Quora Insincere Questions Classification

Vasin Srisupavanich
vs2n19@soton.ac.uk
ID: 31300162

Yu Hong Leung
yhl1u19@soton.ac.uk
ID: 31291279

Peng Chen
pc3n19@soton.ac.uk
ID: 31142354

1 INTRODUCTION

Quora [2] is a platform that offers a convenient way for people across the world to ask questions on any topics and receive answers from others. However, among the questions and comments, some contents are malicious and not genuine for discussion, such as topics about gender, faith, and violence. These questions are called insincere questions and need to be recognized and prohibited. To alleviate this issue, Quora has created a competition on Kaggle called ***Quora Insincere Questions Classification*** [3], with the aim to encourage participants to create machine learning models to identify and flag insincere questions.

Quora provides competitors a training dataset that contains 1,225,132 sincere questions and 80,810 insincere questions, and a test set with 375,806 examples. We can see that the training set is extremely unbalanced so Kaggle requires participants to use F_1 score as metrics. In addition, aiming to measure the models' performance fairly, Kaggle constraints word embedding within the following four kinds:

- GoogleNews-vectors-negative300
- glove.840B.300d
- paragram_300_sl999
- wiki-news-300d-1M

With aim of finding higher F_1 score and learning more machine learning models, we have developed several machine learning models, ranging from simple models like logistic regression, SVM and NBSVM, to advanced models, such as LSTM with attention mechanism and state-of-the-art pre-trained BERT. In this paper, we will explain the process of developing each model, present the results, and discuss the advantages and disadvantages from each techniques. All of our implementation is done using Python. We mainly used `scikit-learn` library for developing machine learning models and `PyTorch` library for our deep learning models. The code are available on our Google Colab page (https://colab.research.google.com/drive/1bQjpu1oG_22sYaHXvHH-lfrCx7lJeA_I).

2 PREPROCESSING

Before developing the machine learning models, we first started by cleaning up the text data. Forum texts should be dirtier compared to news article because of its informality and unconstrained choice of languages and symbols. For this task, we performed the text pre-processing methods based on public kaggle kernel in [6]. There are a total of six steps, which are as follows:

(1) Punctuation and special symbols, such as '←', '×' and '§' are removed.
(2) Math equation and HTML tags are converted to [MATH EQUATION] and [url] respectively.
(3) Stop words such as a, an and the are removed.
(4) Common misspelled words are fixed (e.g. langague -> language).
(5) Common contraction words are expanded (e.g. aren't -> are not).
(6) Lemmatization, the process of changing to root form, is performed using WordNet (e.g. playing -> play).

3 FEATURES

3.1 Bag of Words

The bag-of-words (BOW) model represents each document (each question) as a vector of term frequency of the corpus vocabularies. A term can be a single word, but it can also be extended to n-gram models and count their frequencies in a document. In our model, we have chosen the `sklearn` BOW implementation to include unigram, bigram, and trigram, and exclude stopwords in English. However, the feature vector will suffer from high dimensionality and sparsity if we include every unigram/bigram/trigram in the corpus as a feature.

Therefore, when building the features/vocabularies for the model, we have also excluded terms that appear less than 3 times or in the top 10% frequency as they may be too rare or too common to contain useful information for classification. The model will only keep the top 50000 features ordered by term frequency, which thus represents each question with a 50000-d vector.

3.2 TF-IDF

TF-IDF makes use of term frequency and inverse document frequency to determine which terms are valuable as a feature. Similar to bag-of-words, TF-IDF also makes use of information of term frequency in each document to find frequent words. However, words like "the" that appear too frequently in every document are not useful. Hence, inverse document frequency helps us weigh down and discount words that are common across all documents so that important terms can be found as useful feature. Similarly, we have chosen the same hyperparameters as in bag-of-words model so that only the top 50000 terms are kept as features in TF-IDF.

3.3 Word Embedding

Pre-trained word embeddings are utilised to improve neural network models. Trained with a large corpus, they represent words in vectors so that the numeric representation can capture the syntactic and semantic properties of words. Words that share similar meaning or context would be close in the vector space. In this project, we mainly use the pre-trained GloVe embedding (300 dimensions for 2.2 million vocabularies), which trained on the aggregated global word-word co-occurrence statistics. We also only keep at most 120000 words that appear in the training corpus to improve the training speed for neural network models.

4 MODELS

4.1 Logistic Regression

To get an idea of how difficult the task is, we first implemented a baseline model using a simple logistic regression. First, the input questions were pre-processed as mentioned in section 2, including the process of removing stopwords, symbols, and fixing contractions. For this model, we use both bag of words and TF-IDF as input features. We used `scikit-learn`'s implementation of the logistic regression model with L2 regularization. To tune the regularization coefficient, we performed grid search cross validation with different sets of regularization coefficient. Then, we chose the one that gives the best F_1 score on the training data.

Since our dataset is unbalanced towards one class (sincere question), using the default 0.5 decision threshold gave us a very poor result. To improve the result, we searched for the decision threshold which gives the best F_1 score in the validation set. Finally, the final F_1 score from the test set using bag of words features is 0.575, and using TF-IDF features is 0.590.

4.2 SVM with Naive Bayes features (NBSVM)

The insincere questions analysis task could not be solved only by single baseline models but also by their combinations. Sida Wang and Chris Manning [7] proposed a NBSVM model in which it combines the output of Naive Bayes' classifier's features and feature count vectors, such as N-grams and TF-IDF and then feeds them into a SVM.

Linear classifiers could be summarized in such form for data k.

$$y^{(k)} = sign(\mathbf{w}^T \mathbf{x}^{(k)} + b) \tag{1}$$

Naive Baye's classifier makes decision according to the posterior probability, thus it's weight $\mathbf{w}$ could be expressed in a log count ratio:

$$\mathbf{r} = \log\left(\frac{\mathbf{p}/\|\mathbf{p}\|_1}{\mathbf{q}/\|\mathbf{q}\|_1}\right) \tag{2}$$

where $\mathbf{p} = \alpha + \sum_{i:y^{(i)}=1} \mathbf{f}^{(i)}$ and $\mathbf{q} = \alpha + \sum_{i:y^{(i)}=0} \mathbf{f}^{(i)}$. α is smoothing parameter to avoid $\log(0)$ and $\log(\infty)$ and $\mathbf{f}^{(i)}$ is the feature count vectors. This ratio could also be seen as the weight of evidence (WOE) to help transform a binary value count vector into a continuous weight.

Then to combine NB and SVM, the ratio $\mathbf{r}$ is multiplied element-wise with the feature count vectors $\mathbf{f}^{(i)}$. As we understand, this process is to set apart the data of two classes more because the features that have insincerely characters will be weighted by a negative number, while sincerely characters by a large positive number.

Apart from combining the input with NB features, the authors also proposed a method that could compromise between NB and SVM. They changed the weight $\mathbf{w}$ into following form:

$$\mathbf{w}' = (1-\beta)\bar{w} + \beta\mathbf{w} \tag{3}$$

where $\bar{w} = \|\mathbf{w}\|_1/|V|$ is the mean magnitude of w, $\beta \in [0,1]$ is the interpolation parameter, V is the dimension of the input feature. The paper pointed out that this interpolation could be seen as a form of regularisation. If β is small, WOE term $\mathbf{r}$ will influence more in the SVM's hinge loss: $\max\left(0, 1 - y^{(i)}(\mathbf{w}'^T \mathbf{r} \odot \mathbf{f}^{(i)} + b)\right)$

In practice, we implemented a NBSVM model by using self-defined log count ratio and `sklearn LinearSVC`. As before, to tune hyperparameters we applied grid search to find the best C at around 2, and then we set the argument `'dual'='True'` to tell SVM solve a dual problem.

4.3 XGBoost

XGBoost is a gradient boosting method that is very efficient and has won Kaggle competitions before. By combining the weak learners (i.e. shallow trees), a strong learner can then be found to perform classification and avoid overfitting. New weak learner is added by learning the residual error of the prediction, which also minises the loss using gradient descent. In XGBoost, many hyperparameters can be set to improve the model's performance such as early stopping, learning rate, regularisation terms, and max depth of generated trees.

For this problem, we need to create features from the dataset to make use of decision tree boosting. We have chosen TF-IDF as input features, together with 13 extracted features in the original unprocessed questions, such as capitalisation, punctuation, word count, and word length. Then, the model is boosted for 500 rounds with early stopping for 10 rounds if there is no improvement on AUC for the validation set. Finally, the fitted model also goes through decision threshold searching to improve the F1 score on insincere questions. The final performance on the test set is: precision 0.547 and recall 0.657. The resulting tree contains around 90 nodes with a depth of 6 layers in Figure 1.

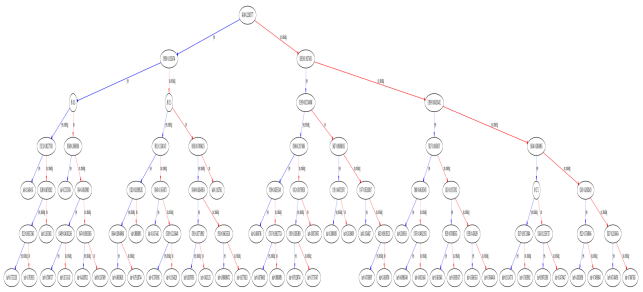

Figure 1: Result of XGBoost

4.4 TextCNN

CNN has been proved to perform well with pretrained word embedding in text classification in [5]. The architecture of textCNN for processing sentences is shown in Figure 2.

Same as the image processing task, the role of convolutional layer is to extract features. Firstly, the input sentences are projected into an embedding space, in our case GloVe-300d. Then a filter of size **[n, emb_dim]** will slide down the whole 'text image' and produce a feature map. It is important to note that for example if the filter size equals to **[2, emb_dim]**, it actually constructs bi-grams as features in the sentence. It is also worth mentioning that we can choose different filter size and number of filters. For instance, if we choose $n = [2, 3, 4]$ and `num_filter=100`, it means that for each filter size, there are 100 different filters, thus producing 300 feature maps in this example.

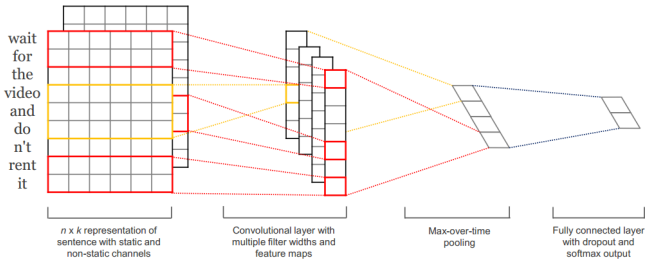

Figure 2: The outline of textCNN

After extracting features, each feature map is passed to an adaptive max pool layer to find the most important feature and decrease the number of parameters at the same time. Since the feature maps have different length due to varying window size and sentence length, max pool operation also helps to deal with variable sentence lengths.

After max pooling, each feature map outputs an value. Also with the above example, we have 300 values now. These values are then concatenated into a vector and passed into a fully connected layer with softmax to get the prediction binary result. For regularisation, dropout is applied to the penultimate layer with a constraint on the weight vectors by L2 norm. In the paper, the author implemented dropout by applying a mask $\mathbf{r}$ on the concatenated features by element-wise multiplication. The elements in the mask follow the Bernoulli distribution with probability p. After that, gradients could only passed unmask routines during training. During the test phase, the learned weights are then weighed by the probability p and the model is tested without dropout.

In our model, the sentences are padding to same length of 72 in the preprocessing. Then the sentences are changed into index form and passed into embedding layer to produce 300 dimension vectors according to Glove. And then, we chose filter size equals to [3,4,5] which represents 3 different word models and 36 filters each to extract semantic features. Finally, we set the fully connected layer a dropout at 0.2 for regularisation and fast training process.

4.5 Plain LSTM

Long Short Term Memory (LSTM) is a type of recurrent neural network (RNN) that are useful for processing sequential data. Unlike traditional RNN, LSTM alleviates the problem of vanishing or exploding gradients, and contains gates that can control the learning behaviour of "long term memory". Our implementation here serves as a baseline RNN for us to compare the performance of other deep learning models later.

First, the original question text is tokenised and embedded using the GloVe embedding, which is then put in the embedding layer of the LSTM. Then, for the LSTM architecture, only single direction is chosen with a hidden size of 128. The final output of LSTM is then passed to a dense layer with ReLU activation, followed by another dense layer to produce the single output prediction with sigmoid function. In addition, dropout layer is chosen in both the LSTM and the ReLU dense layer to avoid overfitting. We have chosen the Adam optimiser with default settings and the binary cross entropy as loss function. Even though the training process is relatively fast, the model is only trained for 10 epochs as the validation loss starts increasing. The model achieves a final F_1 score of 0.656 in the testing set.

4.6 Bidirectional LSTM with Attention Mechanism

Next, we tried to improve our score by incorporating an attention mechanism in LSTM networks. The idea is that the attention mechanism would allow the model to focus on certain words, rather than the whole sentence, as we know that some words carry more weight when deciding whether the questions are insincere or not. The architecture of this model is implemented as follow:

(1) The pre-trained GloVe provided from Kaggle is used as the weight in the embedding layer.
(2) Word embedding features are feeded into the 2 layers bidirectional LSTMs.
(3) Attention mechanism is applied to the output (hidden states) of each of the LSTMs layer.
(4) Max pooling and average pooling is applied to the output of the second layer of the LSTM.
(5) Features from (3) and (4) are concatenated and feeded into the fully connected layer for classification.

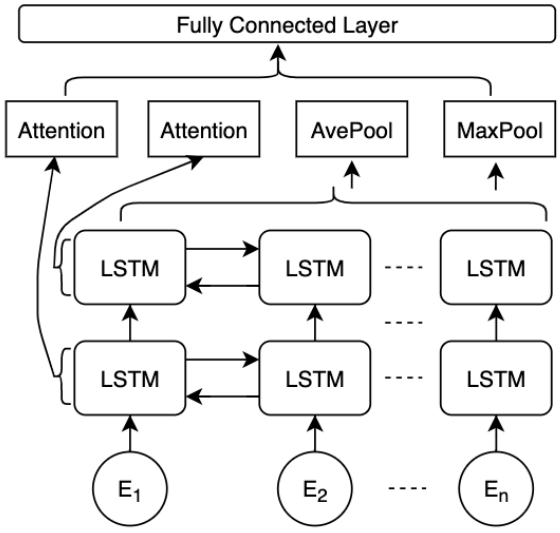

Figure 3: Architecture of bidirectional LSTM with attention mechanism

Figure 3 illustrates the overall architecture of this model. The training process is optimized using Adam with binary cross entropy as a loss function. To prevent the model from overfitting, we also applied dropout with p=0.1, and performed early stopping. We observed that the validation loss started to increase after 5 epochs while the training loss still keeps decreasing, so we decided to train the model for only 5 epochs. We also tuned the learning rate by using k-fold cross-validation. Using Google Colab's GPU, the whole training and inference process took around 10 minutes. After finding the best decision threshold, we were able to achieve an f1 score of 0.673 on the test set.

4.7 Using Pretrained BERT

Even though, the competition does not allow the use of pre-trained BERT models, we were interested to see the result from using them, as they are the current state-of-the-art model for NLP applications. BERT models are based on the newly introduced architecture called Transformers [4], which take away the recurrent units, and replace them with self-attention mechanism and positional encoding. The advantage of the transformer models is that the sentences are processed as a whole, rather than word by word in the recurrent neural networks. The idea is that self attention and positional encoding would enable the transformers to capture long range dependency better than traditional RNNs, as the hidden state of the RNN's unit depends on the previous input sequence. In addition, transformer can be trained in parallel, since there is no dependency between each input sequence. This enables BERT models to be trained on a large scale, with hundreds of millions of parameters, on a large unlabeled dataset. Therefore, the knowledge of the language learned on a large dataset would be well suited in downstream tasks, such as text classification.

Our implementation is based on Hugging Face library [1], which provided us with a simple way to use the pre-trained weight, and pre-defined BERT architecture with linear classification layer. For this task, we chose ROBERTA, a new BERT variant developed by Facebook, since they have less parameters (125M) and can be faster to train compare to the original BERT model. We fine-tuned the model on the ECS's GPU server using batch size of 16 and Adam optimizer with learning rate of 1e-4. Because of the high number of parameters, one epoch of training takes roughly around 7 hours. After training for 5 epochs, we were able to achieve the highest F1 score (0.698) from all of the models we implemented.

5 DISCUSSION

Table 1 summarizes the precision, recall and F_1 score from 7 different models. We can see that the F_1 score from all the models are within the range of 0.6 and 0.7, which are not that far apart. The baseline model of logistic regression with TF-IDF features performed the worst as expected with 0.590 F_1 score. As shown in the table, non deep learning based methods (Logistic regression, NBSVM and XGBoost) have similar performances even though their precision and recall differs. Interestingly, XGBoost achieved a higher recall than any other models we have implemented, which is perhaps due to the precision-recall trade-off. Also, by using deep learning models (LSTM, TextCNN), the F_1 score increased by at least 3% from traditional machine learning based method. Finally, by using recent advanced architectures, such as attention mechanism and BERT, the F_1 score further improves and approaches 0.7, which is getting close to the score of the winner of the competition (0.713).

However, we can see that the recall from nearly all models are relatively low compared to the precision. This means that from all the insincere questions, the model would be able to detect only about 60% of the time, which is still low and not very effective. This is due to the fact that this dataset is highly imbalanced (only 6% of all the questions are insincere). To alleviate this issue, we would need to collect more insincere questions, or find ways to augment or resample the data.

Models	*Precision*	*Recall*	*F1 Score*
Logistic Regression	0.659	0.537	0.590
SVM with NB features	0.668	0.529	0.592
XGBoost	0.547	0.657	0.597
TextCNN	0.670	0.584	0.624
LSTM	0.700	0.618	0.656
LSTM with Attention	0.729	0.625	0.673
Fine-tuning BERT	0.768	0.639	0.698

Table 1: Summary of the score from each model

6 CONCLUSION

In this paper, to classify insincere questions in Quora, we implemented three models and four deep learning networks. We used the results of simple models as a baseline and then achieved higher F_1 score on deep models with the word embedding required by Kaggle. Among them, XGBoost and fine-tuned BERT model got the best result in simple models and deep models respectively. Actually, in the Kaggle leaderboard, there are 829 teams having reached F_1 score over 0.7 and while our team's performance is only in the middle. However, we have gained a lot more knowledge on how to preprocess text data, how to build the word models and load pretrained word embedding, how the models architectures look like, how to implement those models by `sklearn` and `PyTorch`, and how to tune the hyperparameters in each model accordingly.

To go further, we have also considered some future improvements to the models. Firstly, some leading teams used the combination of two or more required word embeddings to get better word representations. For example, the team in the 2nd place on the leaderboard used all four provided word embeddings to train one with 668 dimension. Some teams also computed the average or weighted mean for several of the embeddings. More surprisingly, four kinds of word embeddings are concatenated to form a 1200 dimensional embedding space. In spite of the high dimensionality that makes the training even slower, this method got a good result and converged only after two epochs. Secondly, to improve computation speed, some teams applied dynamic padding sequence in a batch rather than padding it to a fixed length. Choosing a proper dropout ratio and right kind of dropout (e.g. common dropout rather than dropout2d) could also help performance. Finally, there are practical points we can further consider, such as how to utilise the result of k-fold cross-validation to find the best threshold.

REFERENCES

[1] [n.d.]. Hugging Face. https://huggingface.co/.
[2] [n.d.]. Quora. https://www.quora.com/.
[3] [n.d.]. Quora Insincere Questions Classification. https://www.kaggle.com/c/quora-insincere-questions-classification.
[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Association for Computational Linguistics, 4171–4186.
[5] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).
[6] Theo Viel. [n.d.]. Improve your Score with some Text Preprocessing. https://www.kaggle.com/theoviel/improve-your-score-with-some-text-preprocessing.
[7] Sida Wang and Christopher D Manning. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th annual meeting of the association for computational linguistics: Short papers-volume 2*. Association for Computational Linguistics, 90–94.