Applied Machine Learning

Project

Group Members : Pragya Shukla and Sumeet Suley

Topic: Identify Toxicity in Comments

1. Description

- **Toxicity** is defined as anything rude, disrespectful or otherwise likely to make someone leave a discussion.
- A main area of focus is machine learning models could be to identify toxicity in online conversations.
- If these toxic contributions can be identified, we could have a safer, more collaborative internet.

2. Dataset:

- The dataset for this contains text that may be considered profane, vulgar, or offensive.

3. Evaluation:

- `Area under the ROC curve` between the predicted probability and the observed target.

4. Elaboration:

- `comment_text` : This contains the text of a comment which has been classified as toxic or non-toxic (0...1 in the toxic column). The data set's comments are entirely in english and come either from Civil Comments or Wikipedia talk page edits.

5. Prediction:

- We are predicting the probability that a comment is toxic. A toxic comment would receive a 1.0. A benign, non-toxic comment would receive a 0.0. In the test set, all comments are classified as either a 1.0 or a 0.0.

Install and Import Libraries

In [1]:
```
!pip install -q pyicu
```

```
!pip install -q pycld2
!pip install -q polyglot
!pip install -q pyyaml h5py  # Required to save models in HDF5 format
```

In [2]:

```
import os
import re
import pandas as pd
import numpy as np
import tqdm
import transformers
import tensorflow as tf

from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix
from sklearn.model_selection import train_test_split

from polyglot.detect import Detector

import matplotlib.pyplot as plt
```

In [3]:

```
print(tf.__version__)
print(transformers.__version__)
print(tf.keras.__version__)
```

```
2.4.1
4.5.1
2.4.0
```

Data Preprocessing

Here we are preprocessing the dataset by extracting only the English text to perform our next tasks.

In [5]:

```
def get_language(text):
    return Detector(
        "".join(x for x in text if x.isprintable()), quiet=True
    ).languages[0].name

PATH =  "jigsaw-multilingual-toxic-comment-classification"
FILES = os.listdir(PATH)
print(FILES)

TRAIN_PATH = os.path.join(PATH, 'jigsaw-toxic-comment-train.csv')
data = pd.read_csv(TRAIN_PATH)

data["lang"] = data["comment_text"].apply(get_language)
data = data[data['lang'] == 'English']
```

```
['sample_submission.csv', 'validation-processed-seqlen128.csv', 'test-processed
-seqlen128.csv', 'jigsaw-unintended-bias-train-processed-seqlen128.csv', 'valid
ation.csv', 'jigsaw-toxic-comment-train.csv', 'test.csv', 'jigsaw-unintended-bi
as-train.csv', 'jigsaw-toxic-comment-train-processed-seqlen128.csv']
```

See how the dataset looks like now.

In [6]:

```
data.head()
```

Out[6]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_h

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_h
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	

Lets see the distribution of toxic and non-toxic comments.

In [7]:
```
data.toxic.value_counts()
```

Out[7]:
```
0    199700
1     20953
Name: toxic, dtype: int64
```

Split data to Train and Test

In [8]:
```
X = data[['comment_text']]
y = data[['toxic']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

y_test = y_test.toxic.values
y_train = y_train.toxic.values

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(176522, 1)
(176522,)
(44131, 1)
(44131,)
```

Tokenizing the data

Here we are using pre-trained BERT model and have tokenized the text using Transformers Autotokenizer from pretrained model. We have taken sequence length to be 128, in case its not 128, we have used Padding to make it of equal length and finally we have shuffled the data and created out train and test dataset.

In [9]:
```
def map_func(input_ids, masks, labels):
    return {
        'input_ids': input_ids,
        'attention_mask': masks
    }, labels

PRE_TRAINED_MODEL_NAME = 'bert-base-cased'
tokenizer = transformers.AutoTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)

SEQ_LEN = 128
X_train_ids = np.zeros((len(X_train), SEQ_LEN))
X_train_mask = np.zeros((len(X_train), SEQ_LEN))

X_test_ids = np.zeros((len(X_test), SEQ_LEN))
X_test_mask = np.zeros((len(X_test), SEQ_LEN))

for i, sequence in enumerate(X_train['comment_text']):
    tokens = tokenizer.encode_plus(
        sequence, max_length=SEQ_LEN,
        truncation=True, padding='max_length',
        add_special_tokens=True, return_token_type_ids=False,
        return_attention_mask=True, return_tensors='tf'
    )
    X_train_ids[i, :], X_train_mask[i, :] = tokens['input_ids'], tokens['attent

for i, sequence in enumerate(X_test['comment_text']):
    tokens = tokenizer.encode_plus(
        sequence, max_length=SEQ_LEN,
        truncation=True, padding='max_length',
        add_special_tokens=True, return_token_type_ids=False,
        return_attention_mask=True, return_tensors='tf'
    )
    X_test_ids[i, :], X_test_mask[i, :] = tokens['input_ids'], tokens['attentio

train_dataset = tf.data.Dataset.from_tensor_slices((X_train_ids, X_train_mask,
test_dataset = tf.data.Dataset.from_tensor_slices((X_test_ids, X_test_mask, y_t

train_dataset = train_dataset.map(map_func)
test_dataset = test_dataset.map(map_func)

train_dataset = train_dataset.shuffle(100000).batch(32, drop_remainder=True)
test_dataset = test_dataset.shuffle(100000).batch(32, drop_remainder=True)
```

Downloading: 100% 570/570 [00:00<00:00, 19.8kB/s]

Downloading: 100% 213k/213k [00:00<00:00, 684kB/s]

Downloading: 100% 436k/436k [00:00<00:00, 597kB/s]

Downloading: 100% 29.0/29.0 [00:00<00:00, 1.07kB/s]

Model Building

Here we have used pre-trained BERT model to initialize transfer learning. We have

used Relu function in the hidden layer and sigmoid function in the final layer. We have used Adam optimizer, AUC metric for evaluation and binary cross entropy as loss function. We have trained it for 10 epochs with batch size 4096, let's see what's the result.

In [10]:
```
bert = transformers.TFAutoModel.from_pretrained(PRE_TRAINED_MODEL_NAME)

input_ids = tf.keras.layers.Input(shape=(SEQ_LEN, ), name='input_ids', dtype='i
mask = tf.keras.layers.Input(shape=(SEQ_LEN, ), name='attention_mask', dtype='i

embeddings = bert.bert(input_ids, attention_mask=mask)[1]

X = tf.keras.layers.Dense(1024, activation='relu')(embeddings)
y = tf.keras.layers.Dense(1, activation='sigmoid', name='outputs')(X)

model = tf.keras.Model(inputs=[input_ids, mask], outputs=y)

model.layers[2].trainable = False

model.compile(
    optimizer=tf.keras.optimizers.Adam(lr=1e-5, decay=1e-6),
    loss='binary_crossentropy',
    metrics=[tf.keras.metrics.AUC(name='AUC')]
)

r = model.fit(
    train_dataset,
    validation_data=(test_dataset),
    epochs=10,
    batch_size=4096
)
```

Downloading: 100% 527M/527M [00:28<00:00, 19.0MB/s]

```
Some layers from the model checkpoint at bert-base-cased were not used when ini
tializing TFBertModel: ['nsp___cls', 'mlm___cls']
- This IS expected if you are initializing TFBertModel from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing TFBertModel from the checkpoint
of a model that you expect to be exactly identical (initializing a BertForSeque
nceClassification model from a BertForSequenceClassification model).
All the layers of TFBertModel were initialized from the model checkpoint at ber
t-base-cased.
If your task is similar to the task the model of the checkpoint was trained on,
you can already use TFBertModel for predictions without further training.
Epoch 1/10
5516/5516 [==============================] - 1066s 192ms/step - loss: 0.2681 -
AUC: 0.7887 - val_loss: 0.2191 - val_AUC: 0.8768
Epoch 2/10
5516/5516 [==============================] - 1058s 192ms/step - loss: 0.2197 -
AUC: 0.8723 - val_loss: 0.2087 - val_AUC: 0.8965
Epoch 3/10
5516/5516 [==============================] - 1059s 192ms/step - loss: 0.2065 -
AUC: 0.8916 - val_loss: 0.1960 - val_AUC: 0.9050
Epoch 4/10
5516/5516 [==============================] - 1058s 192ms/step - loss: 0.2014 -
AUC: 0.8983 - val_loss: 0.1916 - val_AUC: 0.9098
Epoch 5/10
5516/5516 [==============================] - 1055s 191ms/step - loss: 0.1939 -
```

```
AUC: 0.9035 - val_loss: 0.1867 - val_AUC: 0.9126
Epoch 6/10
5516/5516 [==============================] - 1053s 191ms/step - loss: 0.1920 -
AUC: 0.9059 - val_loss: 0.1904 - val_AUC: 0.9139
Epoch 7/10
5516/5516 [==============================] - 1053s 191ms/step - loss: 0.1907 -
AUC: 0.9084 - val_loss: 0.1846 - val_AUC: 0.9159
Epoch 8/10
5516/5516 [==============================] - 1053s 191ms/step - loss: 0.1899 -
AUC: 0.9061 - val_loss: 0.1808 - val_AUC: 0.9172
Epoch 9/10
5516/5516 [==============================] - 1053s 191ms/step - loss: 0.1867 -
AUC: 0.9117 - val_loss: 0.1814 - val_AUC: 0.9180
Epoch 10/10
5516/5516 [==============================] - 1054s 191ms/step - loss: 0.1849 -
AUC: 0.9134 - val_loss: 0.1794 - val_AUC: 0.9197
```

Evaluation

We have got the loss as 0.1793 and AUC score of 0.9197.

In [11]:

```
model.evaluate(test_dataset)
```

```
1379/1379 [==============================] - 199s 144ms/step - loss: 0.1793 - A
UC: 0.9197
```

Out[11]:

```
[0.1793288141489029, 0.919731855392456]
```

Let's plot the curves for both of them.

In [12]:

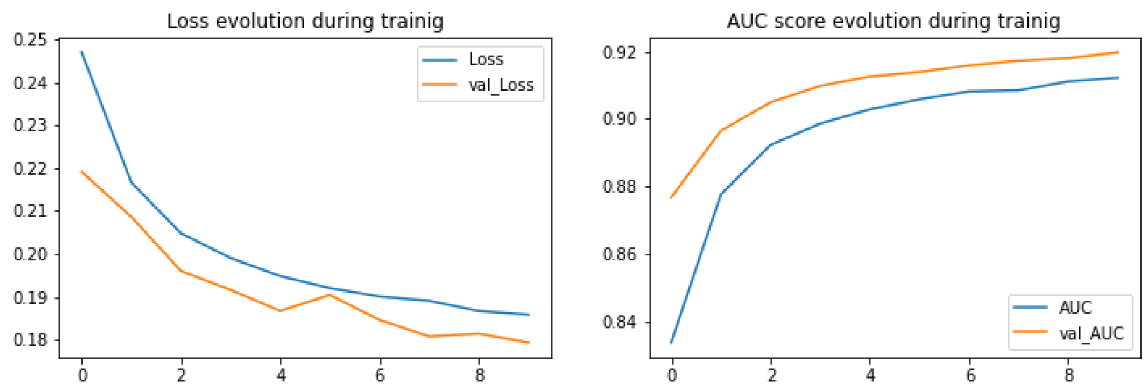
```
def plot_learning_evolution(r):
    plt.figure(figsize=(12, 8))

    plt.subplot(2, 2, 1)
    plt.plot(r.history['loss'], label='Loss')
    plt.plot(r.history['val_loss'], label='val_Loss')
    plt.title('Loss evolution during trainig')
    plt.legend()

    plt.subplot(2, 2, 2)
    plt.plot(r.history['AUC'], label='AUC')
    plt.plot(r.history['val_AUC'], label='val_AUC')
    plt.title('AUC score evolution during trainig')
    plt.legend();
```

In [13]:

```
plot_learning_evolution(r)
```



Saving the model

In [15]:
```
!pip install pyyaml h5py  # Required to save models in HDF5 format
```

```
Requirement already satisfied: pyyaml in /opt/conda/lib/python3.7/site-packages
(5.3.1)
Requirement already satisfied: h5py in /opt/conda/lib/python3.7/site-packages
(2.10.0)
Requirement already satisfied: numpy>=1.7 in /opt/conda/lib/python3.7/site-pack
ages (from h5py) (1.19.5)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (f
rom h5py) (1.15.0)
```

In [16]:
```
tf.keras.models.save_model(model, "hate_speech_10_epochs.hdf5")
```

In [17]:
```
hdf5_model = tf.keras.models.load_model("hate_speech_10_epochs.hdf5")
hdf5_model.summary()
```

```
Model: "model"
________________________________________________________________________________
___________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
===================
input_ids (InputLayer)          [(None, 128)]        0
________________________________________________________________________________
___________________
attention_mask (InputLayer)     [(None, 128)]        0
________________________________________________________________________________
___________________
bert (TFBertMainLayer)          TFBaseModelOutputWit 108310272   input_ids[0]
[0]
                                                                 attention_mask
[0][0]
________________________________________________________________________________
___________________
dense (Dense)                   (None, 1024)         787456      bert[0][1]
________________________________________________________________________________
___________________
outputs (Dense)                 (None, 1)            1025        dense[0][0]
================================================================================
===================
Total params: 109,098,753
Trainable params: 788,481
Non-trainable params: 108,310,272
________________________________________________________________________________
___________________
```

A function to preprocess user's text.

In [19]:
```
def prep_sentence(sentence):
    tokens = tokenizer.encode_plus(
        sentence, max_length=SEQ_LEN,
        truncation=True, padding='max_length',
        add_special_tokens=True, return_token_type_ids=False,
        return_attention_mask=True, return_tensors='tf'
    )
    return {
        'input_ids': tf.cast(tokens['input_ids'], tf.float64),
```

```
        'attention_mask': tf.cast(tokens['attention_mask'], tf.float64)
    }
```

Toxic Comments

In [20]:

```
toxic_speechs = [
    'COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK',
    'MEL GIBSON IS A NAZI BITCH WHO MAKES SHITTY MOVIES. HE HAS SO MUCH BUTTSEX
    "A block ohhhhhhhhhhhhh nooooooooooooo I'm soooo like gonna cry and like sh
    "it is a constructive edit you idiot, every kid of every age should know th
    'honestly ==\nyou need to crawl under a rock and DIE YOU FAT BASTARD\n\n=='
]
```

We know that a toxic comment would receive a 1.0 and a benign, non-toxic comment would receive a 0.0, here is the set of predicted values for this toxic comment.

In [22]:

```
for speech in toxic_speechs:
    prediction = hdf5_model.predict(prep_sentence(speech))
    print(prediction)
```

```
[[0.5430674]]
[[0.38300273]]
[[0.40779674]]
[[0.63300586]]
[[0.5390873]]
```

Non-Toxic Comments

In [23]:

```
non_toxic_speechs = [
    "Gale, you're living proof why wikipedia should NEVER be trusted as fact. I
    'EastEnders Manual of Style \n\nHello, just wanted you to be aware of the E
    'You need to provide high-quality secondary sources (e.g., not original pub
    "I appreciate your responses, guys. I take the recommendation as an admin a
    "Stop reinserting harrassing content on WP:ANI \n\nStop readding this mater
]
```

Predicted values for non-toxic comments.

In [25]:

```
for speech in non_toxic_speechs:
    prediction = hdf5_model.predict(prep_sentence(speech))
    print(prediction)
```

```
[[0.17475249]]
[[0.01649178]]
[[0.00066787]]
[[0.013166]]
[[0.05405284]]
```

Conclusion:

In summary, there are many exciting research directions that transfer learning offers and -- in particular -- many applications that are in need of models that can transfer knowledge to new tasks and adapt to new domains. This was our first attempt to implement transfer learning for NLP using pre-trained model BERT in this project.