Please access the notebook using the below mentioned link :

https://colab.research.google.com/drive/1SvGhFoOm95cnoSRLiVZFy4rv5QyCsG8l?usp=sharing

In [ ]:
```
from sklearn import tree
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import recall_score, precision_score, accuracy_sco
from sklearn.model_selection import train_test_split
from gensim.models import Word2Vec
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn import tree
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
import locale
import nltk
import nltk
import numpy as np
import pandas as pd
import re
import re
import seaborn as sns
import string
import string as string
import sys
import warnings
import gensim

nltk.download('stopwords')
pd.set_option('display.max_columns', 500)
sys.setrecursionlimit(5000)
warnings.filterwarnings("ignore")
```

In [ ]:
```
df_train = pd.read_pickle('/content/drive/MyDrive/Northeastern Projects/
```

In [ ]:
```
df_train.head()
```

Out[3]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hat
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	

In [ ]:
```
df_train = df_train.dropna()
```

In [ ]:
```
features = ['sentence_count', 'word_count', 'unique_word_count',
            'length', 'punctuation_count', 'upper_case_count',
            'stopword_count', '#_count', 'unique_word_count_percent',
            'Punctuation_percent', 'ip_count', 'link_count',
            'article_id_count', 'username_count', 'clean_comment']

target_cols = ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult',
```

In [ ]:
```
def tf_idf(X_train):
  vectorizer = TfidfVectorizer(max_features = 500)
  X_train_tfidf = vectorizer.fit_transform(X_train)

  return X_train_tfidf
```

In [ ]:
```
def word2vec_cbow(X_train, vector_size, window):

  # Create CBOW model
  X_train_cleaned = X_train.apply(lambda x: gensim.utils.simple_preproce

  w2v_model = gensim.models.Word2Vec(X_train_cleaned, vector_size = vect
  # Generate aggregated sentence vectors based on the word vectors for e
  # Replace the words in each text message with the learned word vector

  words = set(w2v_model.wv.index_to_key)
  X_train_vect = np.array([np.array([w2v_model.wv[i] for i in ls if i in

  X_train_vect_avg = []
  for v in X_train_vect:
    if v.size:
      X_train_vect_avg.append(v.mean(axis = 0))
    else:
      X_train_vect_avg.append(np.zeros(100, dtype = float))

  return X_train_vect_avg
```

In [ ]:
```
def evaluate_model(actual, predicted):
  '''
  Evaluates the performance of a classification model by calculating and

  Args:
    actual - true class labels of the target variable
    predicted - predicted class labels of the target variable

  Returns:
    None
  '''
  # Calculate recall score
  recall = recall_score(actual, predicted, average = 'macro')

  # Calculate precision score
  precision = precision_score(actual, predicted, average = 'macro')

  # Calculate accuracy score
  accuracy = accuracy_score(actual, predicted)

  # Calculate F1 score
  f1 = f1_score(actual, predicted, average = 'macro')

  return round(recall, 3), round(precision, 3), round(accuracy, 3), roun
```

In [ ]:
```
def master_fit(data, text_embeddings, features, target_cols, models):
  x = data[features]
  metrics_list = []

  X2 = x.drop(['clean_comment'], axis = 1).values
  comment_list = data['clean_comment']

  for i in text_embeddings:

    if i == 'tfidf':
      X = tf_idf(comment_list).toarray()

    elif i == 'word2vec':
      X = word2vec_cbow(comment_list, 100, 2)

    else:
      pass

    final_X = np.hstack((X, X2))

    for j in target_cols:

      y = data[j]
      X_train, X_test, y_train, y_test = train_test_split(final_X, y, te

      clf = list(models.items())[0][1]
      clf.fit(X_train, y_train)
      ypred = clf.predict(X_test)
      r, p, acc, f1 = evaluate_model(y_test, ypred)
      row = [list(models.items())[0][0], str(i), str(j), r, p, acc, f1]
      metrics_list.append(row)

  metric_df = pd.DataFrame(metrics_list, columns = ['model', 'embedding

  return metric_df
```

In [ ]:
```
models = {'Decision Tree': tree.DecisionTreeClassifier() }
```

In [ ]:
```
master_fit(df_train, ['tfidf', 'word2vec'], features, target_cols, mode
```

Out[11]:

	model	embedding	label	Recall	Precision	Accuracy	F1
0	Decision Tree	tfidf	toxic	0.737	0.740	0.910	0.739
1	Decision Tree	tfidf	severe_toxic	0.626	0.630	0.986	0.628
2	Decision Tree	tfidf	obscene	0.787	0.784	0.956	0.785
3	Decision Tree	tfidf	threat	0.624	0.606	0.995	0.614
4	Decision Tree	tfidf	insult	0.711	0.715	0.946	0.713
5	Decision Tree	tfidf	identity_hate	0.641	0.634	0.987	0.637
6	Decision Tree	word2vec	toxic	0.565	0.562	0.844	0.563
7	Decision Tree	word2vec	severe_toxic	0.546	0.544	0.981	0.545
8	Decision Tree	word2vec	obscene	0.553	0.550	0.906	0.551
9	Decision Tree	word2vec	threat	0.514	0.510	0.993	0.512
10	Decision Tree	word2vec	insult	0.553	0.548	0.910	0.550
11	Decision Tree	word2vec	identity_hate	0.520	0.518	0.982	0.519

In [ ]:
```
models = {'Logistic Regression': LogisticRegression() }
```

In [ ]:
```
master_fit(df_train, ['tfidf', 'word2vec'], features, target_cols, mode
```

Out[13]:

	model	embedding	label	Recall	Precision	Accuracy	F1
0	Logistic Regression	tfidf	toxic	0.548	0.901	0.912	0.564
1	Logistic Regression	tfidf	severe_toxic	0.512	0.636	0.990	0.520
2	Logistic Regression	tfidf	obscene	0.715	0.933	0.967	0.784
3	Logistic Regression	tfidf	threat	0.500	0.499	0.997	0.499
4	Logistic Regression	tfidf	insult	0.503	0.675	0.950	0.493
5	Logistic Regression	tfidf	identity_hate	0.504	0.591	0.991	0.506
6	Logistic Regression	word2vec	toxic	0.513	0.792	0.905	0.501
7	Logistic Regression	word2vec	severe_toxic	0.513	0.737	0.990	0.523
8	Logistic Regression	word2vec	obscene	0.505	0.751	0.946	0.497
9	Logistic Regression	word2vec	threat	0.500	0.499	0.997	0.499
10	Logistic Regression	word2vec	insult	0.504	0.688	0.950	0.495
11	Logistic Regression	word2vec	identity_hate	0.500	0.496	0.991	0.498

In [ ]:
```
models = {'Naive Bayes': MultinomialNB() }
```

In [ ]:
```
master_fit(df_train, ['tfidf', 'word2vec'], features, target_cols, mode
```

Out[15]:

	model	embedding	label	Recall	Precision	Accuracy	F1
0	Naive Bayes	tfidf	toxic	0.611	0.539	0.505	0.430
1	Naive Bayes	tfidf	severe_toxic	0.594	0.575	0.981	0.584
2	Naive Bayes	tfidf	obscene	0.622	0.525	0.507	0.397
3	Naive Bayes	tfidf	threat	0.588	0.509	0.970	0.511
4	Naive Bayes	tfidf	insult	0.619	0.523	0.509	0.395
5	Naive Bayes	tfidf	identity_hate	0.639	0.505	0.597	0.387
6	Naive Bayes	word2vec	toxic	0.597	0.534	0.491	0.420
7	Naive Bayes	word2vec	severe_toxic	0.584	0.567	0.981	0.574
8	Naive Bayes	word2vec	obscene	0.606	0.522	0.492	0.387
9	Naive Bayes	word2vec	threat	0.571	0.507	0.969	0.507
10	Naive Bayes	word2vec	insult	0.606	0.520	0.495	0.386
11	Naive Bayes	word2vec	identity_hate	0.617	0.504	0.579	0.379

High Risk Level

In [ ]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM, Dropout, In
import tensorflow as tf
from keras.models import Model
from tensorflow.keras import backend as K
```

In [ ]:
```
def f1(y_true, y_pred):
  def recall_m(y_true, y_pred):
    TP = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    Positives = K.sum(K.round(K.clip(y_true, 0, 1)))

    recall = TP / (Positives+K.epsilon())
    return recall


  def precision_m(y_true, y_pred):
    TP = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    Pred_Positives = K.sum(K.round(K.clip(y_pred, 0, 1)))

    precision = TP / (Pred_Positives+K.epsilon())
    return precision

  precision, recall = precision_m(y_true, y_pred), recall_m(y_true, y_p

  return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

In [ ]:
```
def build_LSTM(data, model_type):

  if model_type == 'LSTM':
    # Define the LSTM model
    model = Sequential()
    model.add(LSTM(128, input_shape = (data.shape[1], 1), return_sequen
    model.add(Dropout(0.2))
    model.add(LSTM(64))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation = 'sigmoid'))
    model.compile(loss = 'binary_crossentropy', optimizer = 'adam', met

    return model

  elif model_type == 'BiDirectionalLSTM':
    # Define the BiDirectional LSTM model
    model = Sequential()
    model.add(Bidirectional(LSTM(128, input_shape = (data.shape[1], 1),
    model.add(Dropout(0.2))
    model.add(Bidirectional(LSTM(64)))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation = 'sigmoid'))
    model.compile(loss = 'binary_crossentropy', optimizer = 'adam', met

    return model

  else:
    pass
```

In [ ]:
```
def master_fit(data, text_embeddings, features, target_cols, model_type
  x = data[features]
  metrics_list = []

  X2 = x.drop(['clean_comment'], axis = 1).values
  comment_list = data['clean_comment']

  for i in text_embeddings:

    if i == 'tfidf':
      X = tf_idf(comment_list).toarray()

    elif i == 'word2vec':
      X = word2vec_cbow(comment_list, 100, 2)

    else:
      pass

    final_X = np.hstack((X, X2))

    for j in target_cols:
      print('Executing', i, 'embedding for', j, 'label')
      y = data[j]
      X_train, X_test, y_train, y_test = train_test_split(final_X, y, te

      X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
      X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

      LSTM_model = build_LSTM(X_train, model_type)
      LSTM_model.fit(X_train, y_train, epochs = 10, batch_size = 256, ve

      # Evaluate the model
      test_metrics = LSTM_model.evaluate(X_test, y_test, batch_size = 2!

      #submission[i] = y_pred_prob
      r, p, acc, f1 = test_metrics[2], test_metrics[3], test_metrics[1]
      row = [str(model_type), str(i), str(j), round(r, 3), round(p, 3),
      metrics_list.append(row)

  metric_df = pd.DataFrame(metrics_list, columns = ['model', 'embedding

  return metric_df
```

In [ ]:
```
master_fit(df_train, ['tfidf', 'word2vec'], features, target_cols, 'LSTI
```

```
Executing tfidf embedding for toxic label
Executing tfidf embedding for severe_toxic label
Executing tfidf embedding for obscene label
Executing tfidf embedding for threat label
Executing tfidf embedding for insult label
Executing tfidf embedding for identity_hate label
Executing word2vec embedding for toxic label
Executing word2vec embedding for severe_toxic label
Executing word2vec embedding for obscene label
Executing word2vec embedding for threat label
Executing word2vec embedding for insult label
Executing word2vec embedding for identity_hate label
```

Out[16]:

	model	embedding	label	Recall	Precision	Accuracy	F1
0	LSTM	tfidf	toxic	0.108	0.643	0.908	0.178
1	LSTM	tfidf	severe_toxic	0.000	0.000	0.990	0.000
2	LSTM	tfidf	obscene	0.024	0.504	0.946	0.044
3	LSTM	tfidf	threat	0.000	0.000	0.997	0.000
4	LSTM	tfidf	insult	0.009	0.500	0.950	0.015
5	LSTM	tfidf	identity_hate	0.000	0.000	0.991	0.000
6	LSTM	word2vec	toxic	0.115	0.620	0.908	0.189
7	LSTM	word2vec	severe_toxic	0.000	0.000	0.990	0.000
8	LSTM	word2vec	obscene	0.017	0.570	0.946	0.032
9	LSTM	word2vec	threat	0.000	0.000	0.997	0.000
10	LSTM	word2vec	insult	0.010	0.429	0.950	0.017
11	LSTM	word2vec	identity_hate	0.000	0.000	0.991	0.000

In [ ]:
```
master_fit(df_train, ['tfidf', 'word2vec'], features, target_cols, 'BiD:
```

```
Executing tfidf embedding for toxic label
Executing tfidf embedding for severe_toxic label
Executing tfidf embedding for obscene label
Executing tfidf embedding for threat label
Executing tfidf embedding for insult label
Executing tfidf embedding for identity_hate label
Executing word2vec embedding for toxic label
Executing word2vec embedding for severe_toxic label
Executing word2vec embedding for obscene label
Executing word2vec embedding for threat label
Executing word2vec embedding for insult label
Executing word2vec embedding for identity_hate label
```

Out[14]:

	model	embedding	label	Recall	Precision	Accuracy	F1
0	BiDirectionalLSTM	tfidf	toxic	0.114	0.620	0.908	0.187
1	BiDirectionalLSTM	tfidf	severe_toxic	0.000	0.000	0.990	0.000
2	BiDirectionalLSTM	tfidf	obscene	0.008	0.524	0.946	0.014
3	BiDirectionalLSTM	tfidf	threat	0.000	0.000	0.997	0.000
4	BiDirectionalLSTM	tfidf	insult	0.011	0.475	0.950	0.018
5	BiDirectionalLSTM	tfidf	identity_hate	0.000	0.000	0.991	0.000
6	BiDirectionalLSTM	word2vec	toxic	0.171	0.536	0.906	0.254
7	BiDirectionalLSTM	word2vec	severe_toxic	0.011	0.857	0.990	0.014
8	BiDirectionalLSTM	word2vec	obscene	0.018	0.573	0.946	0.033
9	BiDirectionalLSTM	word2vec	threat	0.000	0.000	0.997	0.000
10	BiDirectionalLSTM	word2vec	insult	0.006	0.630	0.950	0.011
11	BiDirectionalLSTM	word2vec	identity_hate	0.000	0.000	0.991	0.000

In [ ]:
```
def get_coefs(word,*arr):
  return word, np.asarray(arr, dtype = 'float32')
```

In [ ]:
```
def build_LSTM(embd, model_type):
  # Define the LSTM model

  if model_type == 'LSTM':
    inp = Input(shape = (100,))
    x = Embedding(500, 100, weights = [embd])(inp)
    x = LSTM(128, return_sequences = True)(x)
    x = Dropout(0.2)(x)
    x = LSTM(128)(x)
    x = Dropout(0.2)(x)
    x = Dense(1, activation = "sigmoid")(x)

    model = Model(inputs = inp, outputs = x)
    model.compile(loss = 'binary_crossentropy', optimizer = 'adam', met

    return model

  elif model_type == 'BiDirectionalLSTM':
    inp = Input(shape = (100,))
    x = Embedding(500, 100, weights = [embd])(inp)
    x = Bidirectional(LSTM(128, return_sequences = True))(x)
    x = Dropout(0.2)(x)
    x = Bidirectional(LSTM(128))(x)
    x = Dropout(0.2)(x)
    x = Dense(1, activation = "sigmoid")(x)

    model = Model(inputs = inp, outputs = x)
    model.compile(loss = 'binary_crossentropy', optimizer = 'adam', met

    return model

  else:
    pass
```

In [ ]:
```
atures, target_cols, model_type):

Frame([])


comment'], axis = 1).values
'clean_comment']





r(num_words = max_features)
ts(list(comment_list))
 = tokenizer.texts_to_sequences(comment_list)
list_tokenized_train, maxlen = maxlen)

ict(get_coefs(*o.strip().split()) for o in open('/content/drive/MyDrive/

embeddings_index.values())
all_embs.mean(), all_embs.std()

er.word_index
eatures, len(word_index))
p.random.normal(emb_mean, emb_std, (nb_words, embed_size))

index.items():
es: continue
 embeddings_index.get(word)
r is not None: embedding_matrix[i] = embedding_vector


:


_train, y_test = train_test_split(X_t, y, test_size = 0.33, random_state

reshape(X_train.shape[0], X_train.shape[1], 1)
shape(X_test.shape[0], X_test.shape[1], 1)

_LSTM(embedding_matrix, str(model_type))
rain, y_train, epochs = 10, batch_size = 256, verbose = 0)

el
M_model.evaluate(X_test, y_test, batch_size = 256, verbose = 0)

_pred_prob
st_metrics[2], test_metrics[3], test_metrics[1], test_metrics[4]
ype), 'Glove', str(j), r, p, acc, f1]
d(row)

rame(metrics_list, columns = ['model', 'embedding', 'label', 'Recall', '

head())
```

In [ ]:
```
glove_emd(df_train, features, target_cols, "LSTM")
```

Out[32]:

	model	embedding	label	Recall	Precision	Accuracy	F1
0	LSTM	Glove	toxic	0.489475	0.841963	0.941852	0.612171
1	LSTM	Glove	severe_toxic	0.315589	0.378132	0.987979	0.281925
2	LSTM	Glove	obscene	0.585659	0.865796	0.972844	0.688993
3	LSTM	Glove	threat	0.296053	0.483871	0.997057	0.143204
4	LSTM	Glove	insult	0.489217	0.702717	0.963976	0.562041
5	LSTM	Glove	identity_hate	0.307856	0.501730	0.991075	0.275827

In [ ]:
```
glove_emd(df_train, features, target_cols, "BiDirectionalLSTM")
```

Out[44]:

	model	embedding	label	Recall	Precision	Accuracy	F1
0	BiDirectionalLSTM	Glove	toxic	0.493606	0.837450	0.941871	0.613573
1	BiDirectionalLSTM	Glove	severe_toxic	0.214829	0.478814	0.989821	0.227986
2	BiDirectionalLSTM	Glove	obscene	0.588131	0.852971	0.972407	0.685463
3	BiDirectionalLSTM	Glove	threat	0.276316	0.461538	0.996981	0.133171
4	BiDirectionalLSTM	Glove	insult	0.519485	0.683425	0.963805	0.577198
5	BiDirectionalLSTM	Glove	identity_hate	0.199575	0.591195	0.991606	0.201707