Hate Speech Detection using Transformers and Deep Learning

By:

Farheen Fatima
Mohamed Derbeli

- Introduction
- Dataset
- Data preprocessing
- Models
- Results

The term hate speech is understood as any type of verbal, written or behavioral communication that attacks or uses derogatory or discriminatory language against a person or group based on what they are. In other words, based on their religion, ethinicity, natiionality, race, color, ancestry, sex or other identity factor, In the problem, we will take you through a hate speech detection model with Machine Learning, Deep Learning and Python.

Hate Speech Detection is generally a task of sentiment classification. So, for training a model that can classify hate speech from a certain piece of text can be achieved by training it on data that is generally used to classify sentiments. So, for the task of hate speech detection model, we will use the Twitter tweets to identity tweets containing Hate Speech.

Dataset & Preprocessing

- We have used Twitter tweets and this datasets contains 31962 instances and 3 features id, tweets and the label.
- We removed the id feature since it does not contribute towards our prediction.
- We removed the stopwords, punctuation marks, urls etc.
- We performed tokenization and lemmatization on our data.
- The dataset had 29720 hate speech tweets and 2242 free speech tweets in the target feature. So we did oversampling of the data for balancing the data.

Models

We implemented various Machine Learning algorithms such as Logistic Regression, Random Forest, Super Vector Machine and Naive Bayes.

We implemented a Neural Network and LSTM model without using transformers and then we built an LSTM model using the Glove Transformer.

Logistic Regression Model with Imbalanced Data

	precision	recall	f1-score	support
0	0.96	1.00	0.98	2984
1	0.95	0.35	0.51	213
accuracy			0.96	3197
macro avg	0.95	0.67	0.74	3197
weighted avg	0.95	0.96	0.95	3197

Logistic Regression, Accuracy Score: 0.955270566155771

Naive Bayes Model with Imbalanced Data

```
print(classification_report(y_test_tf_wob, y_pred_tf_wob))
```

```
              precision    recall  f1-score   support

           0       0.97      0.86      0.91      5928
           1       0.25      0.60      0.35       465

    accuracy                           0.84      6393
   macro avg       0.61      0.73      0.63      6393
weighted avg       0.91      0.84      0.87      6393
```

Logistic Regression Model with Balanced Data

	precision	recall	f1-score	support
0	0.99	0.95	0.97	2948
1	0.95	0.99	0.97	2996
accuracy			0.97	5944
macro avg	0.97	0.97	0.97	5944
weighted avg	0.97	0.97	0.97	5944

Logistic Regression, Accuracy Score: 0.9703903095558546

Naive Bayes Model with Balanced Data

	precision	recall	f1-score	support
0	0.84	0.99	0.91	2948
1	0.98	0.81	0.89	2996
accuracy			0.90	5944
macro avg	0.91	0.90	0.90	5944
weighted avg	0.91	0.90	0.90	5944

Naive Bayes, Accuracy Score: 0.8977119784656796

SVM Model with Balanced Data

	precision	recall	f1-score	support
0	1.00	0.97	0.98	2948
1	0.97	1.00	0.98	2996
accuracy			0.98	5944
macro avg	0.98	0.98	0.98	5944
weighted avg	0.98	0.98	0.98	5944

SVM, Accuracy Score: 0.9835127860026918

Random Forest Model with Balanced Data

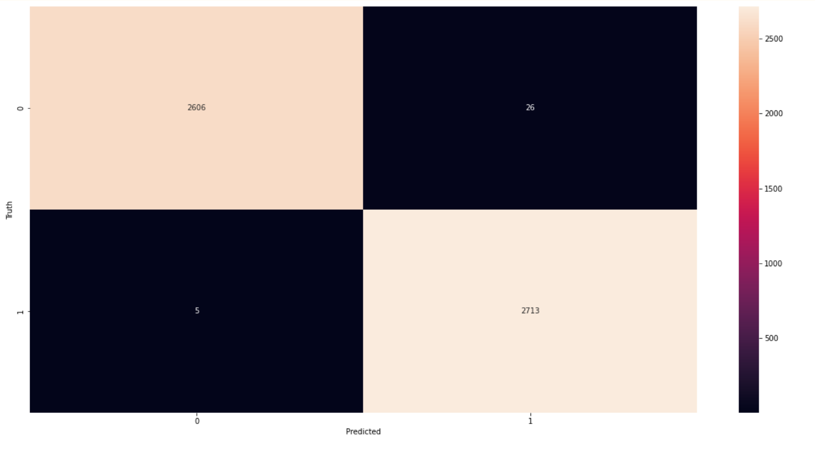
	precision	recall	f1-score	support
0	1.00	0.99	0.99	2948
1	0.99	1.00	0.99	2996
accuracy			0.99	5944
macro avg	0.99	0.99	0.99	5944
weighted avg	0.99	0.99	0.99	5944

Random Forest, Accuracy Score: 0.9944481830417228

LSTM Model without Transformer

```
embedding_mat_col=512

# model = Sequential()
# model.add(Embedding(input_dim=max_words, output_dim=embedding_mat_col, input_length=max_len))
# model.add(SpatialDropout1D(0.4))
# model.add(LSTM(50, dropout=0.4, recurrent_dropout=0.4,input_shape=(None, 512)))
# model.add(Dense(1,activation='softmax'))

# dropout=0.4, recurrent_dropout=0.4,

model_wb = tf.keras.models.Sequential([
    tf.keras.layers.Embedding(input_dim=max_words, output_dim=embedding_mat_col, input_length=max_len),
    tf.keras.layers.LSTM(20, input_shape=(None, 512)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(4, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
    ])

model_wb.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model_wb.summary())
```

Confusion Matrix




```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      2632
           1       0.99      1.00      0.99      2718

    accuracy                           0.99      5350
   macro avg       0.99      0.99      0.99      5350
weighted avg       0.99      0.99      0.99      5350
```

Results using Glove Transformer

```
#word embedding - Glove

from numpy import array
from numpy import asarray
embeddings_index = dict()
f = open('/content/drive/My Drive/glove.6B.100d.txt')
for line in f:
  values = line.split()
  word = values[0]
  coefs = asarray(values[1:], dtype='float32')
  embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))
```

NN Model using Glove

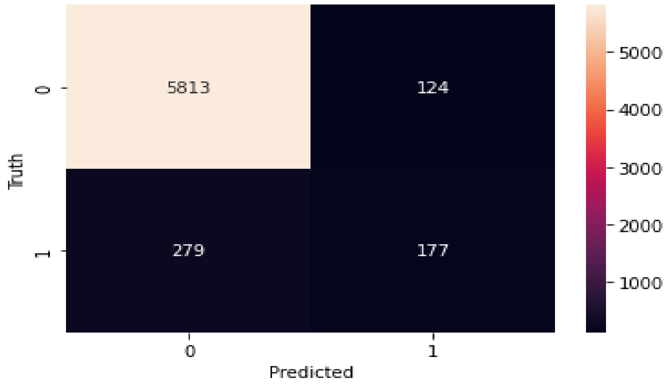
```
# define model
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Embedding
model = Sequential()
e = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=100, trainable=False)
model.add(e)
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
```

```
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# summarize the model
print(model.summary())

```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 100)          3384800

 flatten (Flatten)           (None, 10000)             0

 dense (Dense)               (None, 1)                 10001

=================================================================
Total params: 3,394,801
Trainable params: 10,001
Non-trainable params: 3,384,800
_________________________________________________________________
None
```

Confusion Matrix and Classification Report

	precision	recall	f1-score	support
0	0.95	0.98	0.97	5937
1	0.59	0.39	0.47	456
accuracy			0.94	6393
macro avg	0.77	0.68	0.72	6393
weighted avg	0.93	0.94	0.93	6393

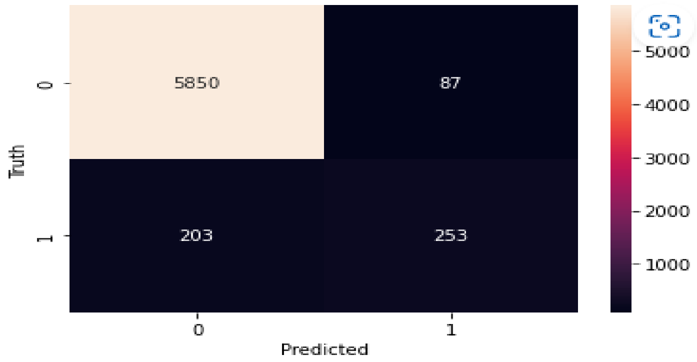


LSTM Model using Glove:

```
#LSTM Model

from keras.layers import Dense, Embedding, LSTM, Bidirectional
from keras.models import Sequential
from keras.layers import Flatten

embedding_layer = Embedding(vocab_size, max_length, weights=[embedding_matrix], input_length=max_length, trainable=False)

embedding_dim =16
input_length = 100
model_lstm = Sequential([embedding_layer,
                         Bidirectional(LSTM(embedding_dim, return_sequences=True)),
                         Bidirectional(LSTM(embedding_dim, )),
                         Dense(6, activation='relu'),
                         Dense(1, activation = 'sigmoid')
                         ])

model_lstm.compile(loss = 'binary_crossentropy', optimizer='adam', metrics = ['accuracy'])
```

Confusion Matrix & Classification Report

```
Text(33.0, 0.5, 'Truth')
```




```
1 print(classification_report(y_test, y_pred))
2
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      5937
           1       0.74      0.55      0.64       456

    accuracy                           0.95      6393
   macro avg       0.86      0.77      0.81      6393
weighted avg       0.95      0.95      0.95      6393
```

Conclusion

- The model performed much better when the data is balanced. Hence, we understood the importance of balanced data while training the model.
- LSTM Model performed much better than Naive Bayes and Neural Network Models.