Classification of Insincere Questions on Quora

Hao Mao (haomao@), Rekha Kumar (rekha123@), Jerry Chen (jchen98@) {stanford.edu}

Code: https://github.com/mhbkb/cs229-project

Introduction:

Quora is a popular online platform for people to pose questions and (usually) get well thought out answers to. While for the most part questions are asked in good faith, a small percentage of bad actors post questions that are insincere or problematic (for example, questions founded upon false premises, or ones that just intend to make a statement of some kind). In the interest of improving the overall community experience, a challenge then is to develop a classifier that will take as input a user generated question and automatically classify that question as sincere or insincere. To this end we experimented with a wide variety of models, including Naive Bayes, Logistic Regression, Averaged Perceptron and Recurrent Neural Networks. We also experimented with Google's pretrained language model, BERT.

Dataset and Features:

Our dataset comes from a Kaggle competition[1] and consists of approximately 1.3 million sample questions, each labeled as either insincere (positive example), or sincere (negative example), with approximately 94% of examples being sincere. Some example data points include:

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

For preprocessing, we first created a copy of the dataset, but in lowercase. We then created several variations of the lowercase dataset using different combinations of preprocessing methods. The four preprocessing methods we used were removing punctuation (and non-english characters), tokenization, stemming, and removing stop words.

1. The punctuation removal step was initially a straightforward removing all punctuation characters from the strings, but after seeing a discussion online[2] we expanded this to also include non-english characters, of which there were many in the dataset.
2. Tokenization used a regex expression to split the string by non-alphanumeric characters (the difference from removing punctuation here is mainly in contractions- removing punctuation changes "what's" to "whats", whereas tokenization changes "what's" to "what s").
3. For stemming we used the popular snowball stemmer from NLTK. This transformed words into their stems- i.e. "fishing" becomes "fish".
4. For stop words removal we similarly utilized NLTK's provided corpus to remove common english 'filler' words like "and", "an", and "the" from the dataset.

These methods were used in various combinations to create the dataset variations. Specifically, we tried using all four, excluding one at a time, just stemming, and tokenization and stemming (7 variants in total).

All 7 of these dataset variants were then converted into a word count vector for use by the Naive Bayes algorithm, and a tf-idf (a statistic combining term frequency and inverse document frequency) vector for Naive Bayes and Logistic Regression. Meanwhile, for CNN, since our model has an embedding layer which applies to world existing pretrained embeddings word2vec pre-trained vectors, we didn't do much data preprocessing except removing some punctuation that we are sure they don't exist in the pre-trained data.

Lastly, for every model that we ran with every dataset, we split this dataset 60/20/20, with 60% becoming the training set, 20% becoming the validation set, and the last 20% becoming the test set. The only exception was with BERT. Due to the slow nature of BERT and a limited cloud computing budget, we were only able to run it with a small subset of the dataset.

Methods

1. Naive Bayes

Naive Bayes is a generative learning model that takes the distribution of words given labels from the training set to compute the probability of a test set statement having a positive or negative label. The distribution from the training set is usually modified somewhat using Laplace Smoothing, which deals with unseen vocabulary in the training set. The exact equations for calculating the distributions is as follows:

$$\phi_{j|y=1} = \frac{1+\sum_{i=1}^{n} 1\{x_j^{(i)}=1 \wedge y^{(i)}=1\}}{2+\sum_{i=1}^{n} 1\{y^{(i)}=1\}}$$

$$\phi_{j|y=0} = \frac{1+\sum_{i=1}^{n} 1\{x_j^{(i)}=1 \wedge y^{(i)}=0\}}{2+\sum_{i=1}^{n} 1\{y^{(i)}=0\}}$$

$$\phi_{y} = \frac{\sum_{i=1}^{n} 1\{y^{(i)}=1\}}{n}$$

We also read that a Naive Bayes could theoretically run with fractional counts, so we also ran a variant using tf-idf scores instead of word counts.[3]

2. *Logistic Regression*

Logistic Regression is an algorithm that learns a set of weights, theta, that gets multiplied with the features and fed into the sigmoid function in order to get the hypothesis:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

The output of the sigmoid is then used to make a prediction of the label, and the goal of logistic regression is to find the weight that minimizes the log-loss.

$$\sum_{i=1}^{n} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))$$

To find the right set of weights, we run gradient descent on the training set:

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$$

3. *Averaged Perceptron*

The perceptron is a classic learning algorithm for finding a linear decision boundary for classification problems. The issue with the vanilla perceptron is that it counts later data points more than it counts the earlier data points. Averaged Perceptron is a modification of the Perceptron algorithm where we maintain a running sum of the averaged weight vector.

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(k)} \left(\boldsymbol{w}^{(k)} \cdot \hat{\boldsymbol{x}} + b^{(k)}\right)\right)$$

W,b are the weight vectors and $c^{(1)},...,c^{(K)}$ are the survival times for each of these vectors.

4. *Recurrent Neural Network*

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. Because of that, we think it may work well for this problem.averg
We built 3 layers in our RNN model (figure 3). The first layer is the embedding layer which fits the tokenized input vector to the embeddings. The second layer is the LSTM[4] (figure 1) layer which has 300x96 hidden layer. The third layer is the GRU[5] (figure 2) layer which has 192x96 hidden layer.

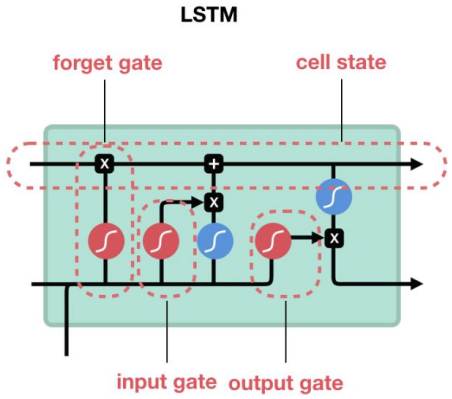


Figure 1

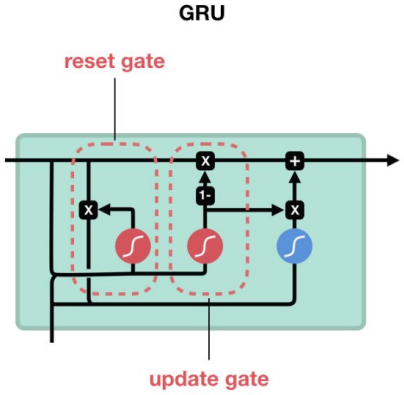


Figure 2

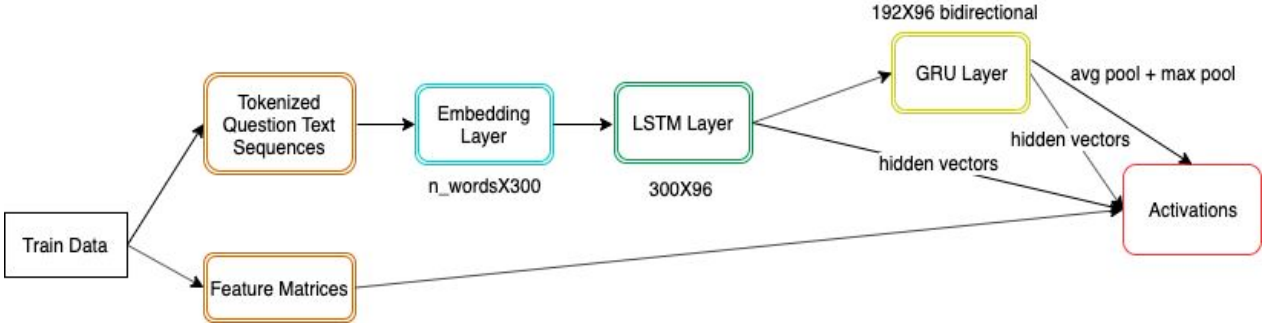
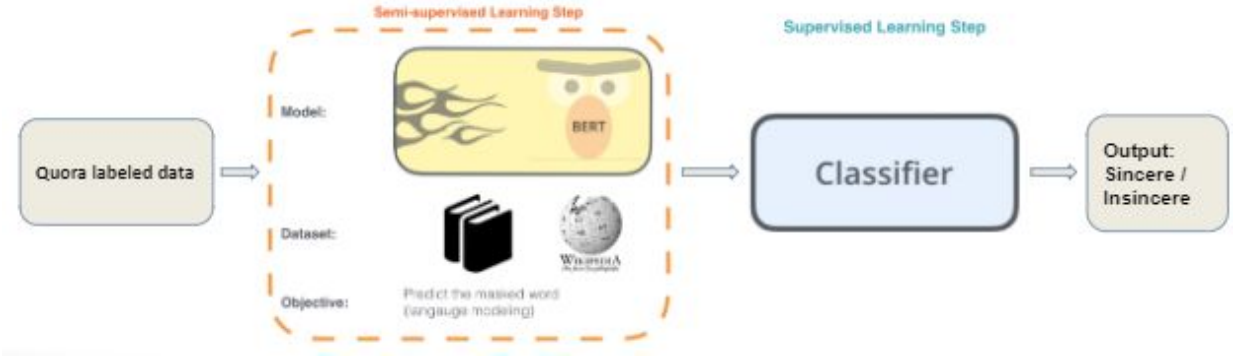


Figure 3

5. Bert based classification model:

BERT (Bidirectional Encoder Representations from Transformers) paper was published by researchers at Google AI Language. It has become a popular technique in the Machine Learning community for a wide variety of NLP tasks. BERT applies the bidirectional training of Transformer, a popular attention model, to language modeling. This is in contrast to previous efforts which looked at a text sequence either from left to right or combined left-to-right and right-to-left training. The paper's results show that a language model which is bidirectionally trained can have a deeper sense of language context and flow than single-direction language models.



BERT can be used for a wide variety of language tasks, while only adding a small layer to the core model. Classification tasks can be done by adding a classification layer on top of the Transformer output for the token.

Experiments, Results and Discussion

Metrics

Our primary metric is the F1 score. This was also the chief metric used on the Kaggle competition site, and using the F1 score allowed us to compare our model performances with other people who participated in the competition, as well as get a sense for the ideal performance to aim for. There was an issue here however; Kaggle had a seperate, unlabeled test set on which we could be judged with, but due to quirks in the submission process, it became unfeasible to run our more complicated models and get results on Kaggle. So unfortunately, we were only able to see our model performance on Kaggle's test set only with Naive Bayes and Logistic Regression. We also looked at accuracy, precision, and recall, with a secondary emphasis on recall because we decided that within the context of this problem, we would want to lower false negatives (insincere questions classified as sincere), which corresponds with better recall.

1. Naive Bayes

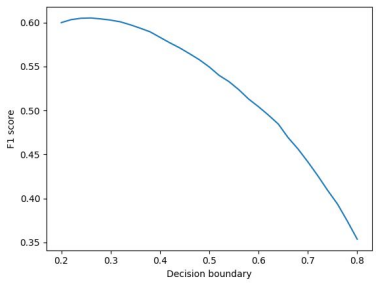
One of the first models we experimented with was the normal Naive Bayes with word counts and laplace smoothing. We ran the algorithm with all seven preprocessed dataset variations, and found pretty minimal differences in performance, with F1 scores varying from 0.52 to 0.55, and accuracies varying from 0.933 to 0.937. The best performing dataset here (based on F1 score) was the dataset that did not have stop words removed, with an F1 score of 0.55 and an accuracy of 0.936. Unfortunately, while the F1 score achieved is alright, the accuracy is slightly underperforming, as a simple classifier that simply outputted 0 for every input would achieve an accuracy of 0.938.

We also tried a variant of Naive Bayes with Tf-idf scores instead of word counts. Again we ran with all seven preprocessed dataset variations, and again we found minimal performance differences, and while accuracies improved marginally to a range of 0.939 to 0.941, the F1 scores dropped dismally to a range of 0.08 to 0.11. These models predicted very few insincere questions as insincere (low true positives), and predicted many insincere as sincere (high false negatives). The (marginally) best performing dataset here was the one where we did not run the punctuation removal step, with an accuracy of 0.941 and an F1 score of 0.11.

Lastly, with both variants above, we tried adjusting the decision boundary from 0.5 to see if that would improve results (we tried it with Logistic Regression, discussed below, and it helped there). After running with boundaries ranging from 0.2 to 0.8, we determined that 0.5 was still the best performing boundary, and we believe that this is because unlike Logistic Regression, Naive Bayes already takes into account class priors when computing prediction probabilities.

2. Logistic Regression

Similarly to Naive Bayes, the first thing we did was run Logistic Regression with all our different preprocessed datasets. We utilized scikit-learn's implementation, leaving most hyperparameters at a default, but we varied the regularization factor, C, using the values 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 100. This was chosen based on advice from CS221 which recommended tuning with values increasing approximately by a factor of 3. Surprisingly, across all datasets, the best regularization factor based on the cross validation set turned out to be 30, and using this regularization and the tokenized and stemmed dataset, we were able to achieve an F1 score of 0.546 and an accuracy of 0.953.



We further experimented with this dataset by changing the decision boundary used for predictions, testing on the cross validation set values from 0.2 to 0.8 at intervals of 0.02, as

seen on the figure to the right. We found the best decision boundary to be 0.26, and with this boundary we achieved a good F1 score of 0.603, although a slightly lower accuracy of 0.947. Additionally, our recall improved from 0.459 to 0.647, however, so we consider this to be overall one of our more successful models. We also tested this model on the Kaggle test set and achieved an F1 score of 0.59991.

3. Averaged Perceptron

Averaged Perceptron was not initially a planned model. We had access to a tool by Microsoft called MLNet which offered a number of prebuilt classification algorithms, so we decided to try them out. Surprisingly, within only 10 iterations, averaged perceptron gave us the best results out of everything we tried, with an F1 score of 0.630 and an accuracy of 0.957.

4. Recurrent Neural Network

With the basic RNN model, we tried different techniques in RNN to refine our model. We configured the LSTM and GRU layers to be bidirectional which increased the F1 score from 0.351 to 0.428. Then we switched from SGD to Adam optimization which produced a big jump in the F1 score to 0.575. We then experimented by searching for the decision boundary using the cross-validation set and found the best boundary to be 0.20, which was pretty close to the best decision boundary for logistic regression (our best guess for why this occurred is because the decision boundary depends somewhat on the class prior distribution). With this boundary, we achieved a high F1 score of 0.625. Finally, we did dropout in our model. Unfortunately, both the result and training time didn't change significantly with this change.

Based on TA feedback, we further tried some additional experiments on the dataset itself. We tried to delete some words to see if it would flip insincere predictions to sincere predictions. For example, "Why is it that every time President Donald Trump's lips move, he lies?" was detected to be insincere in CNN model which matches y-label. If we remove "Donald Trump's", it gets predicted to be a sincere question. This would imply that "Donald Trump" might be learned to be more likely part of an insincere question based on the training data. However, if we replace "Donald Trump's" with "Tom's", it gets predicted to be an insincere question which might be because Tom is a general name, and the generality of the name might also imply insincerity.

Our implementation is based on PyTorch 1.1.0 and trained with CUDA 10.0. Each epoch took about 3 minutes, and we ran 20 epoches for each trial. Including embedding load time, the total time cost for one trial was about 70 minutes.

5. BERT

Much to our misfortune, Bert was exceedingly slow in training. On our first attempt to train with the complete training set it ran for more than a day before deciding to crash without outputs.We kept burning through our cloud credits trying to run BERT (while also using the cloud on RNN), so eventually we realized that with our limited time and CPU/GPU resources, we had to run with a much smaller subset of data. In our first successful run, we used a 10k subset of the training data, with a 1k test subset, and only ran for one epoch. Our results were not particularly good, with a mediocre accuracy of 0.941 and subpar F1 score of 0.293, but we assume this was a result of lack of training time and data.

After this, since we still had a small amount of credits left, we decided to bump up our training set to a 100k subset of the training data and a 20k subset of the test data, and run for 2 epochs. This took up half a day and the remainder of our cloud credits, but we were able to achieve a significantly improved, yet still subpar result, with an accuracy of 0.947 and an F1 score of 0.366. Given this improvement, we believe that with more computing resources and time, we would definitely be able to achieve results on par or even better than the other models that we trained..

Summary of Results

	Training Loss (780k samples)	Testing Loss (260k samples)	Accuracy	F1	Precision	Recall	TP	FN	FP	TN
Baseline (always sincere)	-	-	0.938	0	0	0	0	16115	0	245110
Naive Bayes (w/ tf-idf)	0.165	0.172	0.940	0.102	0.647	0.055	969	15146	385	244725
Naive Bayes (w word counts)	0.204	0.232	0.937	0.552	0.483	0.645	10390	5725	11122	233988
Logistic with regularization	0.093	0.123	0.953	0.546	0.674	0.459	7390	8725	3570	241540

Logistic with regularization and boundary	0.093	0.123	0.947	0.603	0.565	0.647	10434	5681	8037	237073
RNN basic	0.148	0.130	0.947	0.351	0.725	0.231	3729	12386	1417	243693
RNN bidirectional	0.155	0.131	0.949	0.428	0.672	0.324	5231	10884	2559	242551
RNN bidirectional & Adam optimization	0.135	0.116	0.956	0.575	0.689	0.509	8204	7911	3707	241403
RNN bidirectional & Adam & boundary	0.126	0.135	0.951	0.625	0.591	0.662	10671	5444	7380	237730
Averaged Perceptron	-	0.180	0.957	0.630	0.678	0.588	9485	6630	4498	240616
BERT (10k train, 1k test)	-	0.152	0.941	0.293	0.667	0.188	10	54	5	930
BERT (100k training data, 20k test, 2 epochs)	-	0.195	0.947	0.366	0.722	0.245	302	932	116	18650

Conclusion and Future Work

For this project our goal was to classify Quora questions as sincere or insincere. To this end, we tried a variety of classification models and optimizations, and overall got reasonably successful results. Our best models were Averaged Perceptron, bidirectional RNN with Adam optimization and modified decision boundary, and regularized Logistic Regression with an adjusted decision boundary, with F1 scores of 0.630, 0.625, and 0.603 respectively. Overall, we also found that since our dataset was very imbalanced, modifying the decision boundary generally improved our F1 score.

If we were to continue this project in the future, there are a variety of things we would like to try. On the preprocessing side of things, we would like to try using other embedding schemes besides tf-idf, such as word2vec, and perhaps some text normalization (i.e. combining abbreviations/ alternative spellings). On the RNN side of things, there are many more features we could extract to attempt to improve our performance. And lastly, with more time and computing resources (additional powerful GPUs), we could also see what actually happens with BERT given more data and training epochs, something of interest given that BERT showed promising improvements even with very limited data and time.

Contributions:

Hao Mao: Data preprocessing, Logistic Regression, RNN implementation, GCP GPU instance setup, milestone, poster and report writeup

Rekha Kumar: Averaged perceptron, Bert, GCP GPU setup, model comparisons on MLNet, milestone, poster and report writeup

Jerry Chen: Data preprocessing bug fixing, Naive Bayes, Logistic Regression (additional development, experimentation), milestone, poster and report writeup

References:

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

[2] https://www.kaggle.com/c/quora-insincere-questions-classification/discussion/74518

[3] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

[4] https://www.bioinf.jku.at/publications/older/2604.pdf

[5] https://arxiv.org/pdf/1412.3555v1.pdf

[6] Bert - https://arxiv.org/pdf/1810.04805.pdf

[7] Bert for classification https://medium.com/swlh/a-simple-guide-on-using-bert-for-text-classification-bbf041ac8d04

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

[9] https://arxiv.org/pdf/1412.6980.pdf

[10] https://arxiv.org/pdf/1810.08606.pdf