UPPSALA
UNIVERSITET

Report for Natural Computation Methods for Machine Learning

Project

Quora Insincere Questions Classification

Group 10

Vishnu Sharma   Nikhil Karthik Punnam
Phalguni Chopra

May 23, 2019

Abstract

The project we have chosen to do is the Quora Insincere Questions Classification [1]. The goal is to classify the questions into binary categories, a sincere question or an insincere question. We explored a few of the existing text preprocessing techniques in conjunction with deep learning architecture like bi-directional Long Short Term Memory (LSTM) to classify the questions. Our architecture measured an F1-score of 0.57, which by no means is near the score of state of the art models but our main focus was to explore text processing techniques and different ways to to deal with common issues in textual data and see how our model is affected by them.

1 Introduction

Text mining is frequently being used as organizations recognize the unused information contained in the text. Social media platforms such as Facebook and Twitter, have been used effectively by organizations to uncover positive and negative trends that when identified through text mining, can be used to leverage the positive trends and provide corrective action to check any negative comments.

Quora[1] is a platform that allows people to connect with each other, exchange their thoughts and learn from each other. It provides a medium for people to ask questions on a huge variety of topics to a wide audience and contribute in writing answers to them. As such there are people not exactly looking for answers more than to make a statement in such a platform. The project aims to weed out such insincere questions.

The definition of an insincere question as given on the competition site [1] is as follows: '*An insincere question is defined as a question intended to make a statement rather than look for helpful answers.*' The competition site [1] defines some characteristics that can signify that a question is insincere:

- *Has a non-neutral tone*
  - *Has an exaggerated tone to underscore a point about a group of people*
  - *Is rhetorical and meant to imply a statement about a group of people*
- *Is disparaging or inflammatory*
  - *Suggests a discriminatory idea against a protected class of people, or seeks confirmation of a stereotype*
  - *Makes disparaging attacks/insults against a specific person or group of people*

[1] www.quora.com

  - *Based on an outlandish premise about a group of people*
  - *Disparages against a characteristic that is not fixable and not measurable*
- *Isn't grounded in reality*
  - *Based on false information, or contains absurd assumptions*
  - *Uses sexual content (incest, bestiality, pedophilia) for shock value, and not to seek genuine answers*

1.1 Data

The training dataset present on Kaggle is split into three fields

- qid - unique question identifier
- question_text - Quora question text
- target - a question labeled "insincere" has a value of 1, otherwise 0

The qid is a long integer and the question_text field has only text and the target takes values from {0, 1} set. For this project we only use the question_text field and qid is discarded. The text in the question_text field is taken in as input as sequence of words to be fed into our model.
An example of each sincere and insincere question would be,

qid	question_text	target
00017146167b4072ae5f	If lightsabers are created by individual wielders, does each saber have unique powers/abilities?	0
00013ceca3f624b09f42	Which babies are more sweeter to their parents? Dark skin babies or light skin babies?	1

2 Theory

2.1 Recurrent Neural Network (RNN)

RNN is a variant of the Artificial Neural Network (ANN) where the nodes in the hidden layer are connected in such a way that these nodes store data from the previous time step which help in utilizing the connection between the input sequences [2].

2.2 Long-Short Term Memory Networks (LSTM)

LSTM is a type of RNN where the network remembers long sequence of input and tries to find the important sequences to remember in the whole input. This variant works better than RNN because it solves the problem of vanishing gradient where the old data is forgotten completely based on how far it is from the current instance of input [2]. It does not consist of just one activation function to the input but rather an input data inside these neurons goes through a series of gates. Two major checkpoints inside LSTM unit is the forget and the write gate. Write gate helps the RNN decide which part of the information to be retained or written. Forget gate helps RNN forget the non-important information and scrap it and produce a filter through which only the most important information can pass through, which in turn reduces the load of long dependencies and helps remember the previous context.

2.3 Bi-Directional LSTM

Bidirectional LSTMs are an extension of traditional LSTMs that can improve model performance on sequence classification problems [3]. The principle of Bidirectional LSTMs is to connect two hidden layers running in opposite directions, one for positive time direction (forward states), and another for negative time direction (backward states) to a single output. In our case, “time” refers to each word in the input sequence. The advantage bidirectional LSTM offers over traditional LSTM is that the neural network has access to information from the past and the future thus helping it gain context from both directions without the use of more sophisticated LSTM layers.

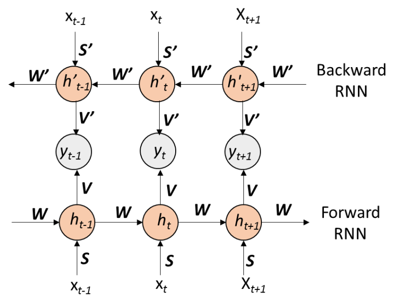


Figure 1: Working of Bi-directional LSTM
Source: https://subscription.packtpub.com/book/big_data_and_business_intelligence /9781787121089/6/ch06lvl1sec69/setting-up-a-bidirectional-rnn-model

In the picture, “w” is the weights that are updated from taking the sequence in its right order (“s”) and “w’” are the weights that are updated from the taking the sequence in reverse (“s’”). “h” and “h’” are the hidden nodes for forward and backward sequences respectively and “x” and “y” are input and output nodes. From the picture above, the forward RNN is responsible for handling the input data in the usual direction and backward RNN is responsible for the reverse sequence of the input data. Both, the forward and backward LSTM combine to give output for that particular time-step (t-1, t, t+1) or word-step(our case).

2.4 Lemmatization

Lemmatization is the process of reducing the words to their root words i.e. “asked”, “asking” will be reduced to “ask”. This mostly helps in reducing the number of unique words but keeping the intent same [4]. This can be done using the NLTK package which is readily available in python [5].

2.5 STOP Words

While working with textual data, there will always be words that occur most frequently but do not contribute significantly to the essence of the text [4]. Such words are known as STOP words. Removing such words helps in reducing the time spent on indexing the words and helps in concentrating on words that actually contribute to the meaning. This can be done using the NLTK package which is readily available in python. [5]

2.6 Contractions

Contractions are shortened words mostly present in spoken vocabulary and ultimately adopted to written. In English, an example of contraction of phrase “I have” would be “I’ve”. We expand such contractions so that we have some sort of standardization of text [6]. A typical way of dealing with contractions is to make a table of all known contractions and expand all the current contractions we found in the text corpus via the given table.

2.7 Word Embeddings

Word embeddings is a natural language modelling technique that turns text into vector representations(made of numbers) of a word. This transformation is necessary because machine learning algorithms require their input to be numbers since the models don’t work on strings of plain text. Embeddings map words or phrases from a vocabulary to a corresponding vector of real numbers. Word embeddings are usually used for following properties:

- Dimensionality Reduction: Usually, when dealing with large corpus of textual data, the dimensionality of vectors formed is very large and word embedding aims to create a vector representation of them in a much lower dimensional space [7]. This is achieved effectively by combining PCA based dimensionality reduction with a post-processing algorithm, to construct word embeddings of lower dimensions.[8]

- Contextual Similarity: For a language model to be able to predict the meaning of text, it needs to be aware of the contextual similarity of words.[9] A measure of similarity of words can be achieved by taking cosine of the word vectors. The standard comparison metric is cosine similarity, which is equivalent to dot product if the vectors are normalized. Other metrics include Euclidean distance, the less-known Tanimoto similarity, which is similar to cosine similarity but with a different normalization factor.[10]

2.8 Class Imbalance

Class imbalance refers to the phenomenon where some classes (labels) of a dataset have more samples than others. This is a problem because the machine learning algorithms will tend to focus on the classification of the samples that are over represented while ignoring or misclassifying the underrepresented samples. In our dataset, we have only 80810 samples of insincere questions and 1225312 samples of sincere questions. The classifier will tend to classify insincere questions as sincere ones. There are several methods to overcome this problem:

- Undersampling- Under sampling is a method where we try to extract samples from the main dataset, in a sense to balance the dataset where the no. of target classes have approximately equal no of samples. We remove some of the majority class samples from the dataset in order to balance the majority and minority class samples, so it has less effect on the machine learning algorithm. However, we could risk discarding useful information as some of the majority class instances might be more representative and we run a risk of removing these instances. [11]
- Oversampling- Instead of removing samples of majority classes, we add more samples of the minority class by replication, so it has more effect on the machine learning algorithm [11]. However, just duplicating the minority classes could lead the classifier to overfit a few examples.
- SMOTE - An over-sampling approach in which the minority class is over-sampled by creating "synthetic" examples rather than by over-sampling with replication [12]. SMOTE iterates over the existing, real minority class instances. At each iteration, it chooses one of the K closest minority class neighbours and synthesizes a new minority instance somewhere between the minority instance and that neighbour. Depending upon the amount of over-sampling required, neighbors from the k nearest neighbors are randomly chosen.

2.9 Performance Measures

Accuracy is not the metric to use when working with an imbalanced dataset. It is very misleading. There are other metrics that give a better story about the performance of the model when working with imbalanced classes such as: Precision, Recall and F1 Score.

- Precision: Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.
- Recall: Recall is the ratio of correctly predicted positive observations to the all observations in sample test data.
- F1-Score: F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. It is used as a measure between the precision and the recall and gives a sense of how balanced the model is in predicting the right labels in whole of the sample.

The formula for F1-score is given by $F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$
Intuitively F1 score is not as easy to understand as accuracy, but it is usually more useful than accuracy, especially if you have an uneven class distribution.

3 Previous Work

Many existing models [13] on Kaggle were built using Bidirectional Gated Recurrent Unit (GRU) with and without any pre-trained embeddings provided in the dataset. First, the training dataset was split into training and validation samples, with training set holding 90% of the data and the validation set holding the remaining 10% data.
For pre-processing, the model filled up the missing values in the text column with “na”. The text column was tokenized and converted into vector sequences. Next, the sequence was padded as needed; if the number of words in the text were greater than the maximum length of the text, then it was truncated to maximum length or if the number of words in the text was less than maximum length, zeros were added for the remaining values.[14]

3.1 Model Architecture

There were two models built and their performances were compared to each other. The models are stated as follows:

- Model 1: Without using pre-trained embeddings- The Bidirectional Gated Recurrent Unit (GRU) model was trained using 1 input layer with 128 neurons, embedding of size 300, maximum number of words in a question to be 100 (maximum length), dropout 0.1 and sigmoid as the activation function.

- Model 2: Using pre-trained embeddings- The same baseline Bidirectional GRU model (Model 1) was rebuilt using three pre-trained embeddings, namely- Glove, Wiki News FastText and Paragram.

Finally the models were compiled using binary cross entropy and “adam” optimizer. The models carried out validation set predictions and test set predictions and calculated the F-1 score.

3.2 Observations

- The pre-trained embeddings model (Model 2) gave better results as compared to non pre-trained embeddings model (Model 1).
- The performance of the different pre-trained embeddings used in Model 2 were almost similar to each other.[14]

4 Proposed Work

We have tried a bi-directional LSTM with both under sampling and Synthetic Minority Over-sampling (SMOTE) for our experiments.

4.1 Pre-Processing

First, we preprocess the data which involves converting the input to lower case, fixing common contractions, removing punctuation, some common stop words and numbers and then lemmatizing the text.
After preprocessing, we observe that out of 1.3 million sentences, 1.2 million sentences are composed of 10 or less words. Hence, we decided that each input to the model should have 10 words in it. If a sentence has less words, then we pad it with appropriate number of padding text such that the sentence has 10 words otherwise, if a sentence has more than 10 words then we take first 10 words of the sentence under consideration.
Next, we build a dictionary item over this data, which holds all the unique words in our data as the key and a unique integer as its value. Also, we add an extra item to this dictionary as “unk” that describes the words that are not present in the training set, but present in the test data.
We split the input dataset into training, validation and test subset to evaluate our model. For this, we first create a test set which contains 10000 samples from the dataset chosen randomly. From the remaining data, we create training and

validation set. Our training set holds 80% of the data and the validation set has the remaining 20% data.

Since, we have a class imbalance problem here, we overcome this by using some sampling. For our experiments, we apply SMOTE and undersampling over the training set and compare the performance of the models trained on the data thus obtained. Our training set initially had 64810 insincere questions and 979762 sincere questions. SMOTE balances the uneven counts and generates dataset with same number of insincere and sincere questions i.e. 979762 samples of each class. SMOTE is proven to be beneficial for synthetic text generation if the input dimensions are not very high. In our case, we have 10-dimensional input data which benefits SMOTE. Similarly, undersampling creates a dataset that comprises of 64810 samples of each class. Using PyTorch Dataloader, we create batches of size 128 for training, test and validation datasets. Now that our data is ready to be fed to the model, we create a basic LSTM model, with two fully connected layer stacked over it. and train it for 20 epochs.

4.2 Model Architecture

Our model comprises of 2 Layers of bidirectional LSTMs, each with 512 neurons. We use an embedding layer, which has rows as the number of items in the dictionary created earlier and 300 columns. Further, we use a dropout of 0.5 in the final layer. We average out the forward and backward feed results from the LSTM and then feed it to a fully connected layer which has batch normalization. Batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. It improves the stability of a neural network. Finally, we apply the sigmoid activation function over the output from previous step which in return gives us the binary text classification for the given input.

4.3 Finetuning the parameters

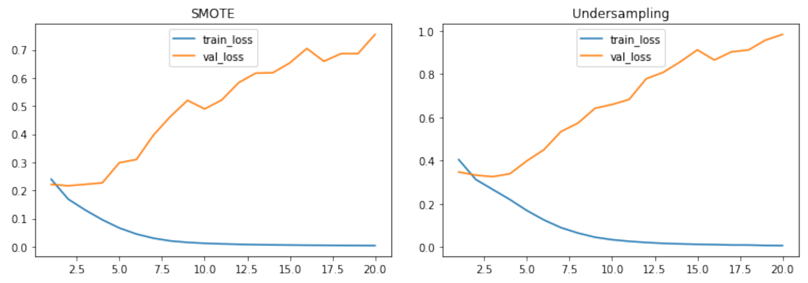
We investigate the following parameters to check the performance of our model:

- batch size: 64, 128, 256, 512
- optimizers: adam, sgd, adamax, rmsprop, adagrad [15]
- learning rate : 0.1, 0.001, 0.0001
- activation function : softmax, sigmoid, relu, tanh [16]
- dropout : 0.2,0.4,0.5,0.6,0.8 [17]
- number of layers for LSTM : 1,2,3,4

For this, we train the model using each of these parameters and choose the parameters for which the model performs the best over validation set.

5 Results and Future Work

We got the best results with batch size- 128, learning rate- 0.001, dropout- 0.5, num_layers- 2, activation function- sigmoid, optimizer- adam. The model trained over undersampled data performs poorly over the test data when compared with the model trained over SMOTE sampled data. We were able to achieve 0.95 f1-score performance for majority class and 0.57 f1-score performance for minority class i.e. insincere questions on the test data using these parameters, where the input data fed to the model is sampled used SMOTE. For the model trained over undersampled data, we get an f1-score of 0.93 for the majority class 0.48 for the minority class.



Whereas, the existing Model 1 got a F1-score of 0.65 without using any embeddings with activation function- sigmoid, optimizer-adam and metrics- accuracy and the best F1-score of 0.67 using Glove embedding in Model 2. The existing model performed better than our proposed model as it used Gated Recurrent Unit (GRU) which performs better than Long short-term memory (LSTM) used in our model, as GRU is easy to modify and doesn't require memory units, so it trains faster and is much more efficient than LSTMs.

An extension or improvement of our work would be to leverage an attention model to identify which parts of the sequences are holding high information that can help predict the correct class. Also, models using Gated Recurrent Unit (GRU) have shown promising results in identifying the classes with high accuracy and can be used in combination with attention mechanism to achieve better results.

References

[1] Kaggle. Quora insincere questions classification, 2018.

[2] Alex Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *CoRR*, abs/1808.03314, 2018.

[3] Mike Schuster, Kuldip K. Paliwal, and A. General. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 1997.

[4] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, New York, NY, USA, 2008.

[5] Edward Loper and Steven Bird. Nltk: The natural language toolkit. pages 63–70, 2002.

[6] Dipanjan Sarkar. *Text Analytics with Python: A Practical Real-World Approach to Gaining Actionable Insights from Your Data.* Apress, Berkely, CA, USA, 1st edition, 2016.

[7] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. pages 1532–1543, 2014.

[8] Vikas Raunak. Effective dimensionality reduction for word embeddings. *CoRR*, abs/1708.03629, 2017.

[9] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[10] Jon Ezeiza Alvarez. A review of word embedding and document similarity algorithms applied to academic text, October 2017.

[11] Ying Liu, Han Tong Loh, and Aixin Sun. Imbalanced text classification: A term weighting approach. *Expert Syst. Appl.*, 36(1):690–701, January 2009.

[12] Kevin W. Bowyer, Nitesh V. Chawla, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813, 2011.

[13] Kaggle. Gru + capsule, 2018.

[14] Kaggle. A look at different embeddings, 2018.

[15] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[16] Steffen Eger, Paul Youssef, and Iryna Gurevych. Is it time to swish? comparing deep learning activation functions across NLP tasks. pages 4415–4424, October-November 2018.

[17] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.