Insincere Question Detection

Benjamin Geyer
George Mason University
bgeyer3@masonlive.gmu.edu

Phillip Pascal
George Mason University
ppascal2@gmu.edu

Abstract:

Toxic internet comments plague social media sites of the internet today. This project will examine toxic / insincere questions as provided by the popular question hosting site Quora. Throughout the research and experiments, there will be an evaluation of several different models. In addition, several preprocessing methodologies will be examined in order to determine the most effective method for preparing the data. The classification model that performed the best comes in the form of neural networks. Hyperparameter tuning on the neural network will be explored in combination with the failures of the other classifier models. The causes of the shortcomings of the many classifiers that were chosen for testing will be analyzed. Future improvements to the model and preprocessing are also proposed.

• Introduction:

A current issue with modern internet websites is dealing with user content that is malicious. This form of malicious content can become especially prevalent on websites with forums dedicated to asking and answering questions. For this project, the focus was on training a classifier to identify if a question was disingenuous in nature. By disingenuous, this means questions that aren't asking for a real answer or ones that are based on lies; this is generally done to elicit a negative response. The dataset used comes from Kaggle [1-1] and was posted by a popular internet site, Quora, which is designed to have users ask and answer questions. This dataset presents several interesting challenges: the most obvious being that the data is text-based and it's made up of over a million entries. Another major issue comes from the fact that the dataset is unbalanced, with only a small percentage of the data being labeled as toxic.

One detail to note is that the project idea is not the same one from the project proposal. The original project, modeling human-like browsing-behavior on previously unvisited sites, was overly ambitious to say the least. That project required a website crawler to be developed, a scraper in order to record users' data for a dataset, a developed functional classifier model, and a validation metric that made sense and was usable by the model. A fully-functional website crawler was developed in javascript using a Node library called Puppeteer (the code for this can be found in the src folder). However, gathering sufficient useful data proved to be a costly task that far outweighed its benefit. In short, the data would have to be scraped from many hundreds of manual browsing sessions using a recording script incorporating different pieces of the scraper and another Node library Puppeteer Recorder. This, combined with the other challenges, led to changing projects to one with a focus on the actual data modeling rather than data collection.

• Related Work:

There are many different approaches to sentiment analysis in Natural Language Processing (NLP) but one of the most common is Neural Networks and more specifically, Long Short Term Memory (LSTM). LSTM's use a variety of methods to encode the context of the words in a sentence. One of these is recurrence which is having the network pass the previous word as an input to the next word which can make early words in a sentence affect words later on. However, this doesn't extend very far with regular recurrence and the influence is lost after only a few words. What is actually useful, the memory part of LSTM's, is long-term dependencies introduced when each neuron passes data along in multiple ways such as using nonlinear functions to prevent the fading of the previous memory and gates to optionally let information through. One last modification is a Bidirectional LSTM which is quite complex but basically can help learn context by looking at the beginning and end of a sentence when considering a specific word by having a channel of data flowing backwards as well as forwards. [1-6]

One vital step in the neural network regardless of the LSTM is embedding the words as feature vectors. This converts the words from strings to vectors of integers. In the competition, there was a list of predefined embeddings that were allowed for use. You can find the use of different embeddings in various submissions, such as [1-7], which uses the GloVe and Paragram embeddings [GloVe reference 1-8]. Throughout almost all of the top submissions that are publicly available, these 2 embeddings are utilized along with some others. The predefined embeddings all have many different advantages and disadvantages. GloVe, for instance, is used to give context between words. It does this by considering pairs of words rather than a simple word-to-word comparison. This allows for context outside the traditional bag-of-words representation. There is, however, a drawback in that it has difficulty in separating opposing concepts, and it can use a lot of memory. The kaggle post, [1-9], gives an in depth breakdown of 4 pre-trained embeddings. From the post, the notable conclusions were that, “Overall pre trained embeddings seem to give better results compared to non-pretrained model.” and that, “The performance of the different pretrained embeddings are almost similar.”

There was also an issue in that kaggle switched the format of the embedding to a zip archive 2 months ago due to this being an old contest. Since the embeddings are too large to unzip on kaggle, code from [1-10] was adapted in order to get specific data from the zip files.

For dealing with the disparity of class labels, an initial source of inspiration came from Cost-sensitive Learning vs. Sampling [1-4] in an attempt to deal with the large disparity of class labels. Some of the more interesting conclusions of the paper stated, “Based on the results from all of the data sets, there is no definitive winner between cost-sensitive learning, oversampling and undersampling”. The second piece that is interesting is the discussion on the disadvantages of oversampling in that, “The main disadvantage with oversampling, from our perspective, is that by making exact copies of existing examples, it makes overfitting likely.”[1-4] And, “A second disadvantage of oversampling is that it increases the number of training examples, thus increasing the learning time.”[1-4] These two disadvantages offer an interesting issue to explore further with the data.

• Solution:

The overall approach taken was to test multiple different models to see how each one would compare while also focusing on and researching Neural Network solutions the most as they had shown the most promise.

For the neural network, after preprocessing the vocabulary size was 200-302K words, in the final solution it was at 302K words. There was also thresholding implemented upon the maximum length of any particular question to be a maximum of 50-55 words per question. After building the vocabulary, an embedding matrix was created by going through the vocabulary and checking if each word was in the dictionary created by GloVe. There was a priority created in which if a word after being processed in different manners (lemmatized > stemmed > capitalized > uppercase > lowercase > base word) the vocab word would then be placed into the embedding matrix if it existed within the dictionary created from GloVe. The model was then built with embeddings from the restricted vocabulary with an embedding of 300 dimensions for each word feature vector.

The network itself was built using Keras with Tensorflow as a backend as this is one of the most common neural network frameworks used in research and in industry. After the embedding layer, the network consisted of an LSTM layer with 100 neurons to capture sufficient contextual complexity for each question. The output of the LSTM was densely combined into a single output neuron with a sigmoid activation function to limit the output to simply 0 to 1 representing the toxicity of the question. For actual class label predictions, the output was thresholded at 0.5 for toxic vs non-toxic. However, submissions to Kaggle consisted of real-valued numbers between 0 and 1 to give a more approximate prediction.

• Experiments:

• Data:

The dataset presented quite a few challenges. There were 1.3 million records in the training set and 300,000 in the test set. The data was in text format and had a binary class label indicating toxic or sincere. The total file size for the training set was over 121 MB and the test set had a size of 34MB. This large size led to fairly long run times. Considering the size of the dataset, noise was also an issue within the training and test sets. Some examples of noise from each class are,

label 1 (from training set, labeled incorrectly as insincere)

- Where does Elon Musk get his haircut?
- I just turned 18, what are the best decisions I can make to generate success?
- Am I able to kick someone out if they're not on my lease?

label 0 (false negative, labeled incorrectly as sincere):

- Trump state of the union speech?
- Why the Poles like me hate Poland?
- Why do you Catholics worship Mary instead of God or Jesus?

The other notable pieces of data came from the embeddings folder. This folder was a zipped archive, from it only Glove was utilized, which when decompressed had a file size of over 6 GB. This led to issues later described as one can't access files in a zip archive on kaggle directly, and the kaggle virtual environment doesn't allow the files to be unzipped due to a max disk space allocation of 4 GB.

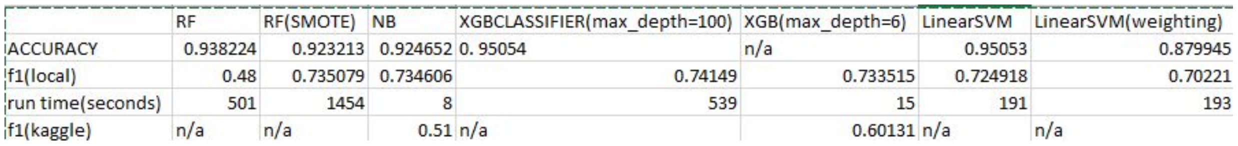
• Experimental setup:

The metric chosen for validation was the f1 score. This choice arises from the large imbalance in the class labels, causing accuracy to be a less meaningful measure. The kaggle competition scores submissions based on f1 score as well. The network trained using binary cross entropy as a metric for training since f1-score isn't differentiable and backpropagation requires a differentiable loss function.
For evaluating every classifier outside of neural networks, the Tfidf Vectorizer from sklearn was overloaded to stem each word when analyzing with PorterStemmer. In the vectorization process, there was also document thresholding performed to limit the number of unique words to be considered. For the Neural Network, to preprocess spacy was used to tokenize the input, remove stop words and punctuation, and to perform lemmatization. This was done to create a vocabulary and a lemmatization vocabulary. During the next step these two vocabularies were used to help construct the embeddings matrix. The embeddings were also in a zip archive which presented an issue in that information from the zip file needed to be retrieved without unzipping the folder (due to size constraints within the virtual IDE provided by kaggle). This process was achieved with the help of the zipfile library and io.TextIOWrapper. After reading the embeddings an embedding matrix was able to be created that could be thrown into the model.

The set up for the classifiers outside of the neural networks were as follows: preprocessing the text data, then running the processed text data through the various classifiers, and using the F1-score, accuracy, and confusion matrices to help evaluate each classifier. Several graphs were originally created to demonstrate the tuning of parameters on these other classifiers, but seeing the poor scores when validating with kaggle they were omitted. The functions for graph creation and tuning still remain in the source code with functions like plot, n_estimate, lr, etc.

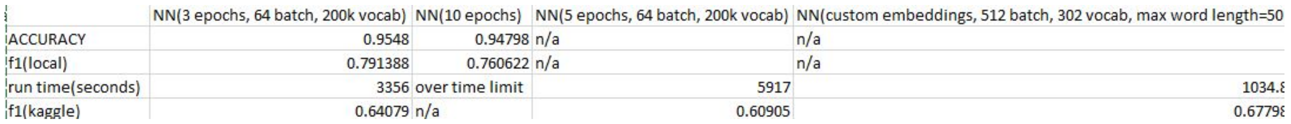
• Brief analysis & conclusion

In the initial testing on the various classifiers (gradient boosting trees, SVM, NB, random forest, random forest with oversampling, and undersampling), there were relatively high training f1 scores ranging from .48-.74 as noted in figure 2-1 below. This was likely due to overfitting the training set though, as when submitted to kaggle the f1 scores for these classifiers all performed very poorly outside of gradient boosting trees. Gradient boosting trees did ok with an f1 score of just over 0.6. As a result most of the parameter tuning and the like was thrown out and the details are largely useless. The one major note from gradient boosting trees was that the max depth had a large effect on the overfitting obviously, and the one decent result in this regard came when switching the max depth from 100 to 6.



	RF	RF(SMOTE)	NB	XGBCLASSIFIER(max_depth=100)	XGB(max_depth=6)	LinearSVM	LinearSVM(weighting)
ACCURACY	0.938224	0.923213	0.924652	0. 95054	n/a	0.95053	0.879945
f1(local)	0.48	0.735079	0.734606	0.74149	0.733515	0.724918	0.70221
run time(seconds)	501	1454	8	539	15	191	193
f1(kaggle)	n/a	n/a	0.51	n/a	0.60131	n/a	n/a

Figure 2-1



	NN(3 epochs, 64 batch, 200k vocab)	NN(10 epochs)	NN(5 epochs, 64 batch, 200k vocab)	NN(custom embeddings, 512 batch, 302 vocab, max word length=50
ACCURACY	0.9548	0.94798	n/a	n/a
f1(local)	0.791388	0.760622	n/a	n/a
run time(seconds)	3356	over time limit	5917	1034.8
f1(kaggle)	0.64079	n/a	0.60905	0.67798

Figure 2-2

One surprising result from the early testing of neural networks was how quickly the model overfit on the dataset. Most NLP solutions in research train for over 50-100+ epochs so coming into this project, a higher number of epochs was initially used but was quickly lowered after discovering how much overfitting was occurring. This was backed up by looking at the approaches other people submitting to the competition took.

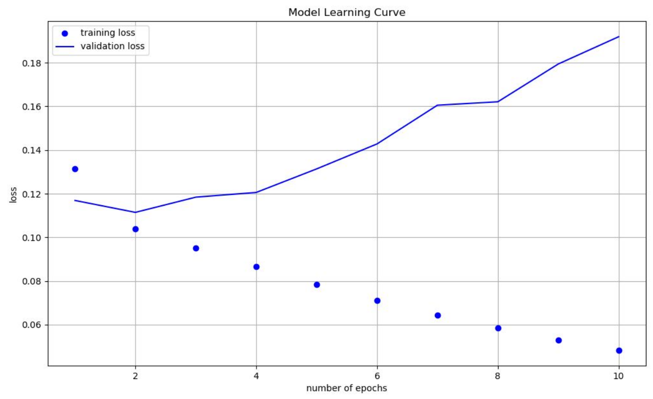


Figure 2-3

Figure 2-3 shows how the network is overfitting after only 2 epochs demonstrated with binary crossentropy as the loss function (lower is better).

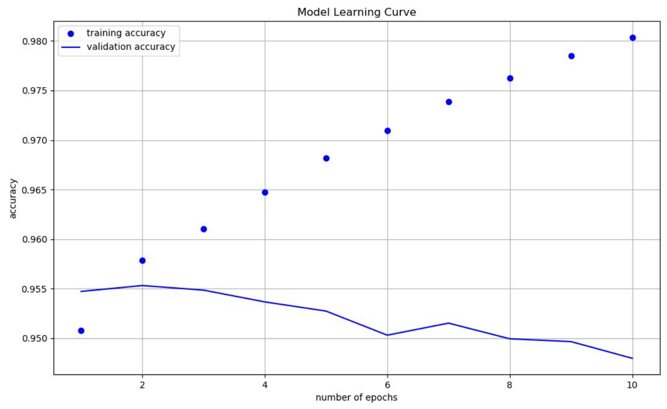


Figure 2-4

Figure 2-4 utilizes accuracy to demonstrate a similar concept of overfitting, but it’s results are less clear when compared to those of crossentropy.

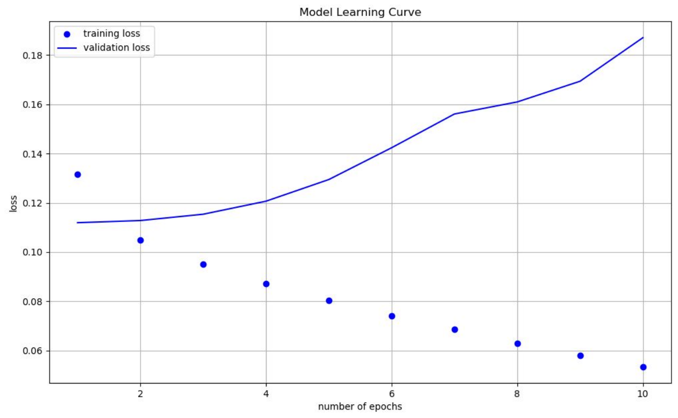


Figure 2-5

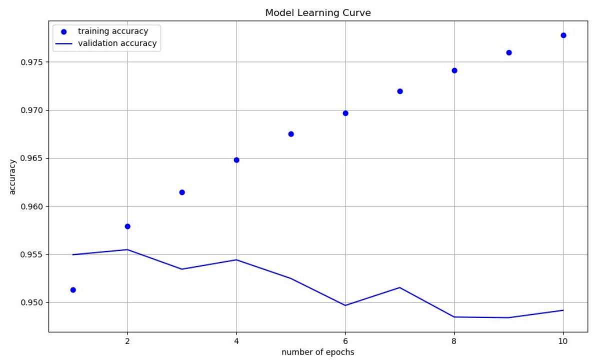


Figure 2-6

These results were run with identical parameters to the previous run but also had a Dropout Layer with 50% dropout probability for the outputs of the LSTM. Dropout is used by many researchers to directly combat overfitting as it randomly ignores some of the outputs of the network to prevent learning the exact dataset. However, as can be observed by the graph, the network still overfitted to the dataset almost identically to the network without Dropout.

One of the most crucial factors for the implementation was the use of the GloVe pre-defined word embeddings compared to learning the word embeddings along with the rest of the neural network. In order to keep the training time roughly similar between the two types of runs, the dimension of the custom learned embeddings was capped at around 100 dimensions. The GloVe embeddings were 300 dimensions which was one of the standard embedding sizes provided by Kaggle.

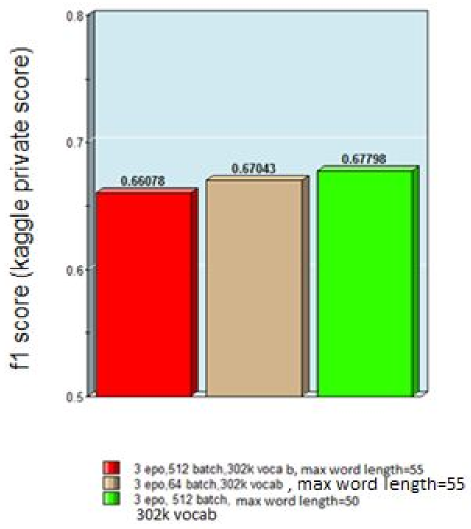


Figure 2-7

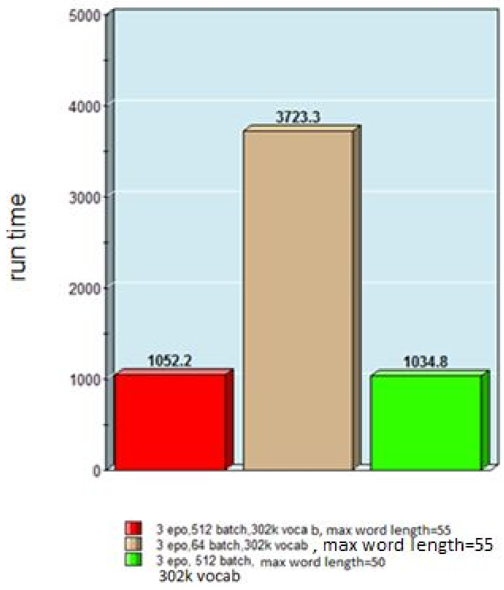


Figure 2-8

Figure 2-7 shows the Kaggle-submitted f1-scores for the neural network with the pretrained GloVe embeddings with various different parameters with the third column being the best submission achieved across all combinations of parameters. The middle column in both graphs shows how using a smaller, finer batch size results in a 3 times increase in runtime in Figure 2-8 while not increasing the score significantly compared to the red column in Figure 2-7.

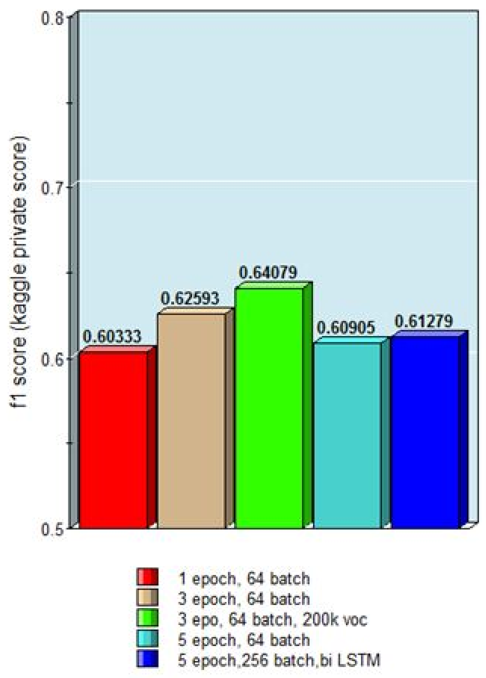


Figure 2-9

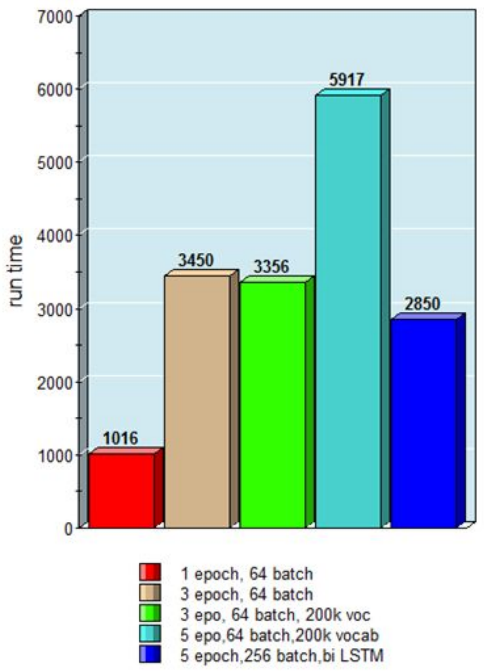


Figure 2-10

Figure 2-9 similarly shows the Kaggle-submitted f1-scores for the neural network with the custom embeddings with various different parameters with the third column being the best submission achieved using custom learned embeddings. Both blue columns in Figure 2-9 show how the model overfits after only 3 epochs. An interesting item in figure 2-9 is that the last column is run with a Bidirectional LSTM but the difference it makes is inconsequential compared to the number of epochs causing overfitting. Figure 2-10 demonstrates the increase in runtime with increasing numbers of epochs as well as a decrease in runtime when the batch size was decreased to 256 for the Bidirectional LSTM.

Conclusion:

Upon analyzing the results, it can be observed that neural networks perform much better than all of the other classifiers tested. Beyond this, it can be discerned that there was likely overfitting occurring in all of the models due to the discrepancy between the f1 score computed locally and the validated f1 score on Kaggle. It is also noted that preprocessing was a crucial factor in the performance of each model, and improved scores could definitely be achieved in the future. Another key detail comes from the noted discrepancy in scores between pre-trained and custom embeddings as one can see that custom embeddings fared much worse than the pre-trained embedding (GloVe). Going forward, utilizing a weighted combination of multiple feature embeddings in the input layer of the model, as noted by [1-9], could lead to significant increases in f1-score. Additionally, more preprocessing could be done to more thoroughly check each embedding for the existence of a word such as attempting to search for typos.

References:

1 (citations)

1. https://www.kaggle.com/c/quora-insincere-questions-classification/overview
2. https://en.wikipedia.org/wiki/Cohen's_kappa
3. https://www.svm-tutorial.com/2014/10/svm-linear-kernel-good-text-classification
4. https://storm.cis.fordham.edu/gweiss/papers/dmin07-weiss.pdf
5. N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer. SMOTE: synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, Volume 16, 321-357, 2002.
6. http://colah.github.io/posts/2015-08-Understanding-LSTMs/ https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using-keras-f8e9ed76f046
7. http://colah.github.io/posts/2015-08-Understanding-LSTMs/ https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using-keras-f8e9ed76f046
8. https://nlp.stanford.edu/projects/glove/
9. https://www.kaggle.com/shujian/different-embeddings-with-attention-fork-fork
10. https://www.kaggle.com/c/quora-insincere-questions-classification/discussion/148324

Team member contribution:

Everyone contributed equally to the project