Hate Speech Detection - Model Design

Kareem Ehab Kassab 900182771
Computer Science and Engineering
The American University in Cairo
kareemikassab@aucegypt.edu

Mahmoud Elshinawy 900183926
Computer Science and Engineering
The American University in Cairo
mahmoudelshenawy@aucegypt.edu

I. Introduction

In the last phase, we experimented with different ready-implementation machine learning models (e.g. Logistic Regression, Neural Network, SVM, Decision Tree, Naive Bayes) and decided on the best model in terms of recall as it is the most relevant performance measure to our problem. Additionally, we tried different values for some hyper-parameters to choose the best values for them as well. As a final result, a Neural Network with 25-neuron one hidden layer gives the best performance in our case.

II. Final Model

Based on Experimenting in the last phase, our final model will consist of a neural network with one input layer, one layer of text embedding, and one hidden layer and one output layer. We decided on using the Word2Vec model for our text embedding, specifically the CBOW algorithm.

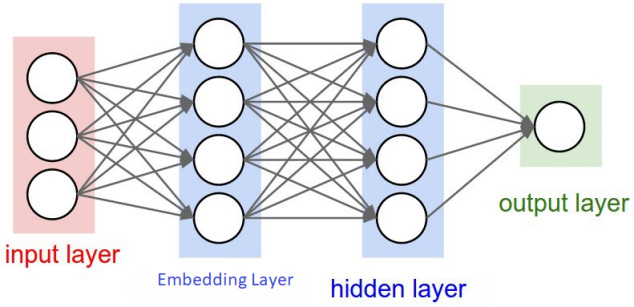

Fig. 1. Overall Final Model Design.

We used the CBOW model as it was more suitable for our case: it is faster to compile, it predicts the word in its window context, and since we want to identify certain offensive words and hate speech in whichever context, we saw no need to invest the extra time in the skip-gram variation. For our CBOW, we used a vector size of 500 and a window size of 4. The window looped over the whole corpus generating a training set of (108,444*500) and testing set of (27,112*500) an 80/20 split.

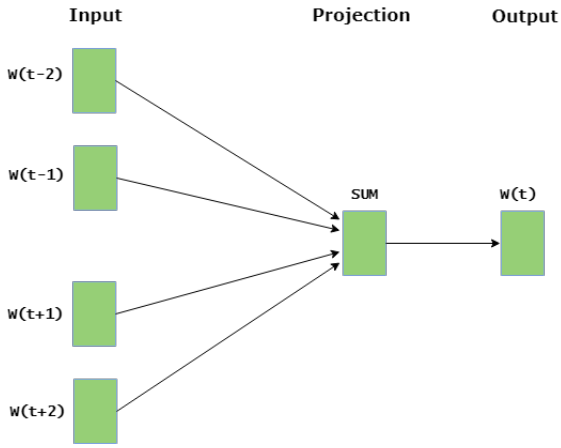

Fig. 2. Overall Final Model Design.

III. Hyper-parameter Choice

Starting from the model above, we aim to try different hyperparameters such as the activation function, solver, and the learning rate to choose the best model with the best hyperparameters.

A. Activation Function

For the hidden layer, We tried different activation functions in the classifier parameter list such as logistic, tanh, and relu. The relu performed better than all other activation functions in our case. With a little more investigation on the 'relu' function, we found that this function outputs values in range (0, infinity). In face, our word embedding vectors contain negative numbers. This means that all negative inputs are going to be zero when processed in the hidden layer.

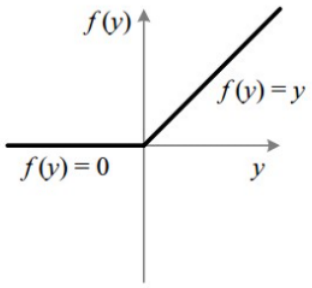

Fig. 3. ReLU Activation Function.

To avoid this, we need to fine-tune the 'relu' function to have the negative values not scaled to zero. There is a modified version of the 'relu' function called 'randomized relu' gives a value of a*y when negative values are encountered rather than outputs zero. The following figure shows the 'randomized relu'. When a=0.01, it is called 'leaky relu' function (Sharma, 2021).

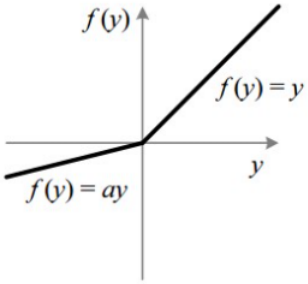

Fig. 4. Randomized ReLU Activation Function.

For the output layer, we choose 'tanh' activation function as our problem is a binary classification where 1 represents HateSpeech and 0 represents NotHateSpeech. The sigmoid function will not be the best choice as it outputs the probability that a certain document belongs to a certain label, not the label (Sharma, 2021).

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

B. Solver

At this point, we used 'relu' as our activation function, and started to experiment with different solvers [lbfgs, sgd, adam]. ADAM solver obtained the best performance among all other solvers. Fig. 5. shows the performance report of the neural network with ADAM solver.

Neural Network Relu adam (Hidden Layer= 25)

```
[9]: from sklearn.neural_network import MLPClassifier
nn_model_reluadam = MLPClassifier(hidden_layer_sizes=(25,),random_state=1,solver='adam', max_iter=5000, activation=
nn_model_reluadam.fit(train_df.values,y_train.values)
nn_results_reluadam = nn_model_reluadam.predict(test_df.values)

[0]: from sklearn.metrics import classification_report, accuracy_score
print(classification_report(y_test.values,nn_results_reluadam))
```

```
              precision    recall  f1-score   support

         0.0       0.81      0.85      0.83     16018
         1.0       0.77      0.72      0.74     11094

    accuracy                           0.80     27112
   macro avg       0.79      0.78      0.79     27112
weighted avg       0.80      0.80      0.80     27112
```

Fig. 5. ADAM Performance Report.

C. Learning Rate

After choosing the best solver, we need to decide on the best learning rate that would minimize the loss and also does not affect the training time much. One way to do it is to start with a relatively large learning rate, e.g. 0.01, and then start to decrease the learning rate more and more until we find the optimal learning rate. We tried 0.01 then 0.001 then 0.0001 then 0.00001, and the more we decrease the learning rate, the better performance we get until we reach the threshold of 0.0001. When we decrease it to 0.00001, the performance gets worse. Therefore, we will be using 0.0001 as our learning rate. Fig. 6. Shows the performance of the model with learning rate 0.0001.

Neural Network Relu adam 0.0001 (Hidden Layer= 25)

```
[27]: from sklearn.neural_network import MLPClassifier
nn_model_reluadam0001 = MLPClassifier(hidden_layer_sizes=(25,),random_state=1,solver='adam',learning_rate_init=0.00
nn_model_reluadam0001.fit(train_df.values,y_train.values)
nn_results_reluadam0001 = nn_model_reluadam0001.predict(test_df.values)

[28]: from sklearn.metrics import classification_report, accuracy_score
print(classification_report(y_test.values,nn_results_reluadam0001))
```

```
              precision    recall  f1-score   support

         0.0       0.81      0.86      0.83     16018
         1.0       0.77      0.71      0.74     11094

    accuracy                           0.80     27112
   macro avg       0.79      0.78      0.79     27112
weighted avg       0.80      0.80      0.80     27112
```

Fig. 6. Learning Rate of 0.0001 Performance Report.

We also tried the 'adaptive' learning rate option, but we did not get any better performance than we got with the constant 0.0001 learning rate. Fig. 7. Shows the performance of the model with adaptive learning rate initiated at 0.0001.

D. Loss Function

Binary Cross-Entropy Loss function is the best fit for a binary classification problem given

Neural Network Relu adam 0.0001 adaptive (Hidden Layer= 25)

```
1]: from sklearn.neural_network import MLPClassifier
    nn_model_reluadam0001 = MLPClassifier(hidden_layer_sizes=(25,),random_state=1,solver='adam',learning_rate='adaptive
    nn_model_reluadam0001.fit(train_df.values,y_train.values)
    nn_results_reluadam0001 = nn_model_reluadam0001.predict(test_df.values)

2]: from sklearn.metrics import classification_report, accuracy_score
    print(classification_report(y_test.values,nn_results_reluadam0001))

                  precision    recall  f1-score   support

              0.0       0.81      0.86      0.83     16018
              1.0       0.77      0.71      0.74     11094

         accuracy                           0.80     27112
        macro avg       0.79      0.78      0.79     27112
     weighted avg       0.80      0.80      0.80     27112
```

Fig. 7. Adaptive Learning Rate of 0.0001 Performance Report.

labels 0,1. This loss function calculates a score that represents the average difference between the actual and predicted probability distributions for predicting class 1.The following is the formula for the binary cross entropy loss function (Godoy, 2019).

$$A = \frac{-1}{output\_size} \sum_{i=1}^{output\_size} y_i * \log_2 yhat_i + (1-y_i) * \log_2 (1-yhat_i)$$

IV. The Utility Application

In this project, we aim to classify and differentiate hate/offensive speech from normal speech. For this process, we are building a Neural Network model that takes text as an input and outputs its class; For this to be possible we need to dive into technical details of preprocessing, testing models and assessing performance, and implementing the model. In this phase, we are concerned with designing our chosen model including details such as hyper parameters– which can be further modified. The user should not be considered with all these details, thus, we are concerned with the software design of the application that allows human interaction with such model: input/output format, backend, frontend, and data flow in the application. Simply, the application aims to take a text sample string as an input and output a 'class' indicating whether it is hate/offensive speech or not.

A. Software Architecture Model

As per design requirements, we are aiming to model our utility application to follow a client-server network model that is based on separation:the model will be on the server, while human interaction will happen through the interface on the client that communicates the input with the server through an API and get the class (hate/offensive speech or not) as a response.

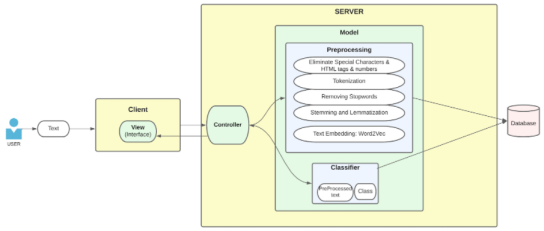

Fig. 8. Software Architecture Model.

This separation need of the utility application makes the MVC (Model View Controller) design pattern suitable. The MVC helps us separate the view (what the human user will be seeing) from the model (classifier and preprocessing) while controlling the communication with APIs in the middle. The view will be on the client and allow the user to input text sample that is to be classified and send it to the controller; it will also display the classification response. The model has 2 parts: preprocessing (including all steps on phase 2) and word2vec text embedding, and the classifier which predicts the class based on the output of the first part– these reside on the server (see fig.9.). The controller will exchange data between client and server and between the preprocessing and classifier parts of the model . The database will be used to store the trained model values and weights. It connects to the server to get the output from the model parts.We are choosing Django to implement the utility application; the reasoning for that being its implementation simplicity, the project is already being implemented in python, it being and supporting the MVC. see the MVC in Django in (fig 9)

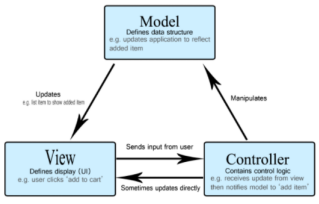

Fig. 9. MVC model.

B. Flow (Training)

First, the Dataset is fed into the preprocessing model: special characters, stop words, numbers are removed, and words are returned to their stem forms. Then, text embedding word2vec is performed. The word2vec puts words in vectors with numerical representations. Then, such vectors are fed to the Neural Network Classifier model to train on. Finally, the trained weights are stored in the database for testing and predicting user inputs.

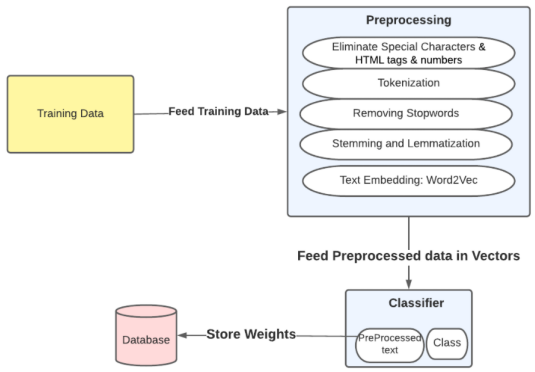

Fig. 10. Flow Training.

C. Flow (Testing)

Similar to the previously explained processes, when the user inputs a test sample string, it will get preprocessed following the same procedures, then its vectors will be fed to the classifier. The classifier will, then, use the already stored weights to be able to predict whether this is hate/offensive speech or not, and outputs that to the user.

D. Backend

The backend will be implemented using python Django; the main reasoning for this is it being easy to implement, and also integrating well with our whole project as we are writing python code either way. It Also offers us the ability to implement it in the front end and connect both using the suitable APIs. Simply the backend will take input and return output using POST and/or GET requests after computing the class based on the trained model.

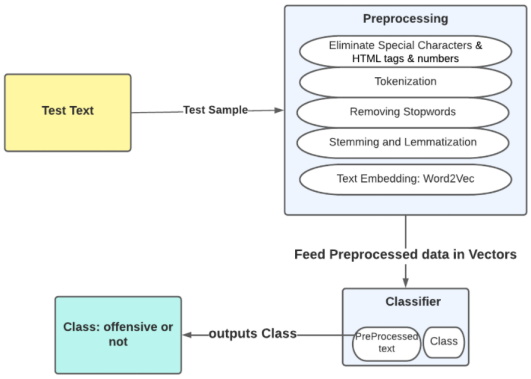

Fig. 11. Flow Testing.

E. Frontend

Similar to the backend, the frontend will also be implemented using Django. We will use its web interface for similar reasoning including it being easy to implement, already being used in the backend and the rest of the project being written in python, and also because Django has a lot of useful packages that will save some time. Django will allow us to connect both our front end and back end. The server model explained above can also be implemented using Django, making it suitable for the whole application model.

REFERENCES

[1] Godoy, D. (2019, February 7). Understanding binary cross-entropy / log loss: A visual explanation. Medium. Retrieved from https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

[2] Sharma, S. (2021, July 4). Activation functions in neural networks. Medium. Retrieved April 3, 2022, from https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6