Quora Insincere Questions Classification

Detect toxic content to improve online conversations

Mohsin Karovaliya, Vikram Patel, Muhammad Ali Haider, Aishwarya Tirumala
mrkarova(200255290), vpatel22(200262802), mhaider2(200224608), atiruma(200258166)

1 Abstract

Quora is a platform that empowers people to learn from each other. On Quora, people generally ask questions and answer each other's questions on various topics such as science, technology and politics. Questions are the building blocks of Quora. Good questions help others share their knowledge with the world and also provide information and solutions about the question to the person asking. Sometimes, questions asked by someone can be deliberately provocative which include being prejudicial framing or calls to confirm hateful stereotypes. 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 are: a non-neutral tone, is disparaging or inflammatory, isn't grounded in reality or uses sexual content. In this project we will focus on classifying these questions as sincere or insincere. Our main goal is to identify and remove insincere and trolling questions to maintain a civil and respectful discourse. After some research, we came up with combinations of feature sets where we summarize the question, understand its sentiment which helps in the classification of the questions. Further, we focus on creating models using various classification algorithms and implement these models for the detection of insincerity in a question.

2 Background and Literature Survey

Research on insincerity of text on the internet is fast growing since it is a major concern. In today's world where social media plays such an important role in our lives, we were able to closely relate the detection of such insincere questions on Quora with the insincere comments and textual aggression on the internet. According to the research by Antonela, Juan Manuel and Daniel Godoy on detecting textual aggression on the internet using deep learning strategies, they implemented support vector machines and recurrent neural network for analyzing a wide range of characters, words and sentiment. They also discuss some challenges which will prove to be similar to the challenges in our project such as, difficulty in detecting the insincerity of a question due to lack of grammatical correctness which might hinder the usage of NLP tools. The notes on deep learning for NLP by Antoine J.-P. Tixier helped us in narrowing down our models and implementing the Neural Networks we have used.

In another study Nobata et al. (2016) aimed at detecting hate speech on 2 million online comments from Yahoo! Finance and News. Four types of features were considered: n-grams, linguistic, syntactic, and embedded semantic features. Comments were pre-processed by normalizing numbers, replacing unknown words with the same token, replacing repeated punctuation. Results showed that combining all features achieved the best F-Measure results. Our focus now is to weed out these questions by developing models that identify and flag out the insincerity. We first read the text and understand the intent of the text by performing sentiment analysis and identifying and insincerity in it.

3 Methods

3.1 Approach

The first task is to perform preprocessing of words using techniques like tokenization, lemmatization, correction of misspelling since we need to remove redundant and incorrect information. After preprocessing, the data needs to be converted into a particular representation so that it can be recognized by the machine learning algorithms. Hence we convert each word in a particular question into a vector of size 300.

3.1.1 GLoVe

GLoVe, an unsupervised learning algorithm can be used to procure word vector representation. The training for the model is performed using aggregated global word-word co-occurrence statistics which we get from a corpus, and the resulting representations show interesting linear substructures of the word vector space. There are two ways to use embeddings:

- Use the unsupervised learning algorithm on a labelled corpus to generate vectors for every word in it with the vector space related to that particular dataset.
- Map the words in the dataset to the existing data of word embeddings provided by the source.

We chose the second approach due to the vastness of the vocabulary covered by embeddings (840 billion tokens) which will yield more context information from the test case.

3.2 Models

In order to provide accurate output, we need to train the model in such a way that the context of every word in the corpus (how each word fits in the question wrt the other words in the question) is taken into account. Hence while training the model we need to consider neighbouring words. Hence a sequential model which is context aware is required. Some of the models are as follows:

3.2.1 Convolutional Neural Network

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks. They have neurons with weights and biases which can be updated with the help of training. The function of each neuron is to receive an input, perform some dot product calculations and return a score for every classification. It is composed of layers like convolutional layer, a pooled layer, a linear rectification layer, and a fully connected layer. Each of the weights and biases in the network is optimized using back propagation[2]. Since the hidden layers compute various dot product operations over inputs which are neighbouring words in this case, it can easily capture the context of the words. Though recent work suggests that convolutional layers may directly be stacked on top of each other , the elementary construct of the CNN is a convolution layer followed by a pooling layer.

Following are the most effective layers on CNN:

Convolution Layer:

Convolution operation preserves the spatial relationship between sections of data by learning features using small vectors of input data. This operation takes a small vector of a fixed size and performs the convolution mathematical operation over the vector input. This is done inorder to capture the essential features (context) of the vector input(sentence) across the number of words equivalent to the size of the convolution vector.

Maxpooling Layer:

Max pooling is a process in which the input is discretized based on a fixed sample size in order to downsample the input, thus extracting only the essential features and allowing to make assumptions regarding the binned sub-regions. It is done by applying a max operation to non-overlapping subregions of the input.

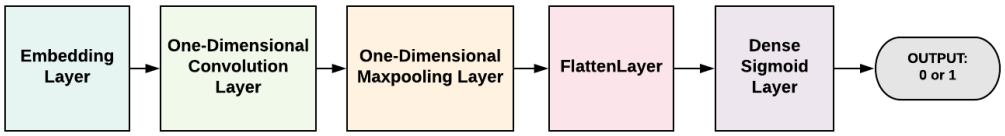


Figure 1: CNN layers

3.2.2 Recurrent Neural Network(RNN)

Recurrent Neural Network(RNN) are a type of Neural Network where the output from previous step are fed as input to the current step. In cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is Hidden state, which remembers some information about a sequence.

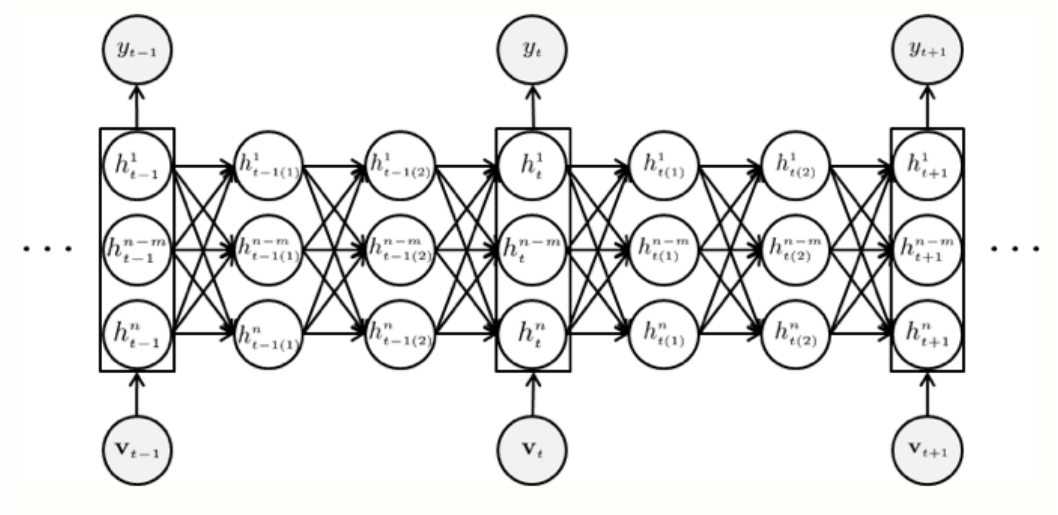


Figure 2: Recurrent Neural Network.
Taken from stackexchange

We have implemented two such RNNs in our project. One is the Bidirectional LSTM and GRU.

3.2.3 Bidirectional LSTM

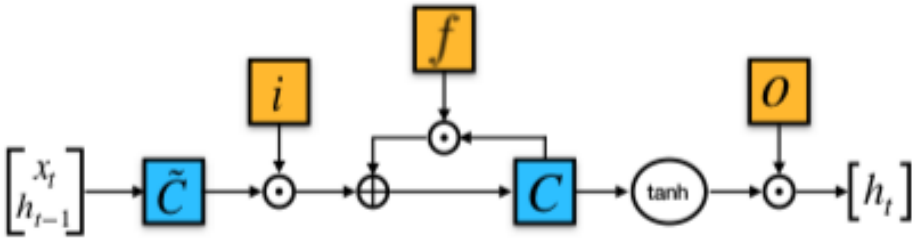
Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture which has feedback connections. It can process a single data point as well as entire sequences of data (such as continuous text, speech or video) which is our current requirement. Given an input sequence $x = (x_1, ..., x_T)$ a bi-directional LSTM network calculates the hidden forward $\overrightarrow{h_t}$ and backward $\overleftarrow{h_t}$ states based on the current input x_t with its previous hidden $state.h_{t-1}$ and memory cell c_{t-1}, as the following:

$$
\begin{aligned}
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
g_t &= tanh(W_c x_t + U_c h_{t-1} + b_c) \\
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
c_t &= i_t \otimes g_t + f_t \otimes c_{t-1} \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
h_t &= o_t \otimes tanh(c_t)
\end{aligned}
$$

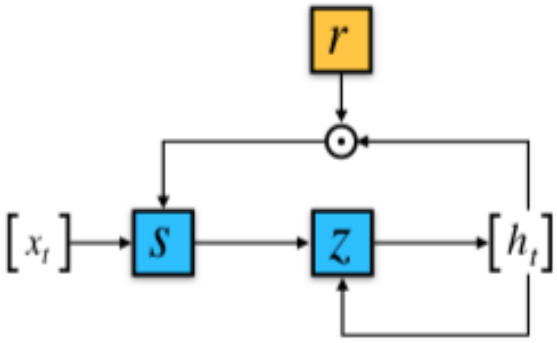
Here σ is sigmoid function, and $\otimes$ is the element-wise multiplication. Here, W_i, W_c, W_f, W_o represents weights and b_i, b_c, b_f, b_o represents bias.

3.2.4 Gated Recurrent Unit

The GRU is the newer generation of Recurrent Neural networks and is pretty similar to an LSTM. In GRU's, instead of the cell state, hidden state is used to transfer information. It also only has two gates, a reset gate and update gate.The update gate is similar in operation to the forget and input gate of an LSTM. which decides the details to be added or removed. The reset gate is used to decide how much previous details is to be forgotten.



(1) Long Short-Term Memory



(2) Gated Recurrent Unit

Figure 3: Illustration of LSTM and GRU gates
Taken from article- Recent trends in deep learning based on NLP

4 Experiment

4.1 Data Description

The dataset which we used contains text representing questions asked, target which classifies questions as sincere or insincere and unique identifier for each question. The dataset was obtained from Kaggle. Since there are only two possible outcomes for our problem, we will be using a binary classification approach. The questions are unstructured and pre-processing is required before applying classification algorithms. Besides, the dataset is highly imbalanced with only about 6% of records tagged as insincere. Therefore downsampling will also be required to reduce high bias.

4.2 Data Pre-processing

We applied the following procedure to pre-process our data:

1. De-contraction: De-contraction is the method of expanding contracted words such as "won't", "haven't" to "will not", "have not", etc. We used a regular expression to convert all the possible contractions to their expanded versions.
2. Cleaning misspells: Our data-set contains many misspellings. We have corrected spellings of all high-frequency words. This process was manual and required observing data and mapping them to their corresponding correct words.
3. spacing misspells: Our data-set contains many spacing misspells. We have corrected spacing of the spellings of all high-frequency words. This process was manual and required observing data and mapping them to their corresponding correct words. For example - deadbody is corrected to deead body.
4. Cleaning latex: Our dataset contains few latex representations which are not useful while creating vectors. We have replaced them into corresponding English words which will be used to create vectors.
5. Spacing the punctuations: In order to separate the punctuations from the words we have made sure that the punctuations are properly spaced.
6. Replacing numbers: Exact numerical values in dataset do not signify vital information and can be discretized to reduce the problem space. We have replaced digits with hashes of the same length. For example, 8 will be converted to #, 28 to ##, 238 to ### and so on. Also, we have added spaces before and after a digit for the purpose of differentiation.

4.3 Tokenization and Embeddings

In order for the data to be readable by our machine learning algorithms, we needed to convert the data into a recognizable format. For this, we have first performed tokenization. In tokenization, each word in our text after pre-processing is mapped to an integer token. These integers are determined based on the frequency of their occurrence in the data. If the frequency of a particular word is high it gets a high integer token value. We have used the Keras Tokenizer API.

After tokenization, we needed to convert the integer tokens to real valued vector. We have implemented three different types of embeddings and compared their results.

- We make use of the pre-trained GloVe embedding where word vector representation exists for 42 Billion different words.
- We have also created embeddings using the GloVe unsupervised algorithm to our particular dataset. There are some drawbacks using the pre-trained embedding in our situation, we might not get the accurate vector of the required word in terms of its context. Also, if we consider the vector distance between two similar words, the distance will be closer in the custom word embedding as compared to the pre-trained embedding
- We tried another method where random vectors are created by the embedding layer and are passed to the models. The models train using these vectors and using back propagation, they keep updating the vectors in the embedding layer.

4.4 Expermient Setup

Before feeding our data to algorithms we need to assign vector size, maximum number of features and length of input. Vector size represents the length of vector used to represent a token. When we used the GloVe pre-trained embeddings, our vector size is 300. Maximum features are the top M words in the dataset with the highest frequency of occurrence. Therefore our entire dataset will be represented only in the form of these features, any other words not present in these features will be ignored. We have set feature size to 250K as they cover 98.79% of total words in our dataset. Fixing question length helped us to feed records to the input layer of neural networks with a fixed number of neurons. In our case, we have set question length to 75 since it is greater than the maximum number of words in the longest question in the training dataset. In order to fix question length, we have used padding, that is if question length is less than 75 we have padded left with zero vectors. Whereas, if the question length is greater than 75 we truncate the right section. After generating a vector representation of each record, we have used stratified sampling with K-Fold Cross-Validation. Since the data is imbalanced, stratified sampling will help to reduce bias and K-fold CV will help to reduce generalization error. In our case, we have set K to 5. For each K we have run 10 epochs. We have built and trained our deep learning models using Keras Tensorflow API. Keras TensorFlow is a high-level API for building and training deep learning models. It's used for fast prototyping, state-of-the-art research, and production.

Another one of the open-source libraries we used is Scikit-learn. Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, naive bayes, and regression, and is designed to interoperate with the Python numerical and scientific libraries - NumPy and SciPy. We used sklearn's GridSearchCV function for these hyper-parameter tuning. We tried following variances for given parameters:

(a) Activation Function (A): [Sigmoid, Softmax, ReLu]

(b) Dropout (D): [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

(c) Batch Size (B): [100, 250, 500]

(d) Epochs (E): [6, 18, 50]

(e) Optimizer (O): [Adadelta, Adam, Adamax]

We found the best parameters to be A = Sigmoid, D = 0.2, B = 500, E = 5, O = Adam

We hypothesised that GRU will perform better amongst all the three models. As GRU belongs to the family of RNNs it has better context retention mechanism and solves gradient diminishing problem as in CNN. Besides, it does not consist of redundant gates as present in the case of LSTM. After running all the three models and calculating metrics such as ROC AUC and Kappa Scores, we found the GRU performed as good as LSTM. Also after calculating performance metrics we found that GRU took about 15% less time to train. We also hypothesised that custom trained embeddings (using unsupervised GloVe algorithm on our dataset) will give better results as compared to pre-trained GloVe embeddings. This is because, in former case embeddings would be generated based on our specific dataset resulting in creation of distinctive values in the vector space. However, after training our models and performing comparative analysis we found that pre-trained embeddings performed better. This could be the case because pre-trained embeddings were generated based on much larger dataset.

5 Results and Discussions

The accuracy we had obtained for each of the models were:
CNN: 90 percent, LSTM: 96 percent and GRU: 96 percent.

Since the accuracy was unusually high we learnt that our data was imbalanced. Accuracy can be high in cases when the data of sincerity is much higher than the insincere questions. Since our data was biased we cannot use accuracy as a measure. Therefore, we check for the ROC values. The ROC values for three of the models are shown below.
We compared our results using the three different embeddings implemented. We got the ROC values for each model with the three embeddings and realized that the pre-trained embeddings performed

the best. The ROC values for the three models using the pre-trained embeddings being: CNN: 0.71, LSTM: 0.80 and GRU: 0.80.

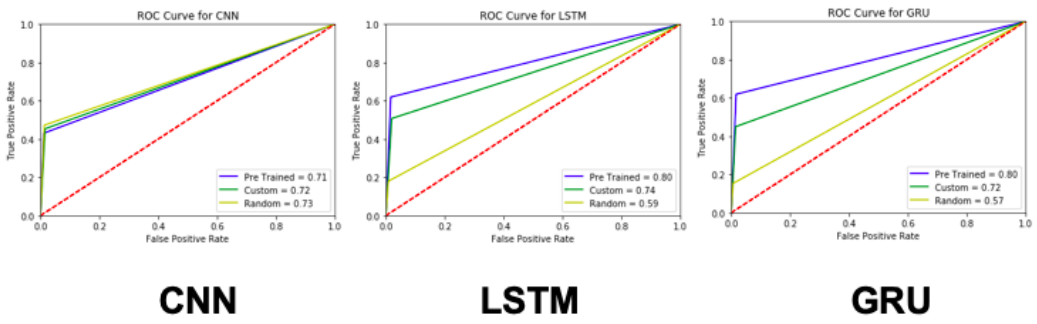


Figure 4: ROC values for the three models.

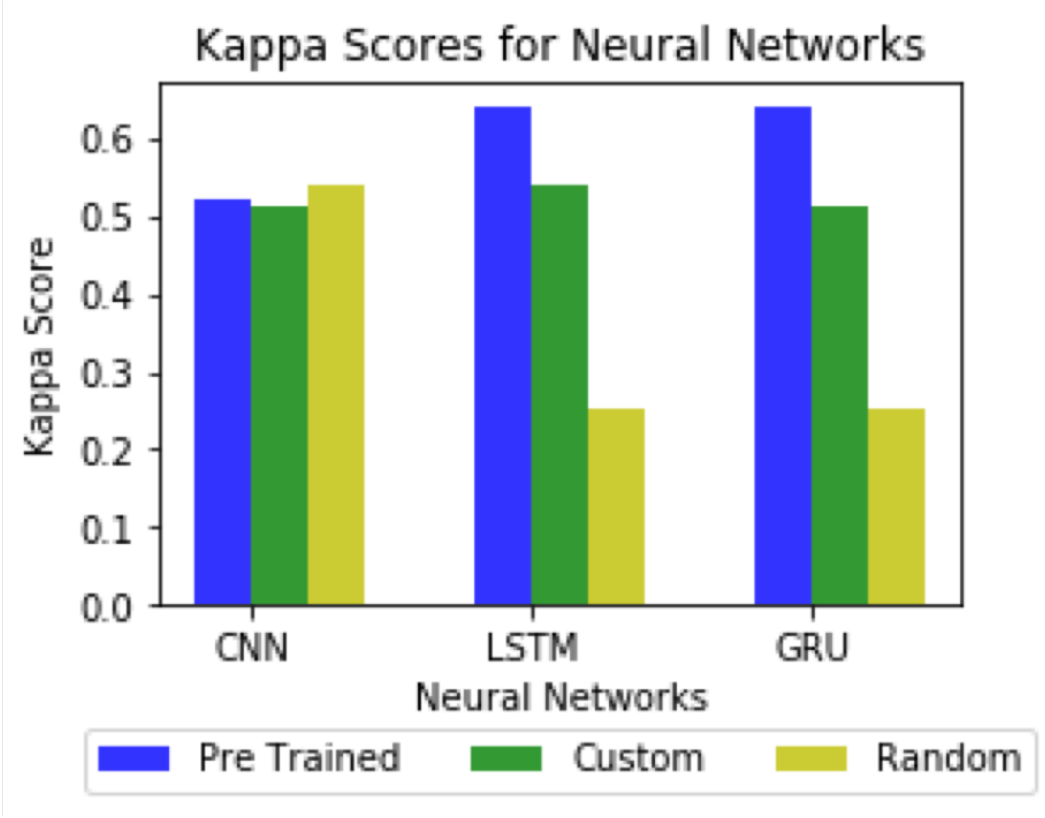


Figure 5: Kappa scores graph.

There are two main differences between GRU and LSTM. The first is that an LSTM has more parameters than a GRU network, so all the pros and cons that come with that - memory size, running time, gradient issues, etc. The other is that LSTM has a separate concept of both the output at each time step and the cell memory at each time step. For GRU, the output and the hidden state at each time step are the same. This may give the LSTM an advantage in learning some latent features of the sequence that are not directly tied to the elements of the sequence. After training both our models we were able to decipher from the results that both LSTM and GRU have given us similar results.

A big argument for CNNs is that they are fast. The most natural fit for CNNs seem to be

classification tasks, such as sentiment analysis or spam detection. Convolutional Neural Networks take advantage of local spatial coherence in the input, which allow them to have fewer weights as some parameters are shared. This process, taking the form of convolutions, makes them especially well suited to extract relevant information at a low computational cost. CNN is used more generally on other types of data that has a spatial relationship. For example, when there is an order relationship between words in a document of text. There is an ordered relationship in the time steps of a time series. Although not specifically developed for non-image data, CNNs achieve state-of-the-art results on problems such as document classification used in sentiment analysis and related problems. CNN did not perform as good as LSTM and GRU in our case. This was due to the vanishing gradient problem. The problem is that in some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value. Also, context retention was another drawback for the performance of CNN being low.

6 Conclusions

After preprocessing the data, we analysed which technique of embedding would efficiently create a vector space such that maximum tokens are mapped and the similar and correlated words are nearby and can be easily used to identify the sincerity of questions. After careful analysis we found that pre-trained Embeddings gave better results and identified that since it was trained on a much larger corpus, it was able to give better results. Negating our assumtion that word embeddings created for our particular dataset will perform better than that of the pre-trained embeddings, we realized that the pre-trained embeddings performed better. While analysing the results of the models, we found that LSTM and GRU gave about the same results but GRU took less time to train thus concluding that even though both the models are Recurrent Neural Networks, GRU took less time since it had fewer gates. After deeper analysis we were also able to ascertain that CNN has narrow context retention and diminishing gradient problem due to small vector size of the convolution and maxpooling layers and hence it gave poor results.

References

[1] A Rumor Events Detection Method Based on Deep Bidirectional GRU Neural Network

[2] Recent trends in deep learning based on Natural language processing

[3] Notes on deep learning for NLP

[4] Global vectors for word representation

[5] Application of Convolutional Neural Network (Cnn)in Microblog Text Classification

[6] Learning temporal representation of transaction amount for fraudulent transaction recognition using CNN, Stacked LSTM, and CNN-LSTM

[7] Article on detecting insincerity in texts

[8] Paper on textual aggression detection

[9] Dataset - https://www.kaggle.com/c/quora-insincere-questions-classification

[10] Paper on dynamic routing between Capsules

[11] Paper on use of Attention layer to solve LSTM problems

7 Github URL

`https://github.com/mohkar/alda_capstone`