

# QUORA INSINCERE QUESTIONS CLASSIFICATION

EE 258 Project II (Team InSinSeer)

Haely Shah

Michael von Pohle

MS (Electrical), Fall'18

San Jose State University

The Quora logo, featuring the word "Quora" in a red, serif font.A red rectangular box containing white text. The text reads: "Insincere Questions Classification", "Detect toxic content to improve", and "online conversations".

**Insincere Questions Classification**

Detect toxic content to improve  
online conversations

## 1) Dataset Description:

Quora is a popular question-and-answer website where the Quora community asks, answers, modifies, and organizes questions. With 300 million monthly users, Quora has over 38 Million questions as of December 2018 [1]. To detect and filter out the toxic content, thus adhering to its “Be Nice. Be Respectful.” Policy, Quora has used manual review and Machine Learning to predict if a question is sincere or not. Quora classifies insincere questions as questions that have a non-neutral tone, are based on false information, and/or use sexual content for shock value.

Insincere Questions Classification is an ongoing Kaggle competition for developing the most accurate Machine Learning models to weed out insincere questions. The dataset provided has five files:

- 1) Embeddings
- 2) Training data (train.csv, 124.21 MB)
- 3) Test data (test.csv, 5.24 MB)
- 4) A sample submission (sample\_submission.csv, 1.3 MB)

The training data consists of 3 data fields: unique question identifier- ‘qid’, Quora question text- ‘question\_text’, and the sincerity label- ‘target’ with target value ‘1’ for insincere questions, and target value ‘0’ for sincere questions. A snippet of the training dataset is shown in Figure 1. The training dataset consists of 1,306,122 questions of which 80,810 (6.19%) are insincere.

|   | qid                  | question_text   | target |
|---|----------------------|---|--------|
| 0 | 00002165364db923c7e6 | How did Quebec nationalists see their province as a nation in the 1960s?          | 0      |
| 1 | 000032939017120e6e44 | Do you have an adopted dog, how would you encourage people to adopt and not shop? | 0      |
| 2 | 0000412ca6e4628ce2cf | Why does velocity affect time? Does velocity affect space geometry?               | 0      |
| 3 | 000042bf85aa498cd78e | How did Otto von Guericke use the Magdeburg hemispheres?                          | 0      |
| 4 | 0000455dfa3e01eae3af | Can I convert montra helicon D to a mountain bike by just changing the tyres?     | 0      |

Figure 1: A snippet of training data

The test data contains just the ‘qid’ and ‘question\_text’ as the data fields and has 56,370 questions to be tested. We have done some basic visualization on train and test data set questions. Figure 2 shows a normalized histogram of word count per question in training and test dataset. Both the datasets have a similar word count and distribution. The average word length per question for training data is 12.80 while it is 12.75 for the test data. The standard deviation for both the dataset is 7. A similar distribution for sincere and insincere questions from the training data is shown in Figure 3. A disparate distribution is seen in sincere and insincere question, with a higher spread in insincere questions.

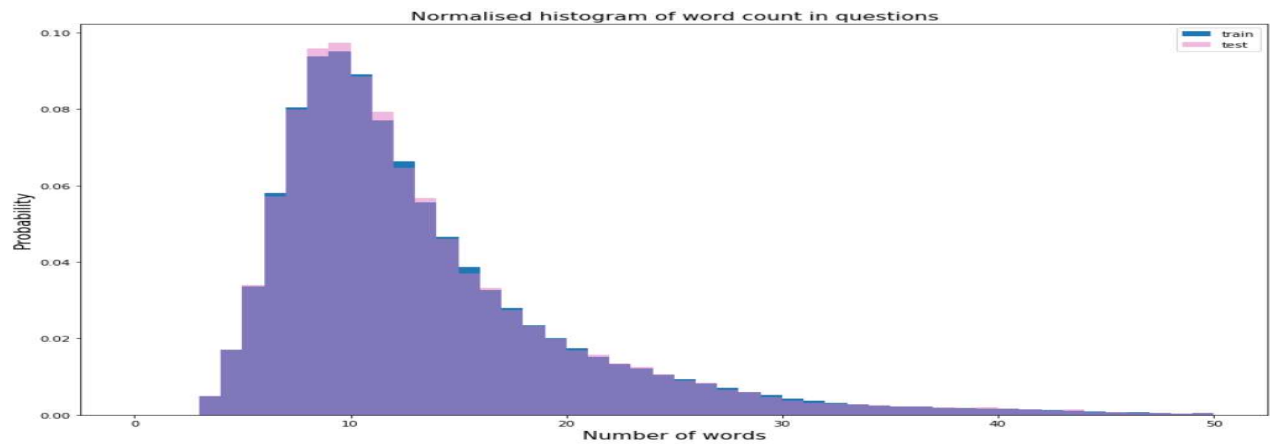


Figure 2: Normalized histogram of word count per questions (Train vs Test)

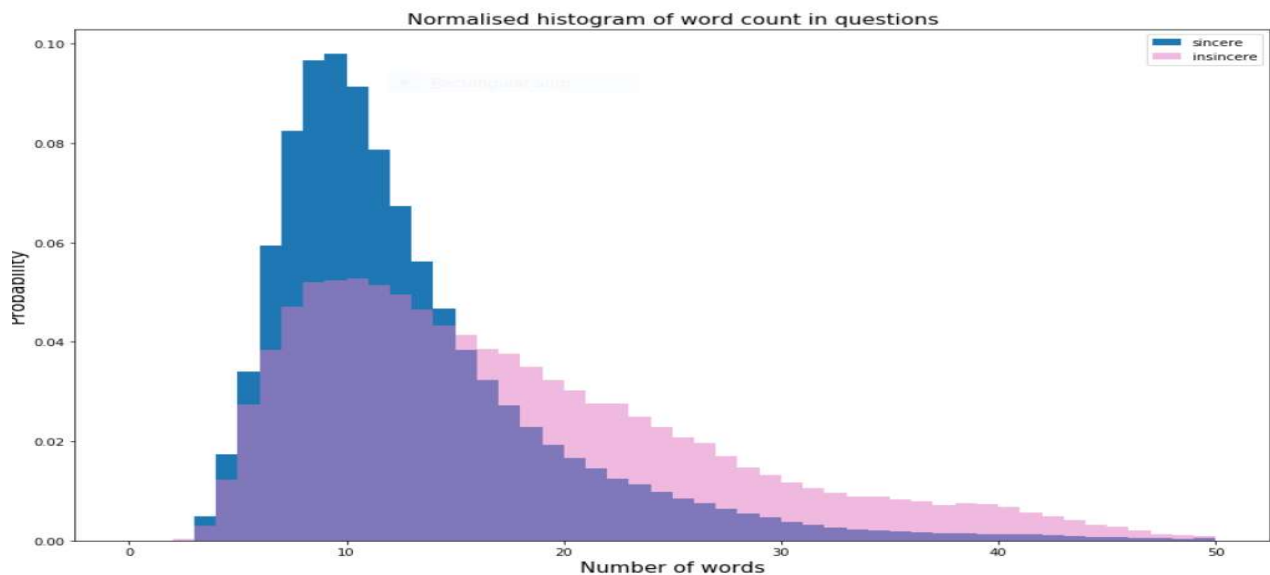


Figure 3: Normalized histogram of word count per questions (Sincere vs Insincere)

A check for missing values shows that there are no missing values in the entire dataset (training and test alike). After basic preprocessing on data (explained in detail in Section 2 (Pre-Processing)), we created word clouds of the most common words in sincere and in insincere questions. A sample word-cloud for insincere questions is shown in Figure 4.

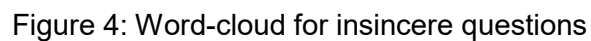


Figure 4: Word-cloud for insincere questions

## 2) Pre-Processing:

An important step to creating any machine learning model is pre-processing. As a part of text pre-processing, we used already existing NLP (Natural Language Processing) libraries and functions. All of the preprocessing is done using NLTK (Natural Language Toolkit).

The first step was to convert all the characters to a common case in order to avoid duplicates due to capitalized and regular letters. We used 'lambda' functions to convert all the text data to lower case. To further reduce the size of training data, we removed all punctuations form the data as punctuations do not add any extra information. The next step was removing certain common words in the language which are universal and do not contribute to the sentiment of the question but are more used for grammar and structure of the sentence. Some of these words are: 'the', 'I', 'you', and 'are'. These words are equally likely to be present in sincere and insincere question types. These words are referred to as 'stopwords'. Removing stop-words reduces the dataset greatly. We used the predefined libraries to remove stop-words.

We were not sure if running a spellcheck over the entire dataset would help the model or not. A plethora of spelling errors are present in any online user community but given the huge amount of proper nouns (names of leaders, historical figures etc.) spellcheck might worsen the model. After seeing a few proper nouns ('montra' to 'contra'), we did not go ahead with spelling correction in further models. As a part of pre-processing, we tried various predefined functions like stemming and lemmatization. Stemming usually stems out the common suffixes from the words, thus reducing the words 'kinder', 'kindest', and kind to 'kind'. But this technique reduces the words without considering the context or the root word in the language. One of the words 'velocity' in the dataset was reduced to 'veloc'. So, NLTK has a better technique called 'lemmatization'. Lemmatization reduces the inflectional words in the language to the common root word. After this basic pre-processing, the question text was greatly reduced and a lot cleaner. A snippet of the processed data is shown in Figure 5.

|   | qid                  | question_text                                       | target |
|---|----------------------|---|--------|
| 0 | 00002165364db923c7e6 | quebec nationalist see province nation 1960s        | 0      |
| 1 | 000032939017120e6e44 | adopted dog would encourage people adopt shop       | 0      |
| 2 | 0000412ca6e4628ce2cf | velocity affect time velocity affect space geometry | 0      |
| 3 | 000042bf85aa498cd78e | otto von guericke used magdeburg hemisphere         | 0      |
| 4 | 0000455dfa3e01eae3af | convert montra helicon mountain bike changing tyre  | 0      |

Figure 5: A snippet of the processed data frame

In text mining, weights are used to retrieve information about the words. Ranking these weights can greatly help optimize the model. One of the most popular weight ranking algorithms is Tf-idf (Term frequency – inverse document frequency). Term frequency measures the frequency of a

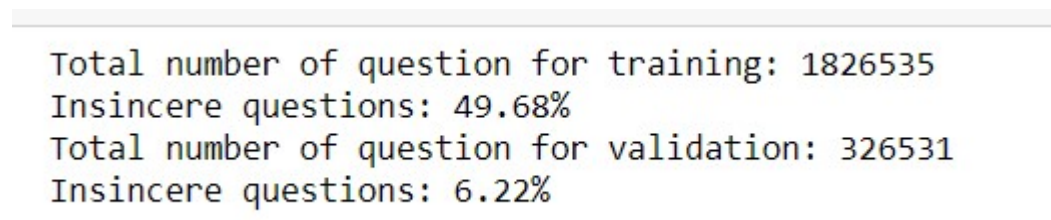
term in the text and inverse document frequency measures the importance of the term. Idf scales up the rare words to get a better score on the weights. Variations of tf-idf are used depending on the requirements of the project. We computed the idf of each of the words and created a new column that has a list of the idf's for the words in that row. Thus, the data frame now has an extra column of idf's for the corresponding word.

Once the words have been reduced to a common base form and a common lower case, we tokenize the words. These converts the words and the alphanumeric characters to tokens or instances in sequence processing. We have used TextBlob library for tokenization. After tokenizing the question text, we use embeddings technique. Embeddings is mapping the words to a vector as a process of providing input to the model. We can either train our own embeddings or use the already created file. We have used 'Glove' embedding provided with the data. All of the words are now converted into a 300 Float word vector/numbers.

Since the questions are not necessarily of the same word length, each of the questions will have different number of tokens and a different length of the idf list. So, we pad the questions and the idf list to a fixed number. We computed the longest question length and padded all the other questions to that maximum length (52 here). Similarly, we padded the idf list to 52 too. This will ensure that all the questions are of a similar dimension.

### 3) Baseline Model:

Given the highly imbalanced training data (6.19% insincere questions), we duplicate the insincere data to create a more balanced training data. This duplication (data augmentation) is done after training and validation split. We have split the data randomly (seed=13) to create a 25% validation data and 75% training data). Figure 6 shows a snippet of basic data information of the training and validation data.



```
Total number of question for training: 1826535
Insincere questions: 49.68%
Total number of question for validation: 326531
Insincere questions: 6.22%
```

Figure 6: Data Augmentation

For our baseline model, we have used a recurrent neural network (RNN). The RNN has 3 layers – embedding, LSTM, Output. All the layers are fully connected networks. The embedding layer receives the tokenized embedded words as its input. For our baseline model, we had not used idf feature extraction. The embedding layer has 52 inputs and converts them to float 300. These floats are fed as inputs to the long short term memory (LSTM) layer. The LSTM layer has tanh as its activation function and has 200 output neurons. The outputs are fed to the output dense layer which classifies the questions using sigmoid activation function. A snippet of the model



summary is shown in Figure 7. The model was trained with 512 batch size and 3 epochs. We have used F1 as the metric for all our models and the raw F1 score (validation) for the baseline model is 0.58. The validation accuracy for the same was 0.89.

| Layer (type)                     | Output Shape    | Param #  |
|----------------------------------|-----------------|----------|
| embedding_1 (Embedding)          | (None, 52, 300) | 59796600 |
| lstm_1 (LSTM)                    | (None, 200)     | 400800   |
| dense_1 (Dense)                  | (None, 1)       | 201      |
| Total params: 60,197,601         |                 |          |
| Trainable params: 401,001        |                 |          |
| Non-trainable params: 59,796,600 |                 |          |

Figure 7: Baseline model

#### 4) Model Improvement:

The first thing we did to improve the model further was include a calculation for an optimal F1 threshold. This code computes F1 score over multiple threshold and then provides the most optimal F1 score. Using this, the baseline model F1 score jumped from 0.58 to 0.61.

We then added the extra feature of idf (discussed in pre-processing). We trained the new inputs along with the original inputs and saw an improvement in F1 score from 0.61 to 0.64.

After fixing the extra feature, we explored various models and hyper-parameters. We trained the same baseline model for different activation functions (RELU and Sigmoid), different epoch sizes (1, 2, 3, 4, 5), different batch sizes (512, 1024), different LSTM output neurons (100, 200, 300). We also tried different dropout rates (0.1, 0.2, 0.3). all of these tweaks resulted into slightly different F1 scores except for RELU. RELU gave an F1 score of 0.34-0.46 which was lower than any of the sigmoid models. Also, increasing the epoch size from 1 to 3 gave us an increased raw F1 score from 0.48 to 0.59. All the other hyper parameters did not significantly affect the model performance.

After this, we tried adding an additional layer of fully connected hidden layer with 16 neurons. We trained this model for 3 epochs and 300 LSTM outputs. The accuracy shot up to a raw F1 of 0.63. But the optimal F1 did not show significant improvement. We also looked into Bi-Directional LSTM. We did not see significant improvements in test F1 score but saw the training F1 score rise from 0.92 to 0.96.

For our final submission, we have extra features with 100 LSTM output neurons. We use regular networks for the extra feature and then concatenate the model with LSTM. Figure 8 shows the snippet of the final model summary. This gave us a test F1 score of 0.63.

| Layer (type)                     | Output Shape    | Param #  | Connected to                        |
|----------------------------------|-----------------|----------|-------------------------------------|
| word_tokens (InputLayer)         | (None, 77)      | 0        |                                     |
| embedding_1 (Embedding)          | (None, 77, 300) | 65975100 | word_tokens[0][0]                   |
| features (InputLayer)            | (None, 77, 1)   | 0        |                                     |
| concatenate_1 (Concatenate)      | (None, 77, 301) | 0        | embedding_1[0][0]<br>features[0][0] |
| lstm_1 (LSTM)                    | (None, 100)     | 160800   | concatenate_1[0][0]                 |
| output (Dense)                   | (None, 1)       | 101      | lstm_1[0][0]                        |
| Total params: 66,136,001         |                 |          |                                     |
| Trainable params: 160,901        |                 |          |                                     |
| Non-trainable params: 65,975,100 |                 |          |                                     |

Figure 8: Final model summary



## 5) Discussions and Conclusions:

We had explored a lot of text mining and RNN papers and articles for the project. A vast majority of the pre-processing was influenced from Kaggle discussions and Medium articles. RNN with LSTM seems to work well with text sequence analysis. Given more time and resources, we wanted to learn and look more into different models (GRU and XgBoost) and explore more into tf-idf feature. We wanted to explore more into additional features like grammar check or character length. We also wanted to calculate a differential frequency for certain words for each type of the data. This differential frequency would be like a difference between idf in sincere and in insincere text. We did look into basic statistics and normalization on how to do it but could not figure it out. We also plan on working on this in the winter break with different models and features.

A significant amount of our time was devoted to code exploration and debugging. This was by far the single biggest issue we had faced.

Kaggle ranking: 1552

|      |     |                     |   |       |   |    |
|------|-----|---------------------|---|-------|---|----|
| 1552 | new | EE258_F18_InSinSeer |   | 0.638 | 2 | 4m |
|------|-----|---------------------|---|-------|---|----|

## 6) References:

- 1) Quora.com official dataset provides a count of the total questions, when you ask a new question
- 2) Ultimate guide to deal with text data, URL: <https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/>, December 16, 2018
- 3) Generating word cloud in Python, URL: <https://www.geeksforgeeks.org/generating-word-cloud-python/>, December 16, 2018