```
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
nltk.download('stopwords')
from tensorflow.keras.preprocessing.text import Tokenizer

from sklearn.preprocessing import LabelEncoder
```

```
     [nltk_data] Downloading package stopwords to /root/nltk_data...
     [nltk_data]   Package stopwords is already up-to-date!
```

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras import datasets, layers, models, preprocessing
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN
from nltk.corpus import stopwords
import pandas as pd
import numpy as np
import seaborn as sb
```

```
df = pd.read_csv('youtoxic_english_1000.csv')
labels = []
y = df.IsToxic
df_y = pd.DataFrame(y, columns=['IsToxic'])

sb.catplot(x="IsToxic", kind='count', data=df_y)
```

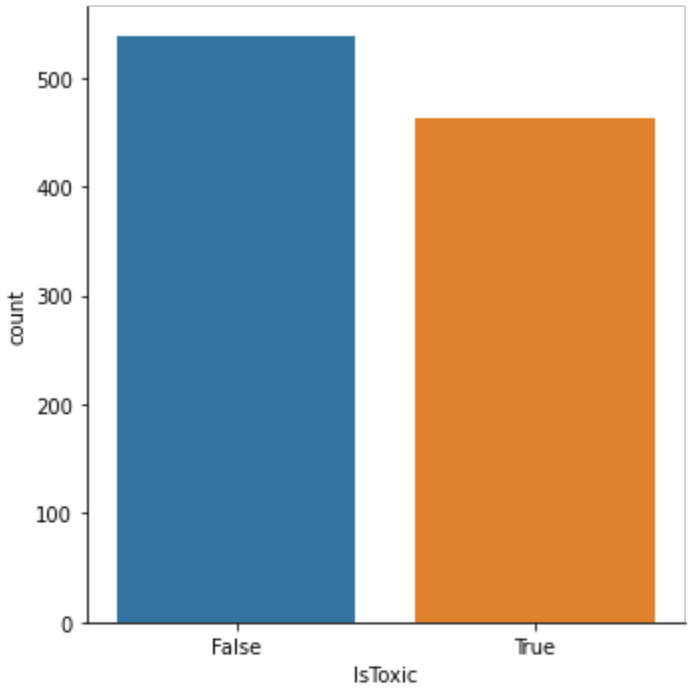
```
i = np.random.rand(len(df)) < 0.8
train = df[i]
test = df[~i]
num_labels = 2
vocab_size = 25000
batch_size = 100

tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.Text)

x_train = tokenizer.texts_to_matrix(train.Text, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.Text, mode='tfidf')
df.apply(LabelEncoder().fit_transform)

encoder = LabelEncoder()
```

```
y_train = encoder.transform(train.IsToxic)
y_test = encoder.transform(test.IsToxic)
```



https://www.kaggle.com/datasets/reihanenamdari/youtube-toxicity-data?resource=download

This dataset contains 1000 comments with labels of IsToxic, IsAbusive, IsThreat, IsProvocative IsObscene, IsHatespeech, IsRacist, IsNationalist, sSexist, IsHomophobic, IsReligiousHate, IsRadicalism. Despite the small dataset, most of the Text in the dataset is fairly long.

In the collab script. We first start by loading in the dataset using pandas. I went ahead and plotted for all 11 columns, but there was alot invalided fields within those columns. So I had to do some data cleaning for those fields. After plotting, we could see that IsToxic has the most count out of all of the comments, followed by IsAbusive.

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-37-258ae25c3c08> in <module>
----> 1 sb.catplot(x="IsThreat", kind='count', data=pd.DataFrame(y, columns=
['IsThreat']))

                              2 frames
/usr/local/lib/python3.8/dist-packages/seaborn/categorical.py in
establish_colors(self, color, palette, saturation)
    317         # Determine the gray color to use for the lines framing the plot
    317         # Determine the gray color to use for the lines framing the plot
    318         light_vals = [colorsys.rgb_to_hls(*c)[1] for c in rgb_colors]
--> 319         lum = min(light_vals) * .6
    320         gray = mpl.colors.rgb2hex((lum, lum, lum))
    321 

ValueError: min() arg is an empty sequence
```

```
model = models.Sequential()
model.add(layers.Dense(16, input_dim=vocab_size, kernel_initializer='normal', activation='
#model.add(layers.Dense(8, activation='relu'))

model.add(layers.Dense(1, kernel_initializer='normal', activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
from keras.utils import np_utils

#y = np_utils.to_categorical(y_train, 1)
#y = np_utils.to_categorical(y_train, 2)
print(x_train.shape)
#WOY = np_utils.to_categorical(OY,5)

#y_trainEnc = y.sort_indices()
#y_testEnc = OY.sort_indices()

model.summary()


Seq = model.fit(x_train,y_train,batch_size=batch_size,epochs=15,validation_split=0.1,verbo
#model.fit(x,y,batch_size=32,epochs=1,callbacks=callbacks,validation_data=(OX,OY))
```

```
(796, 25000)
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_13 (Dense)            (None, 16)                400016    
                                                                 
 dense_14 (Dense)            (None, 1)                 17        
                                                                 
=================================================================
Total params: 400,033
Trainable params: 400,033
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
```

```
    Epoch 1/15
    8/8 [==============================] - 2s 43ms/step - loss: 0.6944 - accuracy: 0.487
    Epoch 2/15
    8/8 [==============================] - 0s 21ms/step - loss: 0.6687 - accuracy: 0.659
    Epoch 3/15
    8/8 [==============================] - 0s 18ms/step - loss: 0.6436 - accuracy: 0.750
    Epoch 4/15
    8/8 [==============================] - 0s 20ms/step - loss: 0.6136 - accuracy: 0.819
    Epoch 5/15
    8/8 [==============================] - 0s 22ms/step - loss: 0.5763 - accuracy: 0.871
    Epoch 6/15
    8/8 [==============================] - 0s 20ms/step - loss: 0.5339 - accuracy: 0.906
    Epoch 7/15
    8/8 [==============================] - 0s 21ms/step - loss: 0.4875 - accuracy: 0.928
    Epoch 8/15
    8/8 [==============================] - 0s 22ms/step - loss: 0.4410 - accuracy: 0.946
    Epoch 9/15
    8/8 [==============================] - 0s 20ms/step - loss: 0.3961 - accuracy: 0.960
    Epoch 10/15
    8/8 [==============================] - 0s 21ms/step - loss: 0.3543 - accuracy: 0.969
    Epoch 11/15
    8/8 [==============================] - 0s 22ms/step - loss: 0.3166 - accuracy: 0.974
    Epoch 12/15
    8/8 [==============================] - 0s 22ms/step - loss: 0.2828 - accuracy: 0.974
    Epoch 13/15
    8/8 [==============================] - 0s 25ms/step - loss: 0.2532 - accuracy: 0.980
    Epoch 14/15
    8/8 [==============================] - 0s 18ms/step - loss: 0.2269 - accuracy: 0.981
    Epoch 15/15
    8/8 [==============================] - 0s 21ms/step - loss: 0.2040 - accuracy: 0.983
```

```
score =  model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)

print('Accuracy: ', score[1])
```

```
    3/3 [==============================] - 0s 6ms/step - loss: 0.6371 - accuracy: 0.6814
    Accuracy:  0.6813725233078003
```

I went ahead and did the Sequential model, but Multi-Label classification is a bit more difficult than I intended, so I scraped all of the columns and stuck with IsToxic as the only target function. The Sequential performed extremely well, at first I had 4 layers ranging from 64, 32, 16, 1, which training accuracy was really well, however the evaluation was very low, maybe due to overfitting. I also tried some loss function, activation functions, and optimizer, which the binary_crossentropy, sigmoidand rmsprop, performed the best. After a few more run, I settled on a 16, 1 layer group which had the best accuracy of .68, a little disappointing but no matter.

```
model1 = Sequential()
model1.add(layers.Embedding(1000, 32))
model1.add(layers.SimpleRNN(32))
```

```
model1.add(layers.SimpleRNN(32))
model1.add(layers.Dense(1, activation='sigmoid'))
model1.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

```
batch_size = 32
maxlen = 800
train_data = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
test_data = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
model1.summary()

RNN = model1.fit(train_data,y_train,batch_size=batch_size,epochs=10,validation_split=0.2,v
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, None, 32)          32000

 simple_rnn (SimpleRNN)      (None, 32)                2080

 dense_10 (Dense)            (None, 1)                 33

=================================================================
Total params: 34,113
Trainable params: 34,113
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
20/20 [==============================] - 6s 213ms/step - loss: 0.6944 - accuracy: 0.5
Epoch 2/10
20/20 [==============================] - 4s 201ms/step - loss: 0.6930 - accuracy: 0.5
Epoch 3/10
20/20 [==============================] - 4s 199ms/step - loss: 0.6937 - accuracy: 0.5
Epoch 4/10
20/20 [==============================] - 4s 199ms/step - loss: 0.6941 - accuracy: 0.4
Epoch 5/10
20/20 [==============================] - 4s 197ms/step - loss: 0.6941 - accuracy: 0.4
Epoch 6/10
20/20 [==============================] - 4s 199ms/step - loss: 0.6975 - accuracy: 0.4
Epoch 7/10
20/20 [==============================] - 4s 197ms/step - loss: 0.6938 - accuracy: 0.5
Epoch 8/10
20/20 [==============================] - 4s 197ms/step - loss: 0.6950 - accuracy: 0.4
Epoch 9/10
20/20 [==============================] - 5s 273ms/step - loss: 0.6934 - accuracy: 0.5
Epoch 10/10
20/20 [==============================] - 4s 198ms/step - loss: 0.6957 - accuracy: 0.4
```

```
from sklearn.metrics import classification_report
```

```
pred = model1.predict(test_data)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
7/7 [==============================] - 1s 46ms/step
              precision    recall  f1-score   support

           0       0.55      1.00      0.71       115
           1       0.00      0.00      0.00        94

    accuracy                           0.55       209
   macro avg       0.28      0.50      0.35       209
weighted avg       0.30      0.55      0.39       209

/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: Unde
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: Unde
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: Unde
  _warn_prf(average, modifier, msg_start, len(result))
```

After that is the RNN model, I went with an Embedding layer, then the RNN layer of size 32 then to the output layer. I decided the max len for each step of the RNN would be 15 as I believe after 15, there is a good part you could understand if the sentence is toxic enough, but also I didn't want to slow down the RNN model. In the end RNN models perform very poorly with an accuracy rate of .55. Most likely due to the size of the dataset.

```
train_data = preprocessing.sequence.pad_sequences(x_train, maxlen=15)
test_data = preprocessing.sequence.pad_sequences(x_test, maxlen=15)

model2 = models.Sequential()
model2.add(layers.Embedding(5000, 8, input_length=15))
model2.add(layers.Flatten())
model2.add(layers.Dense(32, activation='relu'))
model2.add(layers.Dense(8, activation='relu'))
model2.add(layers.Dense(1, activation='sigmoid'))
model2.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model2.summary()

Emb = model2.fit(train_data, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 embedding_2 (Embedding)     (None, 15, 8)             40000

 flatten_2 (Flatten)         (None, 120)               0

 dense_7 (Dense)             (None, 32)                3872

 dense_8 (Dense)             (None, 8)                 264

 dense_9 (Dense)             (None, 1)                 9

=================================================================
Total params: 44,145
Trainable params: 44,145
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
20/20 [==============================] - 1s 13ms/step - loss: 0.6938 - acc: 0.5016 -
Epoch 2/10
20/20 [==============================] - 0s 5ms/step - loss: 0.6936 - acc: 0.4921 -
Epoch 3/10
20/20 [==============================] - 0s 5ms/step - loss: 0.6933 - acc: 0.5047 -
Epoch 4/10
20/20 [==============================] - 0s 5ms/step - loss: 0.6932 - acc: 0.5111 -
Epoch 5/10
20/20 [==============================] - 0s 5ms/step - loss: 0.6932 - acc: 0.5111 -
Epoch 6/10
20/20 [==============================] - 0s 5ms/step - loss: 0.6931 - acc: 0.5111 -
Epoch 7/10
20/20 [==============================] - 0s 5ms/step - loss: 0.6931 - acc: 0.5111 -
Epoch 8/10
20/20 [==============================] - 0s 4ms/step - loss: 0.6933 - acc: 0.5111 -
Epoch 9/10
20/20 [==============================] - 0s 4ms/step - loss: 0.6931 - acc: 0.5111 -
Epoch 10/10
20/20 [==============================] - 0s 5ms/step - loss: 0.6932 - acc: 0.5111 -
```

```python
#score =  model2.evaluate(test_data, y_test, batch_size=32, verbose=1)
pred = model2.predict(test_data)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-58-393c43ec3065> in <module>
      1 #score =  model2.evaluate(test_data, y_test, batch_size=32, verbose=1)
----> 2 pred = model2.predict(x_train)

                                  1 frames

/usr/local/lib/python3.8/dist-packages/keras/engine/training.py in
tf__predict_function(iterator)
     13                 try:
     14                     do_return = True
---> 15                     retval_ = ag__.converted_call(ag__.ld(step_function),
---> 15                     retval_ = ag__.converted_call(ag__.ld(step_function),
(ag__.ld(self), ag__.ld(iterator)), None, fscope)
     16                 except:
     17                     do_return = False

ValueError: in user code:

    File "/usr/local/lib/python3.8/dist-packages/keras/engine/training.py", line
1845, in predict_function  *
        return step_function(self, iterator)
    File "/usr/local/lib/python3.8/dist-packages/keras/engine/training.py", line
1834, in step_function  **
        outputs = model.distribute_strategy.run(run_step, args=(data,))
    File "/usr/local/lib/python3.8/dist-packages/keras/engine/training.py", line
1823, in run_step  **
        outputs = model.predict_step(data)
    File "/usr/local/lib/python3.8/dist-packages/keras/engine/training.py", line
1791, in predict_step
        return self(x, training=False)
    File "/usr/local/lib/python3.8/dist-packages/keras/utils/traceback_utils.py",
line 67, in error_handler
        raise e.with_traceback(filtered_tb) from None
    File "/usr/local/lib/python3.8/dist-packages/keras/engine/input_spec.py", line
```

Next is the Embedding layer, the layer would contain a 5000 vocab for the embedding layer with 15 input length. After the alyer would flattenm the nodes and have a 3 layer 32, 8, 1 setup. Again, the embedding layter didn't do too well with only an accuracy of .51 in train and on the evaluation.

In all, due to the size of the dataset, the models couldn't shine as the DL require a large amount of data unfortunately. But the best model of them all is the sequential model.

Next is the Embedding layer, the layer would
embedding layer with 15 input length. After t
and have a 3 layer 32, 8, 1 setup. Again, the
well with only an accuracy of .51 in train ar

In all, due to the size of the dataset, the r
require a large amount of data unfortunately
the sequential model.