

# Identifying Quora Insincere Questions

Shu Lin Chan, Jiaqi Wang, Yue Wang, Minghui Zhu

---

## Introduction

Quora is the largest question-and-answer website, where millions of questions are be posted, answered and discussed. The dynamic online community possesses approximately 100 million users, thus it is important for Quora to make sure that every question being posted is sincere, so that its users are not offended by insincere questions. However, due to the large amount, it is not possible to identify those insincere questions manually. Thus, machine learning techniques were involved.

Recently, Quora has launched an competition on kaggle that aiming to find a way to identify toxic and insincere questions for them. The team found the competition a good combination of nlp and machine learning, both topics were covered in class, and thus a good choice for final project. We will attempt to solve this problem mainly using two different approaches, 1) Using Naive Bayes and Decision Tree classifiers, 2) Using machine learning techniques, more specifically Neural Networks to predict whether a question is sincere or not. We were interested in using machine learning as it has higher accuracy and is also a hot topic. It was a good opportunity to learn how it works and we could apply it elsewhere in the future.

Our goal for the project is to detect useful features that will help us distinguish insincere questions from sincere ones using natural language processing concepts that are covered in class. And to see the difference between traditional classifiers and neural networks in performing the same task. We want to build a successful classifier to identify those insincere questions. Due to the large amount of data we got (1.3 million inputs for train set, and 56 thousands for test), the project mainly runs on kaggle kernels.

---

---

## Data Preprocessing

Our dataset contains a train set with 1.3 million inputs and a test set of 56 thousands inputs. Each of these input is consisted of a unique id, a text question and a label 0 or 1 that classifies the question as insincere or sincere respectively, sample input is listed:

qid	question_text	target
00002165364db923c7e6	How did Quebec nationalists see their province as a nation in the 1960s?	0

Our goal in data preprocessing is to clean up the text question effectively so that our models could do a better job in prediction. We performed different data clean up techniques on three of our models - Naive Bayes, Decision Tree and Neural Networks.

For Naive Bayes and Decision Tree, we follow the standard data preprocessing steps: tokenize and remove stop words. However, for neural networks, we chose a different approach as we followed advice from experienced users before preprocessing:

1. **Don't use standard preprocessing steps like stemming or stopword removal when you have pre-trained embeddings**
2. **Get your vocabulary as close to the embeddings as possible**

[Source](#)[1]

The reason we are not following the standard data preprocessing steps when doing word count based feature extraction (e.g. TFIDF) is because those “unneeded information” might actually benefit the neural network to figure things out on its own when we add embeddings from language modelling and feature learning techniques. Information that is useless for humans may come in useful for machines.

Embedding layer is key to our neural network model. The purpose of using word embedding is to help machines discover the distributed representation. Embedding layer works like a function that maps a word to a certain combination of vectors, so that it reduce the amount of data that needs to train a model. Thus, the higher percentage the embedding could cover a data, the better prediction we could get. Since Kaggle has provided some useful embeddings files, we just chose one of them:

---

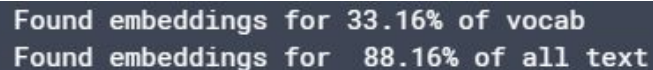
[glove.840B.300d](#)[2] - which is easy to use for beginners and has very good embeddings coverage.

We built a tool, “embedding\_analysis.py”, to calculate our embedding coverage, mainly for our training data set. Below is a snippet of the code:.

```
def check_coverage(vocab, embeddings_index):
    a = {}
    oov = {}
    k = 0
    i = 0
    for word in tqdm(vocab):
        try:
            a[word] = embeddings_index[word]
            k += vocab[word]
        except:
            oov[word] = vocab[word]
            i += vocab[word]
        pass
    print('Found embeddings for {:.2%} of vocab'.format(len(a) / len(vocab)))
    print('Found embeddings for {:.2%} of all text'.format(k / (k + i)))
    sorted_x = sorted(oov.items(), key=operator.itemgetter(1))[::-1]
    return sorted_x
```

This function tell us the percentage of the vocabulary from our data set and the percentage of text that are covered by embedding. Moreover, the function returns a list of words that are not covered by embedding, which will give us some hints about what we can do further to improve our embedding coverage.

Without any preprocessing, we got the result below:



```
Found embeddings for 33.16% of vocab
Found embeddings for 88.16% of all text
```

Only 33.16% of our vocabulary found embeddings, as well as 88.16% of text, making 12% of our data more or less useless.

To improve the embedding coverage, we tried several data cleaning methods:

- 
1. **Split punctuations from words:** our data set is a set of questions that are asked on quora, so many of words are in the form of word+?, eg. "India?", so we split them as "India" and "?". After this preprocessing, both "Indian" and "?" can be covered by our embeddings.
  2. **Replace numbers with unified symbol:** numbers are treated less importantly in our data set, so that we replace eg. "123" with "###", "2018" with "####"
  3. **Replace variations spellings of words to a single one and replace proper nouns to general ones:** e.g. replace "colour" with "color", "wwii" with "world war 2"

After experiments with each of these method, we found that only the first method can obviously improve our embedding coverage. Below is the result after splitting punctuations:

```
Found embeddings for 63.48% of vocab  
Found embeddings for 99.39% of all text
```

We can see that we improved the coverage of vocabulary from 33.16% to 63.48%, and text from 88.16% to 99.39%. Almost every word from all text can find its embedding, which is a big improvement of embedding coverage. Unfortunately, 36.52% of vocabulary is still not covered in embedding and the top 10 words are:

```
[('quorans', 858), ('brexit', 524), ('cryptocurrencies', 499), ('redmi', 383), ('kvpn', 356), ('paytm', 356), ('iiser', 346), ('ethereum', 334), ('iisc', 278), ('jinpig', 211)]
```

No method can help the embedding to cover these words, since we are not capable of modifying the embedding itself. Nevertheless, we are satisfied with the results of the embedding. Thus, we will continue to train the models using features. The code for this can be found in 'embedding\_analysis.py'.

## Data Analysis

The data preprocessing is only a part of training our models. To get a better prediction score, we also need good features, no matter what model we use. Thus, we used some tools to help us find out the characteristics of the data in "data\_analysis.py".

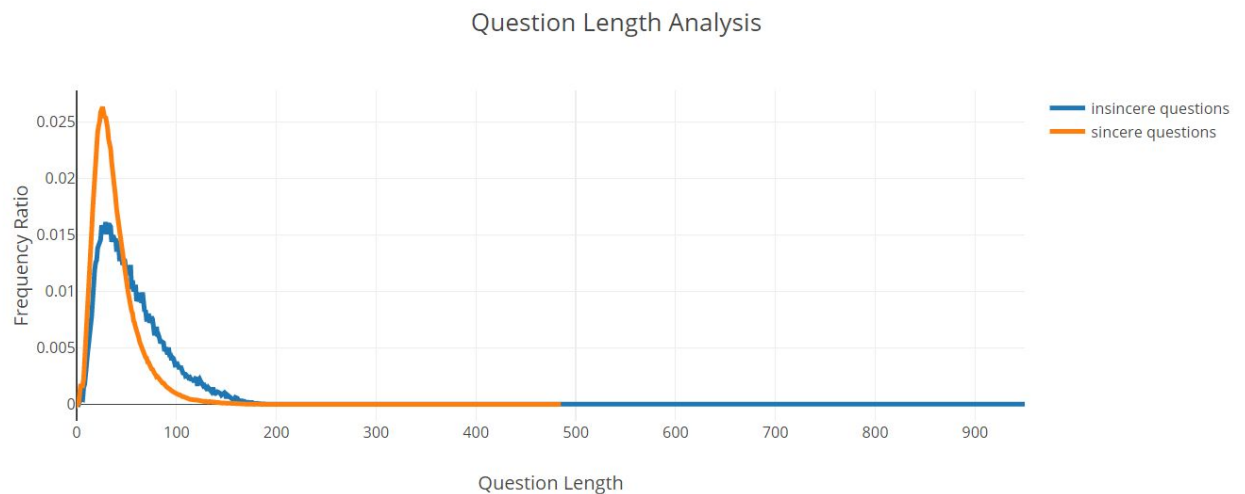
---

In this file, we mainly use two packages: [textstat](#)[3], [plotly](#)[4] as our data analysis tools. Textstat is a great tool which facilitates linguistic computation and analysis. Plotly is a powerful tool in data visualization. With the help of these tools, we are able to characterize and visualize the linguistic features of insincere questions and sincere questions, which help us build some useful features to identify insincere questions. Below is the result of our data analysis:

## 1. Question Length Analysis

Are sincere questions more verbose? Or are insincere questions more verbose? Or are there no differences? With these questions in mind, we start to explore the differences in length between those two different kinds of questions. The plot and table shows the result that insincere questions are actually longer than the sincere ones (SD = 10.9).

Source file: [plot](#), [table](#)



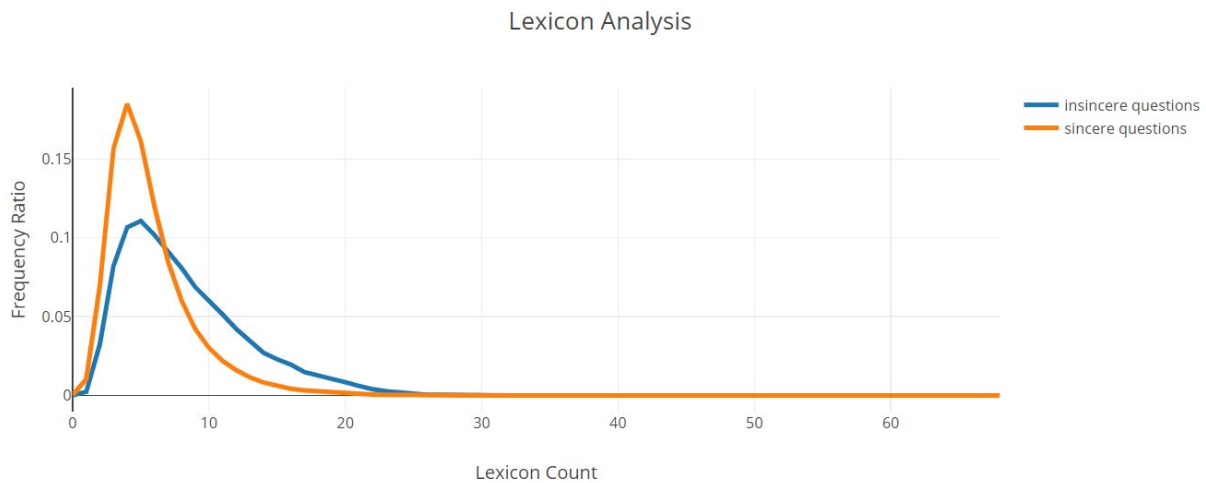
Statistical Measures	Insincere Question	Sincere Question
Mean	55.31975792756353	38.1137963876841
Standard Deviation	32.730975392487814	21.877426761825912
Median	48.0	33.0
Maximum	950	485
Minimum	0	0

---

## 2. Lexicon Analysis

Since we have discovered that insincere questions are generally longer than sincere questions, we wondered if insincere questions contain more lexicon than the sincere ones. After doing the lexicon analysis, we learned that there is a difference, but the difference in lexicon are not that distinct ( $SD = 1.4$ ) as shown in the following graphs.

Source file: [plot](#), [table](#)

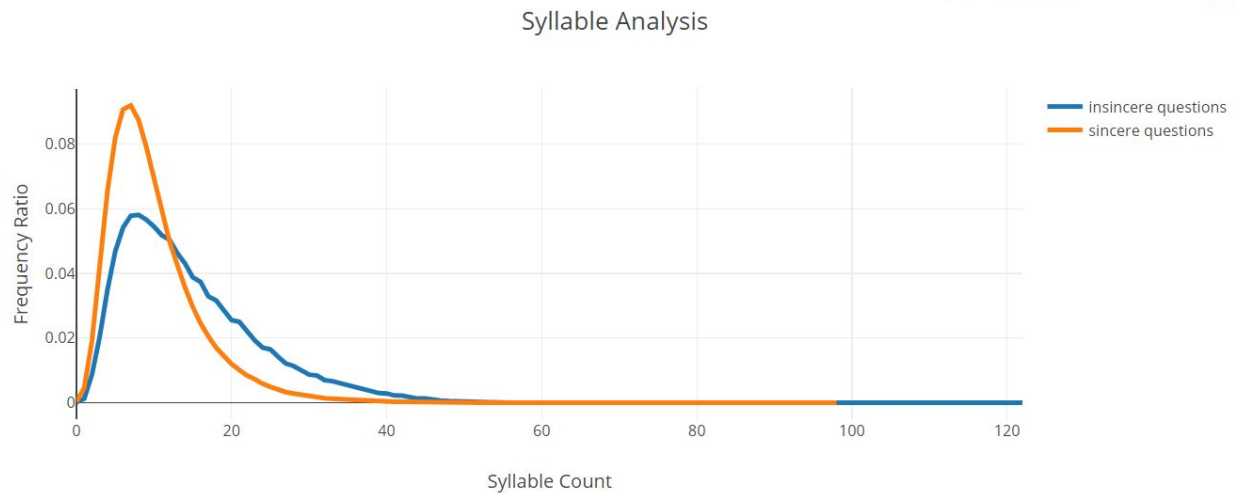


Statistical Measures	Insincere Question	Sincere Question
Mean	8.22627906258684	5.721129578034582
Standard Deviation	4.577483160243389	3.1682391645397354
Median	7.0	5.0
Maximum	42	68
Minimum	0	0

## 3. Syllable Analysis

As the results of lexicon analysis were not that promising, we moved on to analyze the syllables because syllables are the basic units of speech. We did this by using a textstat package to run the analysis. Unfortunately, the results of syllable analysis was not promising as well ( $SD = 2.7$ ). Although the syllable analysis could distinguish insincere questions from sincere questions better than lexicon analysis, the difference was still minor. Below is the results graph.

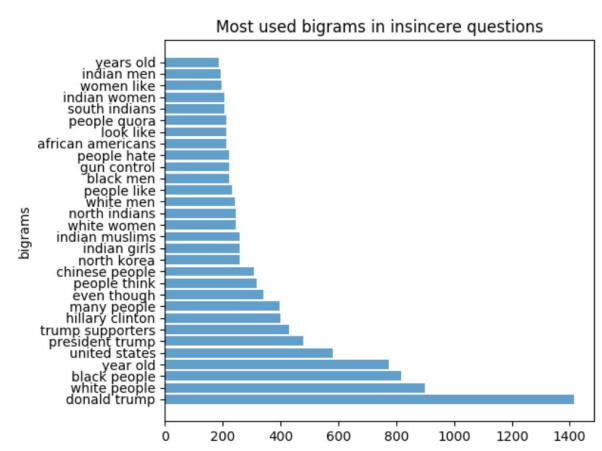
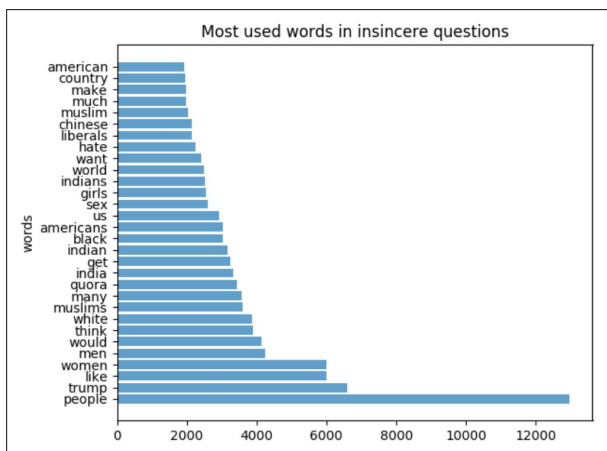
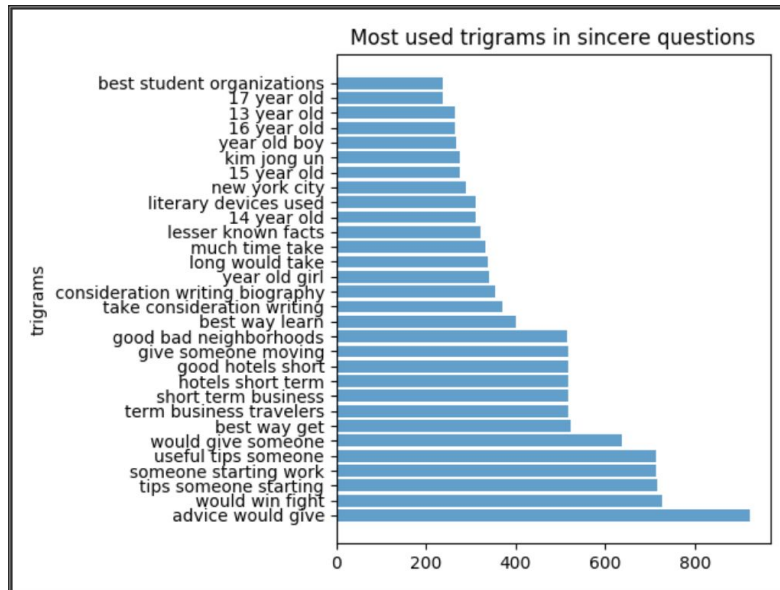
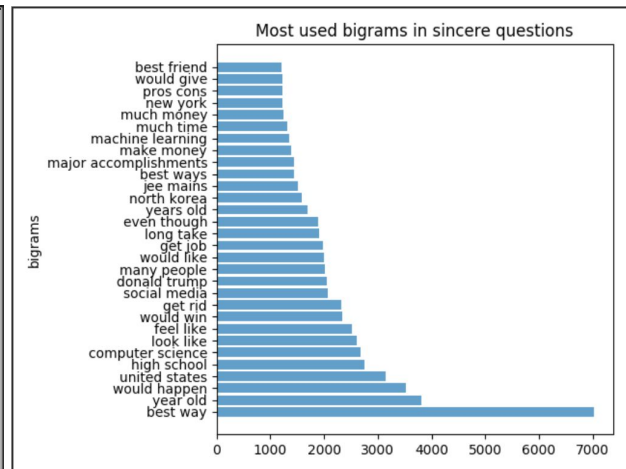
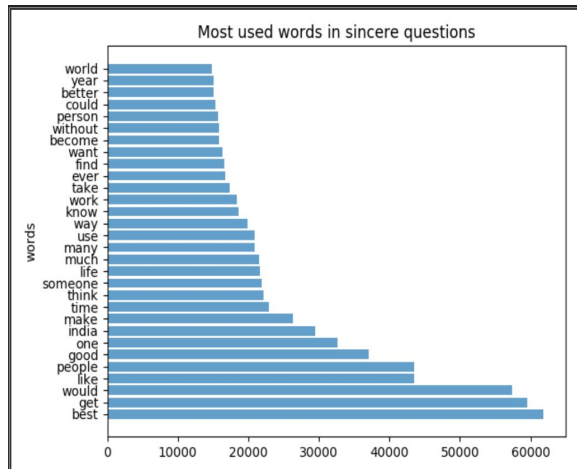
Source file: [plot](#), [table](#)



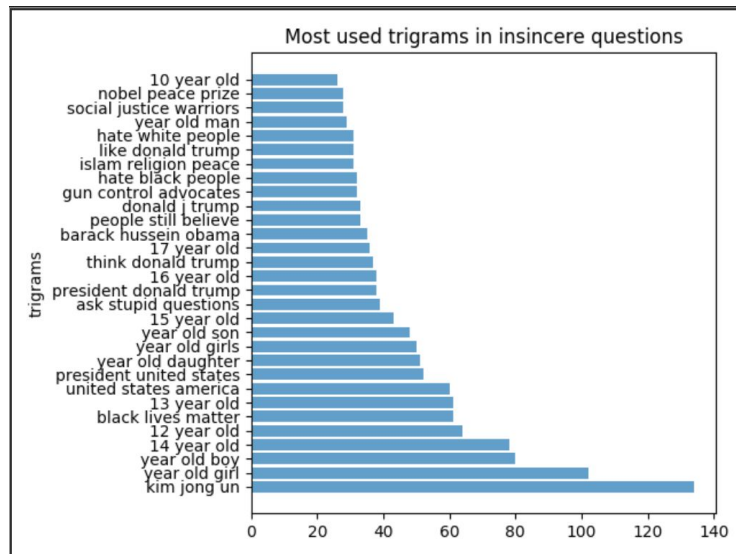
Statistical Measures	Insincere Question	Sincere Question
Mean	14.600071016148455	10.068241443137383
Standard Deviation	8.781736036816339	6.001547584847915
Median	13.0	9.0
Maximum	122	98
Minimum	0	0

#### 4. N-gram Analysis

Apart from question length, syllable and lexicon analyses were disappointing, thus we tried another approach - to see the most used n-grams in sincere questions and insincere questions, and hoped that we could find a difference. In the following result charts, we deduced that the most used n-grams in sincere questions and insincere questions are indeed different. For readability, only 30 most used n-grams were shown.







## Features

Since many questions were on the fence between sincere and insincere, the team had to decide on a few solid features where the question is indeed toxic. There were obvious features where if the question consisted of a curse word then it is insincere. But, there were also questionable insincere questions such as 'What is the reason why we really need Bitcoin?'. This question was considered as insincere even though it has a sarcastic intonation to it. Kaggle competition has defined some insincere questions, according to their definition, there are three distinct types of insincere questions[5]:

1. Any question with a non neutral tone
2. Any question that is disparaging or inflammatory
3. Any question with sexual content

Combined with the information we learned from our data analysis and kaggle, we have come up with 18 features mainly in two categories - raw-text-based features and context-based features.

First two features are baseline features, all the tokens and all the tags. We word\_tokenizer and pos\_tag package in nltk to generate these features. They were used to compare different models.

---

## Raw-Text-Based Features:

As we have discovered during data analysis, insincere questions are generally longer than sincere questions, we decided on using the length of questions as a feature. We expanded on this feature and used number of tokens and number of types in the question as well.

1. **Length of questions**
2. **Number of tokens in the question**
3. **Number of types in the question**

```
features[i][0] = len(texts[i])
features[i][1] = len(tokens[i])
features[i][2] = len(set(tokens[i]))
```

We then added more features like the difference in number of punctuations, the difference in number of stopwords and the average word length between insincere questions and sincere questions. The reason for these features is that insincere questions are more verbose than sincere ones, so we assume they have more counts on punctuations, stopwords and word length.

4. **Number of punctuations**
5. **Number of stopwords**
6. **Average word length in the question**

```
puncts = [',', '.', ':', ')', '(', '-', ...]
stop_words = set(nltk.stopwords.words('english'))

for j in tokens[i]:
    if j in puncts:
        puncts_num += 1
    if j in stop_words:
        Stop_words_num += 1
    sum_words_length += len(str(j))

features[i][3] = punct_num
features[i][4] = stop_word_num
features[i][5] = sum_words_length / len(tokens[i])
```

---

## Context-Based Features:

Raw-text-based features were insufficient in differentiating sincerity, so we added context-based features as well. Some of those context-based features are aligned with the three types of insincere questions that Kaggle has provided, others are n-gram features that we learned from data analysis.

### 1. Exaggerated adjectives

A question with a non neutral tone has an exaggerated tone to underscore a group of people or is rhetorical and meant to imply a statement about a group of people. Hence we added a feature that included exaggerated adjectives such as “super”, “extreme” and “least”. This feature is considered as a binary feature, since an increase in the number of exaggerated adjectives in a question does not mean the question is more exaggerated. Thus, we didn’t measure the degree of exaggeration, we only measured the appearance of exaggeration. The list of exaggerated words were created on our own.

```
exaggerated = ['best', 'worst', 'super', 'extreme', 'most', 'least', 'never'...]

def find_exaggerated(lst):
    for w in lst:
        if w.lower() in exaggerated:
            return 1
        else:
            return 0
```

### 2. Comparison

A question that is disparaging or inflammatory is when the writer attacks a protected group of people or tries and disparage a characteristic that can not be fixed or measured. Basically, any form of racism and stereotype falls under this category. We created this feature by whether the question contains words like “than” or “compared” because comparisons usually include those words. Same with exaggerated adjectives, we do not measure the degree of comparison. For example, a question that contains “more than”

---

with two words in the comparison list, does not contain more information than a question that contains only “better”. We built our list of comparison words from daily experience.

```
comparison = ['better', 'worse', 'than', 'more', 'less', 'compared', 'between',  
'among', 'whereas'...]
```

```
def find_comparison(lst):  
    for w in lst:  
        if w.lower() in comparison:  
            return 1  
        else:  
            return 0
```

### 3. Whether a question contains a group of people

As stated above, a disparaging or inflammatory questions might contains disparaging or comparison of groups of people. Thus, we detect whether there are some specific groups of people in our question text. We used a list of countries to represent people’s nationalities:

```
groups = ['united states of america', 'afghanistan', 'albania', 'algeria',  
'andorra', 'angola'...]  
def find_groups(lst):  
    count = 0  
    for w in lst:  
        if w.lower() in groups:  
            count += 1  
    return count
```

### 4. Contain sexual content

Any questions with sexual content are toxic. There is a feature of list of sexual words found online[6].

```
dirty = ['ass', 'assfucker', 'asshole', 'assholes', 'asswhole'...]  
def find_dirty(lst):  
    count = 0  
    for w in lst:
```

---

```
if w.lower() in dirty:
    count += 1
return count
```

## 5. Most used N-grams in sincere/insincere questions

While analyzing our data set, we found that the n-grams used in sincere questions and insincere questions are significantly different. Thus, we built 6 different features - most frequently used words: unigrams, bigrams and trigrams - in sincere questions and insincere questions respectively. We just use the most frequently used 100 unigrams/bigrams/trigrams in features to increase efficiency in training our models. For this feature, we kept count of how many times these unigrams/bigrams/trigrams appeared in the question.

```
def generate_ngrams(text, n_gram):
    token = [token for token in text.lower().split(" ") if token != "" if token not
in stop_words if token not in puncts]
    ngrams = zip(*[token[i:] for i in range(n_gram)])
    return [" ".join(ngram) for ngram in ngrams]

def sin_list(train_sin, n_gram):
    freq_dict = defaultdict(int)
    for sent in train_sin["question_text"]:
        for word in generate_ngrams(sent, n_gram):
            freq_dict[word] += 1
    sin_list = sorted(freq_dict.items(), key=lambda x: x[1])[:-1]
    sin_list = [w[0] for w in sin_list][:100]
    return set(sin_list)

def insin_list(train_insin, n_gram):
    freq_dict = defaultdict(int)
    for sent in train_insin["question_text"]:
        for word in generate_ngrams(sent, n_gram):
            freq_dict[word] += 1
    insin_list = sorted(freq_dict.items(), key=lambda x: x[1])[:-1]
    insin_list = [w[0] for w in insin_list][:100]
    return set(insin_list)
```

---

## Models

There are several classifiers we could have used to test our features but we have chosen Naive Bayes, Decision Tree and Neural Networks.

### 1. Naive Bayes

In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

### 2. Decision Tree

Decision tree learning uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves).

For those two classifiers, we firstly tried with two baseline features: all the tokens and all the tags, neither of the results are satisfactory (Naive Bayes = 0.291, Decision Tree = 0.125). Then we tried some combination of features, detailed score could be found in the Results section.

Naive Bayes classifier and Decision Tree classifier are based on a common principle - both classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. Because of this, we cannot determine the relationship and patterns between features. Yet, multiple features can appear in one question (e.g. group of people and comparison word, comparison word and another comparison word). Hence, the performance of Naive Bayes classifier and Decision Tree classifier are not satisfactory. The result shows that although we get a relatively high score on validation data (highest validation score: 0.993), those two classifiers perform even worse than random choice between 0 and 1 when it comes to test data (highest test score: 0.291).

### 3. Neural Networks

---

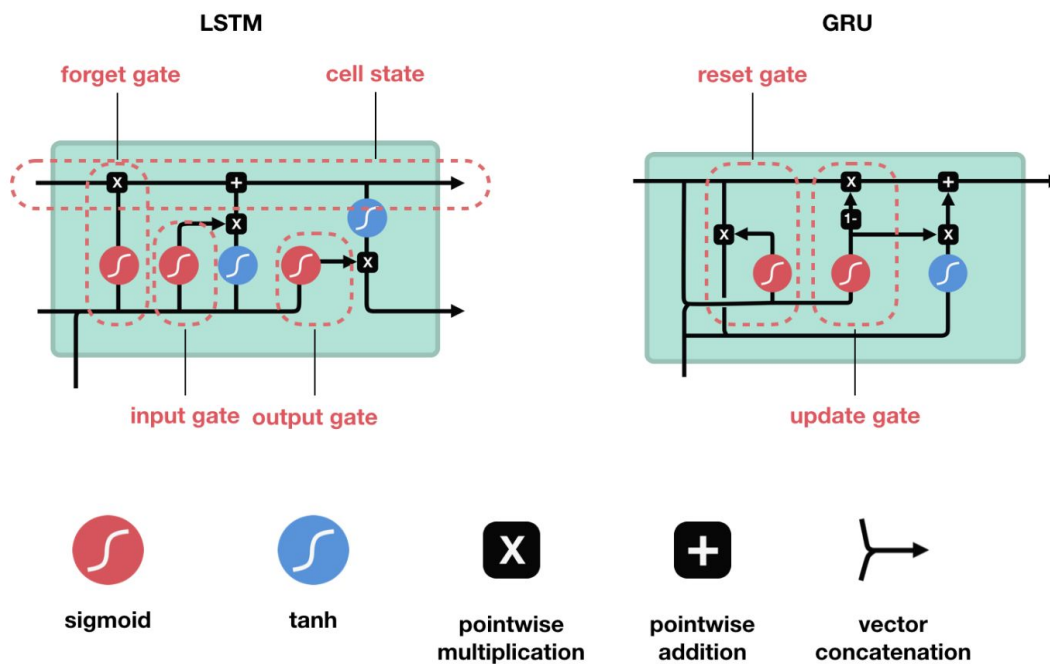
Different neural network models for different sets of features were chosen. Text embeddings for unigrams like Word2vec, GloVe were given so for unigrams we used Embedding layers followed by LSTM layer and GRU layer. We used this model of unigrams as baseline. Yet, there were no embeddings for further features like bigrams, trigrams. So we built another MPL network and merged two models together.

### a. Long Short-term Memory (LSTM)

An RNN composed of LSTM units is often called an LSTM network. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

### b. Gated Recurrent Units (GRU)

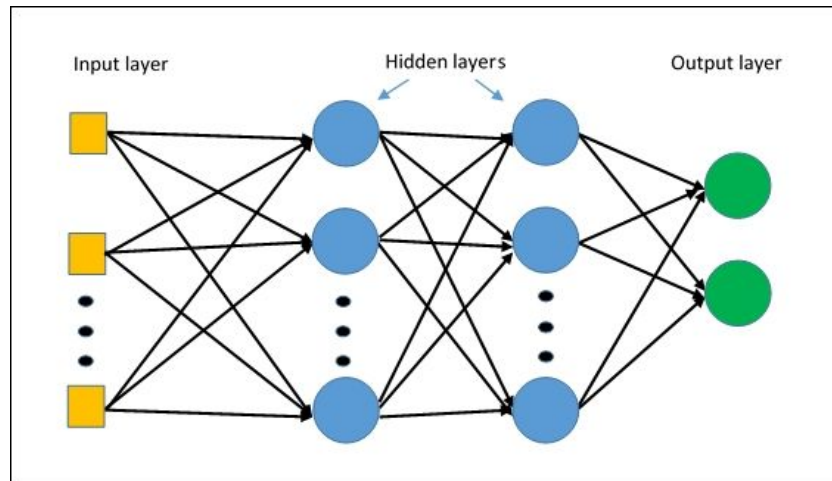
Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks. Similar to LSTM but with fewer parameters, as they lack an output gate, so that have a higher training speed.



---

### c. Multilayer Perceptron (MLP)

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. An MLP consists of, at least, three layers of nodes: an input layer, a hidden layer and an output layer.



We explored neural networks with a large amount of features mentioned above for instance the length of questions, whether the question contains a group of people, whether it contains exaggerated, etc. Detailed result of how each feature performed are listed in the result section.

## Results

Since our data set was too large, we had to run it on kernel. This limited the amount of times we could submit (5 times per day per group) to get the actual accuracy of our model. Thus, for both of the models, we tried a couple of features combinations. We spent most of our time working on Neural Networks because neither the Naive Bayes nor the Decision Tree Classifiers generated a decent accuracy score, in fact the scores were even worse than random choice.



---

## 1. Naive Bayes/ Decision Tree Classifier

Model	Feature Category	Features	Highest Submit Score
Naive Bayes	Raw-text	all the tokens + all the tags	0.291
Decision Tree	Raw-text	all the tokens + all the tags	0.125
Naive Bayes	Raw-text + Context	Length of questions + dirty word + comparison + groups + exaggerated + length of tokens	0.169
Decision Tree	Raw-text + Context	Length of questions + dirty word + comparison + groups + exaggerated + length of tokens	0.041
Naive Bayes	Context	most used sincere unigrams + most used sincere bigrams + most used sincere trigrams + most used insincere unigrams + most used insincere bigrams + most used insincere trigrams	0.049
Decision Tree	Context	most used sincere unigrams + most used sincere bigrams + most used sincere trigrams + most used insincere unigrams + most used insincere bigrams + most used insincere trigrams	0.058
Naive Bayes	Context	most used insincere unigrams + most used insincere bigrams + most used insincere trigrams	0.040
Decision Tree	Context	most used insincere unigrams + most used insincere bigrams + most used insincere trigrams	0.053
Naive Bayes	Raw-text + Context	length of questions + dirty word + comparison + groups + exaggerated + length of tokens + most used sincere unigrams + most used sincere bigrams + most used sincere trigrams + most used insincere unigrams + most used insincere bigrams + most used insincere trigrams	0.207
Decision Tree	Raw-text + Context	length of questions + dirty word + comparison + groups + exaggerated + length of tokens + most used sincere unigrams + most used sincere bigrams + most used sincere trigrams + most used insincere unigrams + most used insincere bigrams + most used insincere trigrams	0.104

---

## 2. Neural Network:

Model	Feature Category	Features	Highest Submit Score
LSTM + GRU	Raw-Text	length of questions	0.695
LSTM + GRU	Raw-Text	length of question + number of tokens	0.693
LSTM + GRU	Raw-Text	length of question + number of types	0.690
LSTM + GRU	Context	groups	0.692
LSTM + GRU	Context	exaggerate	0.69
LSTM + GRU	Context	comparison	0.691
LSTM + GRU	Context	dirty word	0.692
LSTM + GRU	Context	groups + comparison + exaggerate + dirty word	0.689
LSTM + GRU	Context	groups + length of questions	0.693
LSTM + GRU	Context	dirty word + length of questions	0.693
LSTM + GRU	Context	groups + length of questions + dirty word	0.692
LSTM + GRU	Context	Most used insincere trigrams	0.691
LSTM + GRU	Context	len of questions + most used insincere unigrams	0.692
LSTM + GRU	Context	len of questions + most used insincere bigrams	0.690
LSTM + GRU	Context	len of questions + most used insincere trigrams	0.691
LSTM + GRU	Context	len of questions + most used sincere unigrams	0.691
LSTM + GRU + MLP	Raw-Text	length of questions	0.693
LSTM + GRU + MLP	Raw-Text	length of questions + number of punctuations	0.691
LSTM + GRU + MLP	Raw-Text	length of questions + number of stop words	0.691

---

LSTM + GRU + MLP	Raw-Text	length of questions + average length of tokens	0.691
---------------------	----------	--	-------

## Conclusion

The summary of what was done on the data set for naive bayes and decision tree classifiers is performing data preprocessing, removing stopwords, and lemmatizing the text. For neural networks, we use embeddings to reduce the amount of data used to train the model. After we experimented on several data cleaning techniques to further benefit our model's accuracy, we found that only split punctuations resulted in improvements in embedding coverage.

Next, we learned that insincere questions were generally longer than sincere questions and that the words (unigrams, bigrams and trigrams) that were used in those two question sets are significantly different from each other as well while doing data analysis. We then built our feature sets based on those two features to validate the models.

As expected, Neural network models work much better than Naive Bayes classifier and Decision Tree classifier. It is because features in Naive Bayes and Decision Tree are independent from each other, so the model is unable to learn the hidden relationships between features. While testing with features, we found that for neural networks models, adding more features is not always a good idea, because having one feature can perform better than having a combination of several features. This is proven by the highest score we got, which is 0.695 just from one feature: length of questions. However, adding more features to Naive Bayes classifier and Decision Tree classifier would indeed improve the accuracy, even though the results are still not promising.

Currently, we have no clue why only the length of questions boosts the accuracy so we want to try out different neural network models, as well as different layer densities to improve accuracy. We also would like to perform a more thorough data analysis, so that we can extract more linguistic features. Since we are all beginners in computational linguistic and data science, we will continue to learn different data processing techniques as well as model building and training techniques!

---

## References

[1] How to preprocessing when using embeddings,

<https://www.kaggle.com/christofhenkel/how-to-preprocessing-when-using-embeddings>

[2] GloVe: Global Vectors for Word Representation, <https://nlp.stanford.edu/projects/glove/>

[3] Textstat: python package to calculate readability statistics of a text object - paragraphs, sentences, articles, <https://github.com/shivam5992/textstat>

[4] Plotly: Modern Analytics Apps for the Enterprise, <https://plot.ly/#/>

[5] Quora Insincere Questions Classification:

<https://www.kaggle.com/c/quora-insincere-questions-classification/data>

[6] Bad words list: <https://gist.github.com/aldeka/3456141>

[7] Naive Bayes, Wikipedia, [https://en.wikipedia.org/wiki/Naive\\_Bayes\\_classifier](https://en.wikipedia.org/wiki/Naive_Bayes_classifier)

[8] Decision Tree Learning, Wikipedia, [https://en.wikipedia.org/wiki/Decision\\_tree\\_learning](https://en.wikipedia.org/wiki/Decision_tree_learning)

[9] Long Short-term Memory, Wikipedia,

[https://en.wikipedia.org/wiki/Long\\_short-term\\_memory](https://en.wikipedia.org/wiki/Long_short-term_memory)

[10] Gated Recurrent Unit, Wikipedia, [https://en.wikipedia.org/wiki/Gated\\_recurrent\\_unit](https://en.wikipedia.org/wiki/Gated_recurrent_unit)

[11] Multilayer Perceptron, Wikipedia, [https://en.wikipedia.org/wiki/Multilayer\\_perceptron](https://en.wikipedia.org/wiki/Multilayer_perceptron)

[12] Illustrated Guide to LSTM's and GRU's: A step by step explanation, Towards Data Science,

<https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

[13] Multi Layer Perceptron, O'Reilly,

<https://www.oreilly.com/library/view/getting-started-with/9781786468574/ch04s04.html>