB)

- **Accuracy (Naive Bayes): 0.94**
 - Naive Bayes achieved a good level of accuracy, indicating overall correctness in classifying questions.
- **Precision (Naive Bayes): 0.48**
 - The precision score suggests that when Naive Bayes predicts a question as insincere, it is correct about 48% of the time. It's not as high as desired but provides a trade-off with recall.
- **Recall (Naive Bayes): 0.52**
 - Naive Bayes demonstrates a relatively higher recall, indicating its effectiveness in capturing insincere questions.
- **F1 Score (Naive Bayes): 0.50**
 - The F1 score for Naive Bayes suggests a balanced performance, achieving a trade-off between precision and recall.

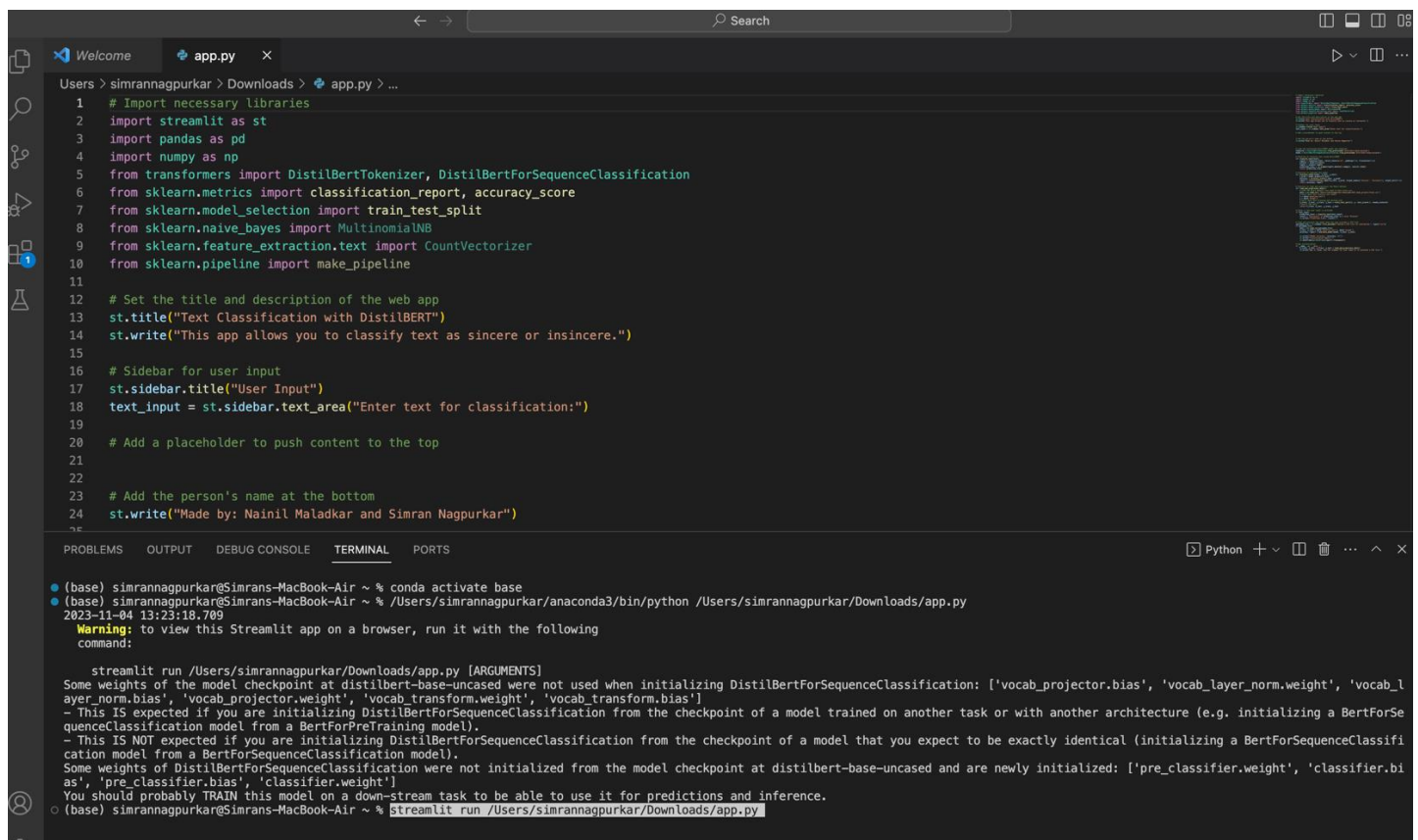
CNN (Convolutional Neural Network)

- **Accuracy (CNN): 0.95**
 - The CNN model, like Logistic Regression, achieved a high level of accuracy, correctly classifying a significant portion of the questions.
- **Precision (CNN): 0.66**
 - CNN demonstrates a higher precision score, indicating that when it predicts a question as insincere, it is correct about 66% of the time. This is a strong precision score.
- **Recall (CNN): 0.45**
 - While the recall score for CNN is respectable, it could be further improved to capture more insincere questions.
- **F1 Score (CNN): 0.53**
 - The F1 score for CNN is the highest among the models, indicating a good balance between precision and recall.

STREAMLIT APPLICATION BERT IMPLEMENTATION:

BERT (Bidirectional Encoder Representations from Transformers) is a popular and powerful algorithm for natural language processing (NLP) tasks, and it is often used in various applications, including streamlining applications.

- One of the key strengths of BERT is its ability to understand context from both the left and right sides of a word. Traditional models like LSTMs and RNNs process text sequentially in one direction, which can limit their understanding of context. BERT, on the other hand, employs a bidirectional architecture that captures the full context of a word within a sentence.
- BERT models are pretrained on large corpora of text data, which means they already have extensive knowledge about language and can be fine-tuned for specific NLP tasks. This pretraining allows BERT to learn rich and general-purpose representations of words and phrases, making it highly adaptable to a wide range of NLP applications.



```
1 # Import necessary libraries
2 import streamlit as st
3 import pandas as pd
4 import numpy as np
5 from transformers import DistilBertTokenizer, DistilBertForSequenceClassification
6 from sklearn.metrics import classification_report, accuracy_score
7 from sklearn.model_selection import train_test_split
8 from sklearn.naive_bayes import MultinomialNB
9 from sklearn.feature_extraction.text import CountVectorizer
10 from sklearn.pipeline import make_pipeline
11
12 # Set the title and description of the web app
13 st.title("Text Classification with DistilBERT")
14 st.write("This app allows you to classify text as sincere or insincere.")
15
16 # Sidebar for user input
17 st.sidebar.title("User Input")
18 text_input = st.sidebar.text_area("Enter text for classification:")
19
20 # Add a placeholder to push content to the top
21
22
23 # Add the person's name at the bottom
24 st.write("Made by: Nainil Maladkar and Simran Nagpurkar")
25
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

(base) simrannagpurkar@Simrans-MacBook-Air ~ % conda activate base
(base) simrannagpurkar@Simrans-MacBook-Air ~ % /Users/simrannagpurkar/anaconda3/bin/python /Users/simrannagpurkar/Downloads/app.py
2023-11-04 13:23:18.709
Warning: to view this Streamlit app on a browser, run it with the following command:

streamlit run /Users/simrannagpurkar/Downloads/app.py [ARGUMENTS]
Some weights of the model checkpoint at distilbert-base-uncased were not used when initializing DistilBertForSequenceClassification: ['vocab_projector.bias', 'vocab_layer_norm.weight', 'vocab_l
ayer_norm.bias', 'vocab_projector.weight', 'vocab_transform.weight', 'vocab_transform.bias']
- This IS expected if you are initializing DistilBertForSequenceClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSe
quenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing DistilBertForSequenceClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassifi
cation model from a BertForSequenceClassification model).
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized: ['pre_classifier.weight', 'classifier.bi
as', 'pre_classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
(base) simrannagpurkar@Simrans-MacBook-Air ~ % streamlit run /Users/simrannagpurkar/Downloads/app.py

```
26
27
28 # Load the pretrained DistilBERT model and tokenizer
29 tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
30 model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased')
31
32 # Function to classify text using DistilBERT
33 def classify_text(text):
34     inputs = tokenizer(text, return_tensors='pt', padding=True, truncation=True)
35     outputs = model(**inputs)
36     logits = outputs.logits
37     predicted_class = np.argmax(logits.detach().numpy(), axis=1).item()
38     return predicted_class
39
40 # Function to evaluate the model
41 def evaluate_model(model, X_test, y_test):
42     y_pred = model.predict(X_test)
43     accuracy = accuracy_score(y_test, y_pred)
44     report = classification_report(y_test, y_pred, target_names=['Sincere', 'Insincere'], output_dict=True)
45     return accuracy, report
46
47 # Function to load and preprocess the Quora dataset
48 def load_and_preprocess_data():
49     # Load the Quora dataset (you need to have train.csv)
50     data = pd.read_csv('/Users/simrannagpurkar/Downloads/NLP_team_project/train.csv')
51     # Split data into features and labels
52     X = data['question_text']
53     y = data['target']
54     # Split data into training and testing sets
55     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
56     # Return the data
57     return X_train, X_test, y_train, y_test
58
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

Some weights of the model checkpoint at distilbert-base-uncased were not used when initializing DistilBertForSequenceClassification: ['vocab_projector.bias', 'vocab_r_norm.weight', 'vocab_layer_norm.bias', 'vocab_projector.weight', 'vocab_transform.weight']
- This IS expected if you are initializing DistilBertForSequenceClassification from the checkpoint of a model trained on another task or with another architecture (e.g. another language)
- This IS NOT expected if you are initializing DistilBertForSequenceClassification from the checkpoint of a model that you expect to be exactly identical (initialization model from a BertForSequenceClassification model).
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight', 'pre_classifier.bias', 'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

- This application is designed to facilitate the classification of text as "sincere" or "insincere".
- When a user enters text in the provided input area and submits it, the code uses the DistilBERT model to analyze the text.
- It then predicts whether the input text is "sincere" or "insincere" based on the model's understanding of language and context. The predicted class is displayed as the output.

Deploy

User Input

Enter text for classification:

How can I start learning a new language effectively?

Upload a CSV file for evaluation:

Drag and drop file here

Limit 200MB per file • CSV

Browse files

Text Classification with DistilBERT

This app allows you to classify text as sincere or insincere.

Made by: Nainil Maladkar and Simran Nagpurkar

Predicted Class: Sincere


App is ready. Use the sidebar for user input or to evaluate a CSV file.

- **Example Input:** When a user enters the text "How can I start learning a new language effectively?" into your application, it performs a classification task.
- **Predicted Class:** The application processes the input text and assigns it to a specific category or class. In this particular case, the predicted category is referred to as "sincere."
- **Sincere Class:** "Sincere" is the label or category that the application has assigned to the input text. This class indicates that the question about starting to learn a new language effectively is recognized as genuine or serious in nature.

User Input

Enter text for classification:

Why are all politicians such corrupt liars?



Upload a CSV file for evaluation:

Drag and drop file here

Limit 200MB per file • CSV

Browse files

Text Classification with DistilBERT

This app allows you to classify text as sincere or insincere.

Made by: Nainil Maladkar and Simran Nagpurkar

Predicted Class: Insincere

App is ready. Use the sidebar for user input or to evaluate a CSV file.

- **Example Input:** When a user enters the text "Why are all politician such corrupt liars?" into your application, it performs a classification task.
- **Predicted Class:** The application processes the input text and assigns it to a specific category or class. In this particular case, the predicted category is referred to as "Insincere."
- **InSincere Class:** "InSincere" is the label or category that the application has assigned to the input text. This class suggests that the application believes that the text may be intentionally deceptive, dishonest, or insincere in its tone or intent.

The application allows users to input text for classification, upload a CSV file for model evaluation, and view classification results and model performance metrics. It leverages DistilBERT, a pre-trained language model, to perform text classification tasks.

STENGTH, WEAKNESS AND IMPROVEMENTS:

Logistic Regression

Strengths:

- User-friendly due to its simplicity and ease of interpretation.
- Capable of achieving a high level of accuracy, making it a reliable baseline model.

Weaknesses:

- May underperform when dealing with complex patterns in the data.
- Prone to overfitting without proper regularization.
- Lack of feature interaction can lead to missed nuances in text data.

Naive Bayes

Strengths:

- Efficient and straightforward to train with minimal computation.
- Offers balanced precision and recall, making it suitable for binary classification tasks.

Weaknesses:

- Assumes feature independence which may not hold true, affecting the classification quality.
- Its probabilistic nature can result in classification errors when probabilities are misestimated.

Convolutional Neural Network (CNN)

Strengths:

- Excellent at capturing complex patterns and local contexts in the data.
- High accuracy and precision rates, along with satisfactory recall and F1 scores.

Weaknesses:

- Requires significant data and computational resources to train.
- Can be more costly in terms of computational expenses compared to simpler models.

Analysis and Discussion

The evaluation of the three models suggests that CNNs provide a powerful mechanism for text classification, especially in identifying insincere questions in datasets like Quora's. Despite their computational cost, CNNs' ability to handle the intricacies of language makes them a strong candidate for complex NLP tasks.

Potential Improvements

Logistic Regression: Implementing advanced feature selection and regularization techniques can mitigate overfitting and improve model generalization.

Naive Bayes: Exploring feature engineering and alternative Naive Bayes variations could enhance its predictive capacity.

CNN: Integrating pre-trained word embeddings and expanding convolutional layers may increase the depth of text understanding.

Conclusion

Each model possesses unique characteristics that make it suitable for specific scenarios within text classification. Logistic Regression and Naive Bayes offer simplicity and computational efficiency, whereas CNNs excel in performance at the cost of complexity and computational demands. The choice of model should be influenced by the specific requirements of the task, data availability, and computational resources.

Additional Considerations

In applying these models, it is crucial to consider factors such as class imbalance, data preprocessing, and the trade-off between model interpretability and accuracy. Furthermore, the ever-evolving landscape of NLP suggests that continued research and experimentation with hybrid models and new architectures will be instrumental in advancing the field.

ACKNOWLEDGEMENT AND REFERENCE:

Majority of work has been divided among team members
Nainil worked on the data preprocessing and EDA along with feature engineering
For the text processing part, Nainil worked on TFIDF tokenization as an input to models.
Additionally, Model implementation including Naïve Bayes and CNN was carried out by Nainil
Nainil built a framework to use DistilBERT for training of web model.

Simran worked on some feature engineering and carried out text preprocessing using Bag of Words and implemented Stemming, stop word removal and vectorization.
Logistical regression was also implemented to processed data.
Simran made a streamlit application with input from user for interactive application for DistilBERT model.

References

1. NLP lecture python notebook by Prof. Junwei Huang
2. Medium blog on Quora Question Classification:
(<https://medium.com/@amitbalharakr93/quora-insincere-question-classification-2f19a973273b>)
3. Towards DataScience blog on Quora Question Classification:
(<https://towardsdatascience.com/quora-insincere-questions-classification-d5a655370c47>)
4. Microsoft Bing AI chat
5. `Quora Insincere Questions Classification`
(<https://www.kaggle.com/c/quora-insincere-questions-classification>) Kaggle Dataset