Linguistics 167 Final Project

Chat Abuse Classifier

Prabhjot Singh, Rene Robert Ndizeye, Jamicko Tan, Timothy Lue

Introduction

In today's society, the world of the internet, especially social media, is now widely accessible to anyone who wishes to browse it. From the oldest members of our society to the youngest, anyone can learn new information or interact with anyone from around the world. The inspiration for this project arose from a real world need to make the internet a safe place to share ideas freely and respectfully. Especially young children who actively engage in online activities and conversations on a daily basis are the most vulnerable to be exposed to unsafe and unmoderated content. These interactions are most likely to cause severe mental health issues, so our team decided to work on a project for which the goal would be tackling such harsh situations or at least attempt to find a temporary solution to this matter.

Using the dataset from the "Toxic Comment Classification Challenge" on Kaggle, we came up with our own project that is to analyze a set of conversations and then tell if a person is trying to bully the other person or if both people are being abusive to each other. We implemented, trained, and then tested our deep learning model using tools and techniques learnt in class. We wanted our deep learning model to be able to both analyze and understand the context of a conversation between two people and then classifying each person's responses to be either abusive or not.

Model

A High Level Look

Let's begin by taking a look at the model as just a single machine that takes in a sentence as an input and outputs whether the sentence was abusively directed towards someone or not. The approach that we take in this project is to feed the sentence sequentially to an Elman Recurrent Neural Network, which generates a hidden state after processing the entire sentence, which is then fed into a Multi-Layer Perceptron which predicts the class of the sentence.

Now that we have an idea of what the model does at a high level, let's discuss how the entire process actually takes place,

Generating the Tensors from the Sentence Strings

After preprocessing of the input data file, the data that goes into the model is a tuple of the sentences and their respective labels. The sentences themselves are a list of word tokens, which have been cleaned, using regex, so that they don't have any unwanted characters in them (later we found that this helps increase the accuracy of the model.)

Now that we have a list of sentences, each of which is a list of words, we want to convert the words to their respective word embeddings. For this project we used the 50 dimensions GloVe dataset to represent the word embeddings. Now instead of loading all the word embeddings for our entire dataset at once, what we did to make our model more resource efficient was to load in the word embeddings per batch of the dataset. So right before a batch would go for training, its word embeddings would be generated.

Now that we have a batch of sentences, where each sentence is a list of word embeddings, we convert the lists to pytorch tensor objects, and then feed this as input to our model. A single datapoint in our project dataset consists of multiple word embeddings tensors that make a specific sentence. In this paragraph from here on, a sentence means a list of word embedding tensors. For this project, we used an Elman Network as the recurrent neural network to process the text, and for classification we used a multilayer perceptron.

How the model works

Once our model takes in a sentence, it feeds the embedding of the first word into a Multilayer Perceptron to generate the first hidden state of dimension 50 for the Elman network, The weights and biases of this MLP will be learned during the training process, so that it can produce the best first hidden state.

The output from the first MLP, which is our initial hidden state is fed into the recurrent neural network along with the word embedding of the first word. Whatever hidden state is generated by the RNN is fed back into the RNN as $h_{previous}$ to process the second word, and so forth. We use a loop to go through all the words of the sentence being fed into the RNN, which allows the model to be generalized for sentences of any given length. After the hidden state is generated from the last word of the sentence, we are ready to feed it into the MultiLayer Perceptron for classification.

$$h_{current} = ReLU(W_{ih}x + b_{ih} + W_{hh}h + b_{hh})$$

(x is the word embedding vector, and h is the previous hidden state)

The 50 dimension hidden state generated by after processing the last word of the sentence, is then fed into a MultiLayer Perceptron, which has two hidden states each of which have 50 nodes, and at the end it has two classes. For the layers of the multilayer perceptron we are using the Linear function of Pytorch, then applying the ReLU Non Linearity, and at the end we are taking the softmax of the generated outputs.

The first layer of the MLP takes in the dimension 50 tensor generated by the rnn, and applies a linear transformation

$$y = xA^T + b$$

Source: Pytorch documentation, x is the input tensor

We then apply a Nonlinearity to this generated output, to generate our first hidden layer

$$y' = ReLU(xA^T + b)$$

We then do the same thing for the second hidden layer, take the first hidden layer as input, after the output of the second hidden layer is generated to a target of two classes, we then take the softmax of the values that were generated.

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

The output of the MLP is then compared against the actual labels to compute the loss, and then the adam optimizer is used to compute the gradient and learn the parameters.

For this model we used a batch size of 32, a learning rate of 0.0001 . To compute the loss we used the built in function MSELoss of pytorch with the reduction parameter set to 'sum' .

Replicating the model six times

For this project our goal was to create a model that can predict 6 different types of abuses. So initially we created the model as described above, to just predict a single label, and once we were happy with the results of the first model, we created five more model objects from the SimpleRNN model class that we had defined. The object oriented approach to this project saved us greatly from having to write redundant code.

Datasets and Preprocessing

This project is using the dataset from the "Toxic Comment Classification Challenge" on Kaggle. https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data

The dataset consists of a large number of Wikipedia comments which have been labeled by human raters for different toxic behaviors. These types of toxicity include: `toxic, severe_toxic, obscene, threat, insult, and identity_hate`. Each comment is assigned six-tuple of binary labels, that correlates to the types of toxicity. A comment having a 1 means that the comment is one of the six types of toxicity cited previously and having a 0 means otherwise.

For the training and testing data sets, we first clean each sentence removing new line characters into a space and converting each word into all lowercase. We then create a sentence-label tuple such that each sentence is an array of each word, and each label in the tuple is a six-tuple binary representation of each toxicity case.

For the vector representation of each word, we used the GloVe dataset provided by Stanford. This dataset maps each word into a 50-tuple whose values are produced by their trained model. We then converted each word in the sentence array into a 50-tuple of their vector

representation and we discarded each word in the sentence that do not exist in the GloVe dictionary.

Initially, our data set has 160,000+ data points. distributed such that there are 143346 neutral points, 15294 toxic points, 1595 severe toxic, 8449 obscene, 478 threat, 7877 insults, and 1405 identity hate. Note that toxicity is a precondition for severe toxic, such that each point labeled severe toxic is also labelled toxic. After seeing the skew in distribution of data, we ended up purging most of the neutral points and cutting it down to 30000 neutral points, leaving us a training set of around 46225 which yielded an increase in performance.

Lastly, after all of the steps above, each data point is a `<sentence,labels>` tuple wherein `sentence` is an array 50-tuple word embeddings of each word in the sentence that exists in the GloVe dictionary. On the other hand, `labels` are a 6-tuple binary representation of the labels signaling whether or not the sentence falls in a toxicity category.

Results

To analyze how well our model did, we calculated the average precision-recall score on our predictions for each of the six different toxicity labels. Precision is the fraction of relevant instances among the retrieved instances and recall is the fraction of the total amount of relevant instances that were actually retrieved. High precision means that our model returned substantially more relevant results than irrelevant ones, while high recall means that our model returned most of the relevant results. To give us a proper visualization of our results, we then graphed the precision-recall curve based on the calculated average precision-recall score.

After the cleaning and preprocessing, we split our dataset into three parts: train_set, intermediate_test_set, final_test_set, each of which had a size of 30000, 10000, and 6225. We trained our model on the train_set and calculated the average precision-recall score on our predictions with the intermediate_test_set and the final_test_set.

As seen in the table and graphs below, the average precision-recall score was the highest for the label toxic and the lowest for the label threat. This means that our model returned most of the relevant results and more relevant results than irrelevant ones for the label toxic, but not for the label threat.

Result of Precision Recall For the intermediate_test_set,

Average precision-recall score: 0.78 label : toxic

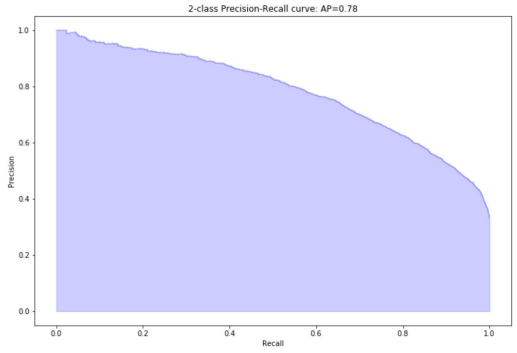
Average precision-recall score: 0.20 label : severe_toxic

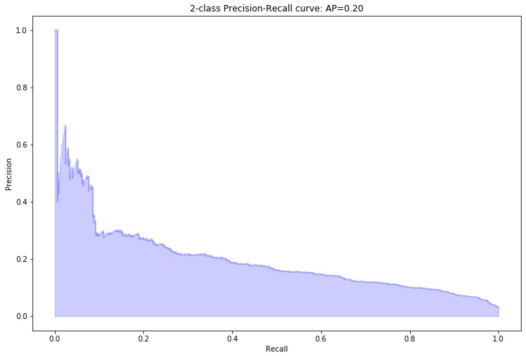
Average precision-recall score: 0.63 label : obscene

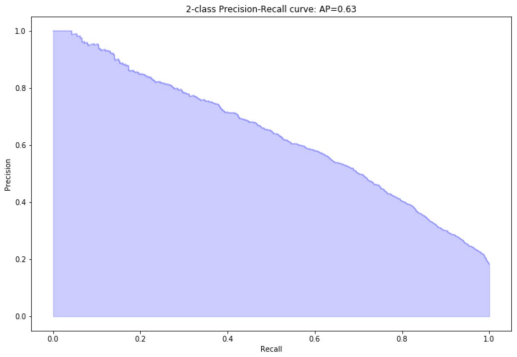
Average precision-recall score: 0.01 label : threat

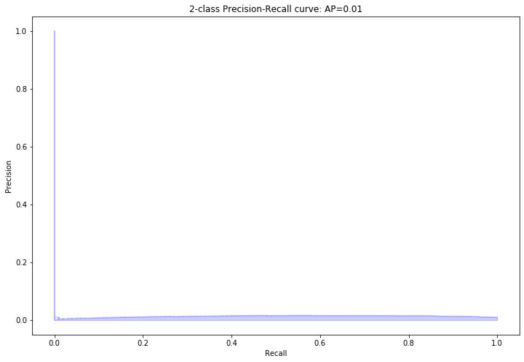
Average precision-recall score: 0.58 label : insult

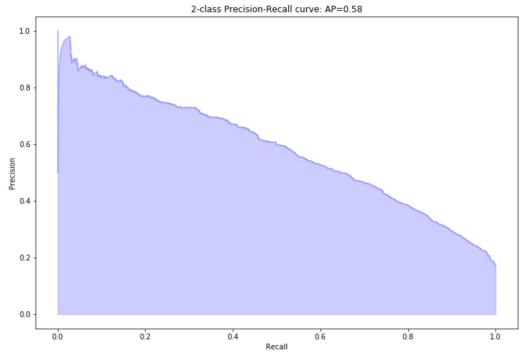
Average precision-recall score: 0.11 label : identity_hate

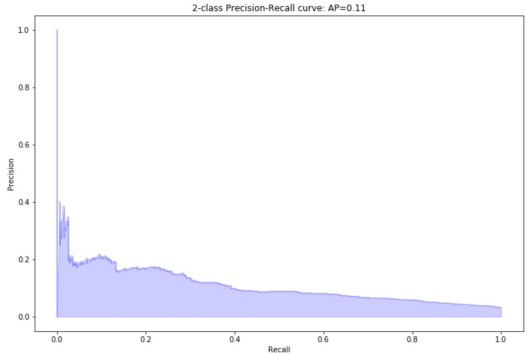












Result of Precision Recall For the final_test_set,

Average precision-recall score: 0.80 label : toxic

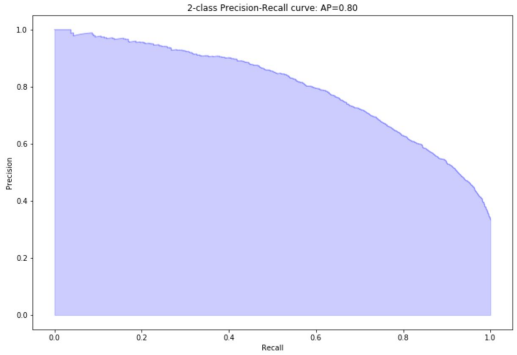
Average precision-recall score: 0.20 label : severe_toxic

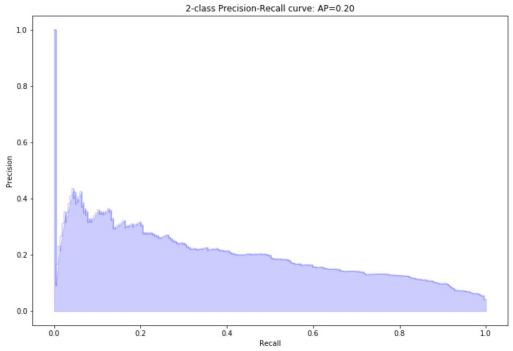
Average precision-recall score: 0.64 label : obscene

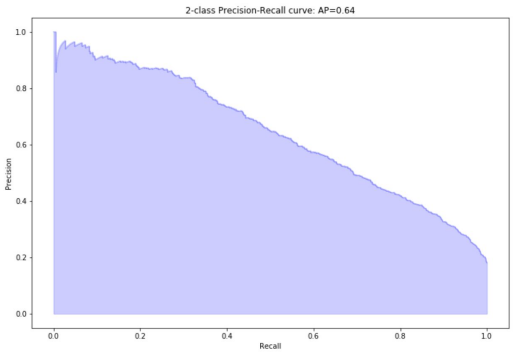
Average precision-recall score: 0.01 label : threat

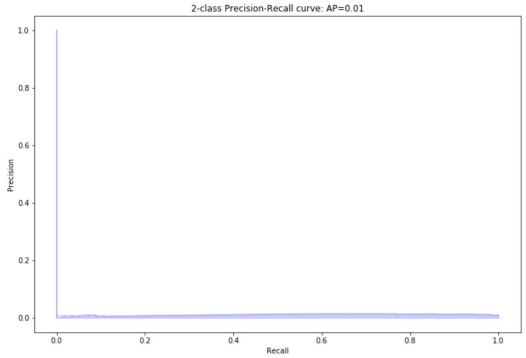
Average precision-recall score: 0.58 label : insult

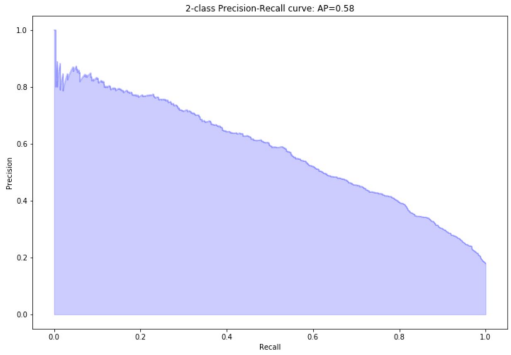
Average precision-recall score: 0.09 label : identity_hate

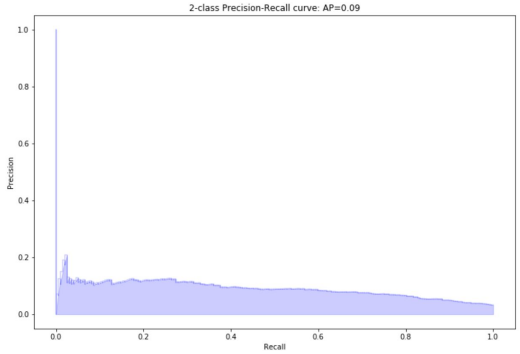












Graphical User Interface

During our presentation, we presented a graphical user interface (GUI) that allowed users to interact with our model by typing in speech inputs for two users talking to one another. If any of the speakers was abusive, the GUI would use our model to firstly determine who was being abusive and then assign the specific label or labels of toxicity that the speaker used. For situations where no speaker was being abusive, the GUI would state that neither speaker was abusive.

Conclusion

What we learnt from our experiments is that having more data points helps significantly with the accuracy and precision of the model. In the case of the toxic label we had 15,294 sentences out of 46,225 that were labeled as toxic. That allowed the model to achieve an average precision recall score of 0.80 on the final test set. On the other hand for the threat label we only had 478 sentences out of 46,225 that were labeled as threats, so not having enough data points caused the model to achieve a very low precision-recall score of 0.01.

From this project we also learnt the importance and benefits of using an elman network for language processing, and their ability to retain information about the order of the words in a given sentence.