Hate Speech - Sentiment Classification

```
!pip install datasets # we are installing huggingface datasets
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/s
Collecting datasets
  Downloading datasets-2.11.0-py3-none-any.whl (468 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 468.7/468.7 kB 5.6 MB/s eta 0:00:00
Collecting aiohttp
  Downloading aiohttp-3.8.4-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.0 MB
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 28.4 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.9/dist-packages (from d
Collecting huggingface-hub<1.0.0,>=0.11.0
  Downloading huggingface_hub-0.13.4-py3-none-any.whl (200 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 200.1/200.1 kB 12.8 MB/s eta 0:00:00
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.9/dist-packages (fro
Requirement already satisfied: packaging in /usr/local/lib/python3.9/dist-packages (from dat
Requirement already satisfied: fsspec[http]>=2021.11.1 in /usr/local/lib/python3.9/dist-pack
Requirement already satisfied: pandas in /usr/local/lib/python3.9/dist-packages (from datase
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.9/dist-packages (from d
Collecting multiprocess
  Downloading multiprocess-0.70.14-py39-none-any.whl (132 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 132.9/132.9 kB 10.4 MB/s eta 0:00:00
Collecting responses<0.19
  Downloading responses-0.18.0-py3-none-any.whl (38 kB)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.9/dist-packages (from
Collecting dill<0.3.7,>=0.3.0
  Downloading dill-0.3.6-py3-none-any.whl (110 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 110.5/110.5 kB 7.0 MB/s eta 0:00:00
Collecting xxhash
  Downloading xxhash-3.2.0-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (212 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 212.2/212.2 kB 8.7 MB/s eta 0:00:00
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.9/dist-packages (f
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.9/dist-packages (from
Collecting multidict<7.0,>=4.5
  Downloading multidict-6.0.4-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (114
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 114.2/114.2 kB 2.6 MB/s eta 0:00:00
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in /usr/local/lib/python3.9/dist
Collecting yarl<2.0,>=1.0
  Downloading yarl-1.8.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (264 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 264.6/264.6 kB 9.6 MB/s eta 0:00:00
Collecting frozenlist>=1.1.1
  Downloading frozenlist-1.3.3-cp39-cp39-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 158.8/158.8 kB 8.6 MB/s eta 0:00:00
Collecting aiosignal>=1.1.2
  Downloading aiosignal-1.3.1-py3-none-any.whl (7.6 kB)
Collecting async-timeout<5.0,>=4.0.0a3
  Downloading async_timeout-4.0.2-py3-none-any.whl (5.8 kB)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.9/dist-p
Requirement already satisfied: filelock in /usr/local/lib/python3.9/dist-packages (from hugg
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packag
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages (from pyth
Installing collected packages: xxhash, multidict, frozenlist, dill, async-timeout, yarl, res
Successfully installed aiohttp-3.8.4 aiosignal-1.3.1 async-timeout-4.0.2 datasets-2.11.0 dil
```

```
import pandas as pd
from datasets import load_dataset
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import confusion_matrix
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn import svm
from sklearn.ensemble import VotingClassifier
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
dataset = load_dataset("tweets_hate_speech_detection")
```

```
Downloading builder script: 100%   3.27k/3.27k [00:00<00:00, 176kB/s]
Downloading metadata: 100%   1.84k/1.84k [00:00<00:00, 115kB/s]
Downloading readme: 100%   5.46k/5.46k [00:00<00:00, 262kB/s]
Downloading and preparing dataset tweets_hate_speech_detection/default to /root/.cache/huggi
Downloading data files: 100%   2/2 [00:01<00:00, 1.60it/s]
Downloading data:   3.10M/? [00:00<00:00, 23.0MB/s]
Downloading data:   1.64M/? [00:00<00:00, 27.6MB/s]

Dataset tweets_hate_speech_detection downloaded and prepared to /root/.cache/huggingface/dat
100%   2/2 [00:00<00:00, 65.71it/s]
```

```
dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['label', 'tweet'],
        num_rows: 31962
    })
    test: Dataset({
        features: ['label', 'tweet'],
        num_rows: 17197
    })
})
```

We used [Hate-Speech Twitter Dataset] from hugging face to create hate speech detection model which is a subset of sentiment analysis. In this study, we shall apply various techniques in NLP such as linear and non-linear algorithms to analyse the hate speech text.

Text pre-processing

```
X = dataset['train']['tweet']
y = dataset['train']['label']
```

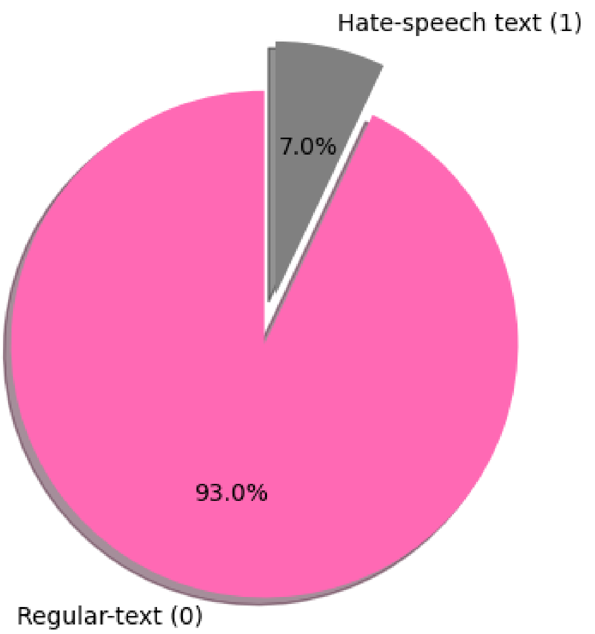
```
d = {'text_tweet': X, 'label_tweet': y}
df = pd.DataFrame(data=d)
print("Size of dataset : ",df.shape[0])
print("Number of columns : ",df.shape[1])
print(df.head())
```

```
Size of dataset :  31962
Number of columns :  2
                                          text_tweet  label_tweet
0  @user when a father is dysfunctional and is so...            0
1  @user @user thanks for #lyft credit i can't us...            0
2                                bihday your majesty            0
3  #model   i love u take with u all the time in ...            0
4             factsguide: society now    #motivation            0
```

```
df['label_tweet'].value_counts() #0 = non-hate speech , 1 = hate speech
```

```
0    29720
1     2242
Name: label_tweet, dtype: int64
```

```
mylabels = ["Regular-text (0)", "Hate-speech text (1)",]
mycolors = ["hotpink","gray"]
plt.pie(df['label_tweet'].value_counts(),labels = mylabels, colors = mycolors, startangle=90, aut
#plt.title("Proportion of the hate speech in dataset")
plt.show()
```




```
extra_word =['A', 'And','He','She','What','The','I','one','man','It','said','two','would','like',
punctuations='''!()-,[]{};:"@/,.\%^&’”?@ð#$*&“±!!! ð-_—'''

df['text_tweet'] = df['text_tweet'].apply(lambda x: " ".join(x.lower() for x in x.split())) #lowe
stop_word = stopwords.words('english')
df['text_tweet']  = df['text_tweet'].apply(lambda x: " ".join([word for word in x.split() if word
) #stop-word
df['text_tweet']  = df['text_tweet'].apply(lambda x: " ".join([word for word in x.split() if word
) #extra-word
df['text_tweet']  = df['text_tweet'].apply(lambda x: " ".join([word for word in x.split() if word
) #punctuations
```

```
df.head(3)
```

	text_tweet	label_tweet
0	father dysfunctional selfish drags kids dysfun...	0
1	thanks #lyft credit can't use cause offer whee...	0
2	bihday majesty	0

Set random_state for reproducibility and stratify to ensure split the same amount of label classes when split into train and test set

```
X_train, X_test, y_train, y_test = train_test_split(df['text_tweet'], df['label_tweet'], test_siz
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
((25569,), (25569,), (6393,), (6393,))
```

```
# for training dataset
values, counts = np.unique(y_train, return_counts=True)
print("number of non-hate speech [0] :", counts[0], "number of hate speech [1] : ", counts[1])
```

```
number of non-hate speech [0] : 23775 number of hate speech [1] :  1794
```

```
# for testing dataset
values, counts = np.unique(y_test, return_counts=True)
print("number of non-hate speech [0] :", counts[0], "number of hate speech [1] : ", counts[1])
```

```
number of non-hate speech [0] : 5945 number of hate speech [1] :  448
```

0 = non-hate speech , 1 = hate speech

Baseline Method

A basic approach for identifying hate speech is using a keyword-based approach.

```
hate_words = ['bad','terrible','hate','awful','disappointing','disgusting','dislike','loathe','ab

def detect_hatespeech(sentence):
  sentiment = 0
  words = sentence.lower().split()
  for word in words:
    if word in hate_words:
      sentiment = 1
      break
  return sentiment

result_train = []
result_test = []

for sentence in X_train:
  result_train.append(detect_hatespeech(sentence))
for sentence in X_test:
  result_test.append(detect_hatespeech(sentence))

print("Training set Accuracy : ", accuracy_score(y_train, result_train))
print("Testing set Accuracy : ", accuracy_score(y_test, result_test))
```

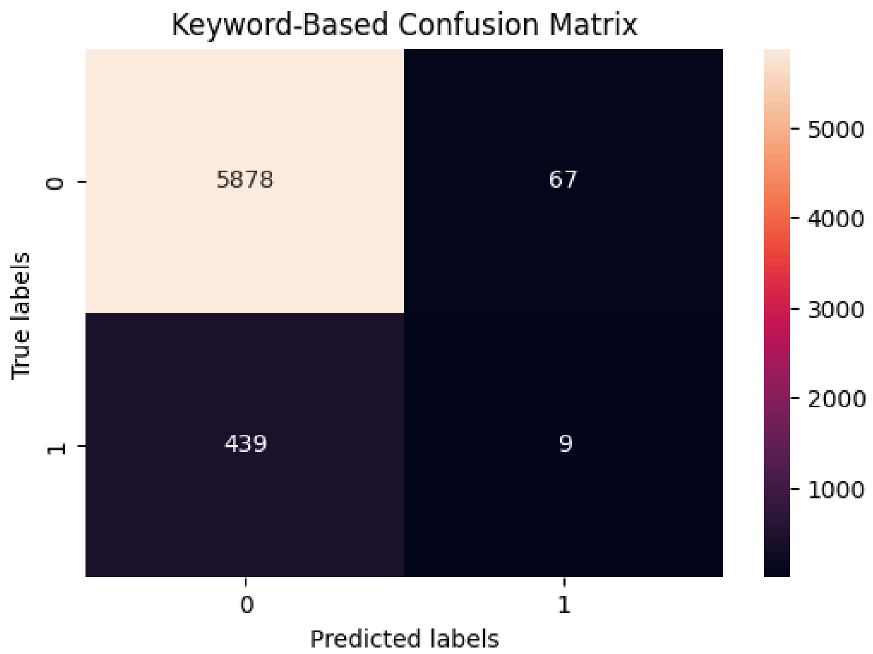
```
Training set Accuracy :  0.9213891822128358
Testing set Accuracy :  0.9208509307054591
```

```
cm = confusion_matrix(y_train, result_train)
print(cm)
```

```
[[23522   253]
 [ 1757    37]]
```

```
#sns.set_theme(style='darkgrid')
#plt.style.use("dark_background")

cm = confusion_matrix(y_test, result_test)
plt.figure(figsize=(6,4))
ax= plt.subplot()
sns.heatmap(cm, annot=True, fmt='g', ax=ax);
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Keyword-Based Confusion Matrix');
ax.xaxis.set_ticklabels(['0', '1']); ax.yaxis.set_ticklabels(['0', '1']);
```



Note : our algorithm based on keyword failed to detect hate speech. Therefore, we shall apply machine learning algorithm to create the machine learning model that can detect hate speech in the context.

Machine Learning Algorithms

```
# vectorizer = CountVectorizer(ngram_range = (1, 1), stop_words='english')
  train_features = vectorizer.fit_transform(X_train)
  test_features = vectorizer.transform(X_test)
```

Data needs to be converted into its vector representation for the purpose of building machine learning model. The vectors representation can be generated using different methods such as bag-of-words, TF-IDF, and word embeddings.

Uni-Gram model

```
vectorizer = CountVectorizer(ngram_range = (1, 1), stop_words='english')
train_features = vectorizer.fit_transform(X_train)
test_features = vectorizer.transform(X_test)

np.random.seed(42)
LR_model = LogisticRegression()
LR_model.fit(train_features, y_train);
print("Logistic Model Acc : ", LR_model.score(train_features, y_train))

SVM_model=svm.SVC()
SVM_model.fit(train_features,y_train);
print("SVM Model Acc : ",SVM_model.score(train_features, y_train))

estimators=[('Logistic Regression', LR_model), ('Support vector machine', SVM_model)]
ensemble = VotingClassifier(estimators, voting='hard')
ensemble.fit(train_features, y_train)
print("Ensemble Model ", ensemble.score(train_features, y_train))
```

```
Logistic Model Acc :  0.9851773632132661
SVM Model Acc :  0.9849035941960969
Ensemble Model  0.9829872110759122
```

Bi-Gram model

```
vectorizer = CountVectorizer(ngram_range = (2, 2), stop_words='english')
train_features = vectorizer.fit_transform(X_train)
test_features = vectorizer.transform(X_test)

np.random.seed(42)
LR_model = LogisticRegression()
LR_model.fit(train_features, y_train);
print("Logistic Model Acc : ", LR_model.score(train_features, y_train))

SVM_model=svm.SVC()
SVM_model.fit(train_features,y_train);
print("SVM Model Acc : ",SVM_model.score(train_features, y_train))

estimators=[('Logistic Regression', LR_model), ('Support vector machine', SVM_model)]
ensemble = VotingClassifier(estimators, voting='hard')
ensemble.fit(train_features, y_train)
print("Ensemble Model ", ensemble.score(train_features, y_train))
```

```
Logistic Model Acc :  0.9790762251163518
SVM Model Acc :  0.9734835151941804
Ensemble Model  0.9730533067386288
```

Tri-Gram model

```
vectorizer = CountVectorizer(ngram_range = (3, 3), stop_words='english')
train_features = vectorizer.fit_transform(X_train)
test_features = vectorizer.transform(X_test)

np.random.seed(42)
LR_model = LogisticRegression()
LR_model.fit(train_features, y_train);
print("Logistic Model Acc : ", LR_model.score(train_features, y_train))

SVM_model=svm.SVC()
SVM_model.fit(train_features,y_train);
print("SVM Model Acc : ",SVM_model.score(train_features, y_train))
```

```
estimators=[('Logistic Regression', LR_model), ('Support vector machine', SVM_model)]
ensemble = VotingClassifier(estimators, voting='hard')
ensemble.fit(train_features, y_train)
print("Ensemble Model ", ensemble.score(train_features, y_train))
```

```
Logistic Model Acc :  0.9700027376901716
SVM Model Acc :  0.9714106926356134
Ensemble Model  0.9699636278305761
```

bag-of-words + Logistic Regression

```
for i in range(3) :
  print(i+1, "gram start training : ")
  vectorizer = CountVectorizer(ngram_range = (i+1, i+1), stop_words='english')
  train_features = vectorizer.fit_transform(X_train)
  test_features = vectorizer.transform(X_test)

  param_grid = {'C': [0.1, 1, 10],
                'penalty': ['l2']}
  LR_model = LogisticRegression(random_state=42, max_iter=1000)
  LR_grid_search = GridSearchCV(LR_model, param_grid, cv=5, scoring='accuracy',verbose=3)
  LR_grid_search.fit(train_features, y_train);
```

```
1 gram start training :
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV 1/5] END .................C=0.1, penalty=l2;, score=0.945 total time=   0.8s
[CV 2/5] END .................C=0.1, penalty=l2;, score=0.943 total time=   0.3s
[CV 3/5] END .................C=0.1, penalty=l2;, score=0.946 total time=   0.4s
[CV 4/5] END .................C=0.1, penalty=l2;, score=0.947 total time=   0.3s
[CV 5/5] END .................C=0.1, penalty=l2;, score=0.947 total time=   0.7s
[CV 1/5] END ...................C=1, penalty=l2;, score=0.959 total time=   0.7s
[CV 2/5] END ...................C=1, penalty=l2;, score=0.956 total time=   1.0s
[CV 3/5] END ...................C=1, penalty=l2;, score=0.957 total time=   0.8s
[CV 4/5] END ...................C=1, penalty=l2;, score=0.960 total time=   0.9s
[CV 5/5] END ...................C=1, penalty=l2;, score=0.960 total time=   1.2s
[CV 1/5] END ..................C=10, penalty=l2;, score=0.964 total time=   1.3s
[CV 2/5] END ..................C=10, penalty=l2;, score=0.961 total time=   1.2s
[CV 3/5] END ..................C=10, penalty=l2;, score=0.959 total time=   1.1s
[CV 4/5] END ..................C=10, penalty=l2;, score=0.963 total time=   2.2s
[CV 5/5] END ..................C=10, penalty=l2;, score=0.961 total time=   2.2s
2 gram start training :
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV 1/5] END .................C=0.1, penalty=l2;, score=0.935 total time=   0.8s
[CV 2/5] END .................C=0.1, penalty=l2;, score=0.936 total time=   1.4s
[CV 3/5] END .................C=0.1, penalty=l2;, score=0.934 total time=   1.0s
[CV 4/5] END .................C=0.1, penalty=l2;, score=0.935 total time=   0.9s
[CV 5/5] END .................C=0.1, penalty=l2;, score=0.937 total time=   1.1s
[CV 1/5] END ...................C=1, penalty=l2;, score=0.944 total time=   2.2s
[CV 2/5] END ...................C=1, penalty=l2;, score=0.945 total time=   3.4s
[CV 3/5] END ...................C=1, penalty=l2;, score=0.945 total time=   1.8s
[CV 4/5] END ...................C=1, penalty=l2;, score=0.945 total time=   1.7s
[CV 5/5] END ...................C=1, penalty=l2;, score=0.947 total time=   1.9s
[CV 1/5] END ..................C=10, penalty=l2;, score=0.949 total time=   3.2s
[CV 2/5] END ..................C=10, penalty=l2;, score=0.951 total time=   2.0s
[CV 3/5] END ..................C=10, penalty=l2;, score=0.949 total time=   4.5s
[CV 4/5] END ..................C=10, penalty=l2;, score=0.950 total time=   2.8s
[CV 5/5] END ..................C=10, penalty=l2;, score=0.953 total time=   2.7s
3 gram start training :
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV 1/5] END .................C=0.1, penalty=l2;, score=0.934 total time=   1.2s
[CV 2/5] END .................C=0.1, penalty=l2;, score=0.935 total time=   2.0s
[CV 3/5] END .................C=0.1, penalty=l2;, score=0.933 total time=   1.9s
[CV 4/5] END .................C=0.1, penalty=l2;, score=0.935 total time=   1.3s
[CV 5/5] END .................C=0.1, penalty=l2;, score=0.936 total time=   1.1s
[CV 1/5] END ...................C=1, penalty=l2;, score=0.941 total time=   1.2s
[CV 2/5] END ...................C=1, penalty=l2;, score=0.942 total time=   1.4s
```

```
[CV 3/5] END ...................C=1, penalty=l2;, score=0.941 total time=   1.9s
[CV 4/5] END ...................C=1, penalty=l2;, score=0.941 total time=   1.2s
[CV 5/5] END ...................C=1, penalty=l2;, score=0.943 total time=   1.4s
[CV 1/5] END ..................C=10, penalty=l2;, score=0.947 total time=   4.1s
[CV 2/5] END ..................C=10, penalty=l2;, score=0.947 total time=   2.5s
[CV 3/5] END ..................C=10, penalty=l2;, score=0.947 total time=   2.6s
[CV 4/5] END ..................C=10, penalty=l2;, score=0.948 total time=   4.8s
[CV 5/5] END ..................C=10, penalty=l2;, score=0.950 total time=   6.5s
```

```
vectorizer = CountVectorizer(ngram_range = (1, 1), stop_words='english')
train_features = vectorizer.fit_transform(X_train)
test_features = vectorizer.transform(X_test)

param_grid = {'C': [0.1, 1, 10],
              'penalty': ['l2']}
LR_model = LogisticRegression(random_state=42, max_iter=1000)
LR_grid_search = GridSearchCV(LR_model, param_grid, cv=5, scoring='accuracy',verbose=0)
LR_grid_search.fit(train_features, y_train);
print("Best hyperparameters: ", LR_grid_search.best_params_)
print("Best accuracy score: ", LR_grid_search.best_score_)
```

```
Best hyperparameters:  {'C': 10, 'penalty': 'l2'}
Best accuracy score:  0.9614376491373182
```

Error Analysis for Logistic Regression There are 239 tweet that Logistic model fail to capture and below is some of example of our result.

```
result = LR_grid_search.predict(test_features)
error_mask = result != y_test
df = pd.DataFrame({'sentence': X_test[error_mask], 'actual label': y_test[error_mask], 'preict la
df.head(5)
```

	sentence	actual label	preict label
20064	immature trying make fool #xenophobe #immature...	1	0
24767	porn vids web free sex	1	0
7638	president #woodrowwilson held private screenin...	1	0
6224	sums voted #brexit; #littleenglander syndrome ...	1	0
10187	.@user gf used uber without forcing language p...	1	0

```
result = LR_grid_search.predict(test_features)
cm = confusion_matrix(y_test, result)
plt.figure(figsize=(6,4))
ax= plt.subplot()
sns.heatmap(cm, annot=True, fmt='g', ax=ax);
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Logistic Regression Confusion Matrix');
ax.xaxis.set_ticklabels(['0', '1']); ax.yaxis.set_ticklabels(['0', '1']);
```

Note : Using CountVectorizer + Logistic Regression model improve the accuracy to detect hate speech in the sentence than searching based on only keyword

TF-IDF + Logistic Regression

```
# Equivalent to CountVectorizer followed by TfidfTransformer.
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(ngram_range = (1, 1), stop_words='english')
train_features = vectorizer.fit_transform(X_train)
test_features = vectorizer.transform(X_test)


for i in range(3) :
  print(i+1, "gram start training : ")
  vectorizer = TfidfVectorizer(ngram_range = (i+1, i+1), stop_words='english')
  train_features = vectorizer.fit_transform(X_train)
  test_features = vectorizer.transform(X_test)

  param_grid = {'C': [0.1, 1, 10],
                'penalty': ['l2']}
  LR_model = LogisticRegression(random_state=42, max_iter=1000)
  LR_grid_search = GridSearchCV(LR_model, param_grid, cv=5, scoring='accuracy',verbose=3)
  LR_grid_search.fit(train_features, y_train);
```

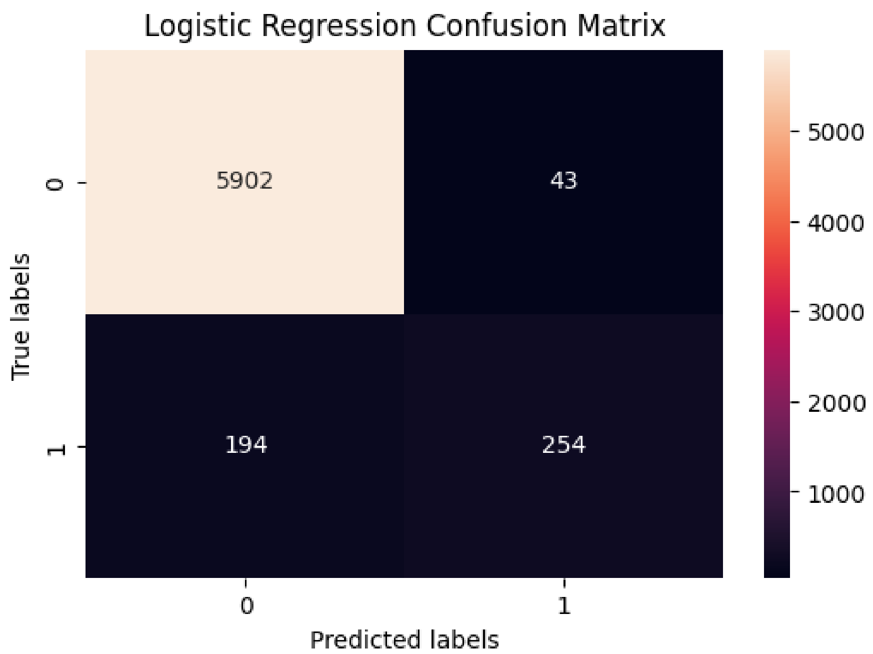
```
1 gram start training :
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV 1/5] END .................C=0.1, penalty=l2;, score=0.932 total time=   0.3s
[CV 2/5] END .................C=0.1, penalty=l2;, score=0.932 total time=   0.3s
[CV 3/5] END .................C=0.1, penalty=l2;, score=0.931 total time=   1.1s
[CV 4/5] END .................C=0.1, penalty=l2;, score=0.930 total time=   0.4s
[CV 5/5] END .................C=0.1, penalty=l2;, score=0.930 total time=   0.5s
[CV 1/5] END ...................C=1, penalty=l2;, score=0.944 total time=   0.9s
[CV 2/5] END ...................C=1, penalty=l2;, score=0.944 total time=   0.8s
[CV 3/5] END ...................C=1, penalty=l2;, score=0.944 total time=   0.4s
[CV 4/5] END ...................C=1, penalty=l2;, score=0.945 total time=   0.6s
[CV 5/5] END ...................C=1, penalty=l2;, score=0.946 total time=   1.4s
[CV 1/5] END ..................C=10, penalty=l2;, score=0.961 total time=   2.1s
[CV 2/5] END ..................C=10, penalty=l2;, score=0.958 total time=   1.3s
[CV 3/5] END ..................C=10, penalty=l2;, score=0.957 total time=   1.2s
[CV 4/5] END ..................C=10, penalty=l2;, score=0.960 total time=   1.7s
[CV 5/5] END ..................C=10, penalty=l2;, score=0.960 total time=   1.2s
2 gram start training :
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV 1/5] END .................C=0.1, penalty=l2;, score=0.930 total time=   1.0s
[CV 2/5] END .................C=0.1, penalty=l2;, score=0.930 total time=   1.1s
[CV 3/5] END .................C=0.1, penalty=l2;, score=0.930 total time=   1.8s
[CV 4/5] END .................C=0.1, penalty=l2;, score=0.930 total time=   1.3s
[CV 5/5] END .................C=0.1, penalty=l2;, score=0.930 total time=   1.3s
[CV 1/5] END ...................C=1, penalty=l2;, score=0.935 total time=   1.8s
[CV 2/5] END ...................C=1, penalty=l2;, score=0.937 total time=   1.4s
[CV 3/5] END ...................C=1, penalty=l2;, score=0.935 total time=   1.7s
[CV 4/5] END ...................C=1, penalty=l2;, score=0.936 total time=   1.4s
```

```
[CV 5/5] END ...................C=1, penalty=l2;, score=0.938 total time=   1.3s
[CV 1/5] END ..................C=10, penalty=l2;, score=0.946 total time=   4.3s
[CV 2/5] END ..................C=10, penalty=l2;, score=0.947 total time=   2.1s
[CV 3/5] END ..................C=10, penalty=l2;, score=0.946 total time=   2.9s
[CV 4/5] END ..................C=10, penalty=l2;, score=0.947 total time=   3.2s
[CV 5/5] END ..................C=10, penalty=l2;, score=0.949 total time=   3.9s
3 gram start training :
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV 1/5] END .................C=0.1, penalty=l2;, score=0.930 total time=   1.0s
[CV 2/5] END .................C=0.1, penalty=l2;, score=0.930 total time=   1.6s
[CV 3/5] END .................C=0.1, penalty=l2;, score=0.930 total time=   1.2s
[CV 4/5] END .................C=0.1, penalty=l2;, score=0.930 total time=   1.6s
[CV 5/5] END .................C=0.1, penalty=l2;, score=0.930 total time=   0.8s
[CV 1/5] END ...................C=1, penalty=l2;, score=0.934 total time=   2.2s
[CV 2/5] END ...................C=1, penalty=l2;, score=0.937 total time=   1.8s
[CV 3/5] END ...................C=1, penalty=l2;, score=0.935 total time=   1.4s
[CV 4/5] END ...................C=1, penalty=l2;, score=0.936 total time=   1.2s
[CV 5/5] END ...................C=1, penalty=l2;, score=0.937 total time=   1.6s
[CV 1/5] END ..................C=10, penalty=l2;, score=0.944 total time=   3.2s
[CV 2/5] END ..................C=10, penalty=l2;, score=0.945 total time=   3.2s
[CV 3/5] END ..................C=10, penalty=l2;, score=0.946 total time=   3.5s
[CV 4/5] END ..................C=10, penalty=l2;, score=0.945 total time=   2.5s
[CV 5/5] END ..................C=10, penalty=l2;, score=0.947 total time=   3.1s
```

```
param_grid = {'C': [0.1, 1, 10],
              'penalty': ['l2']}
LR_model = LogisticRegression(random_state=42, max_iter=1000)
LR_grid_search = GridSearchCV(LR_model, param_grid, cv=5, scoring='accuracy',verbose=0)
LR_grid_search.fit(train_features, y_train);
print("Best hyperparameters: ", LR_grid_search.best_params_)
print("Best accuracy score: ", LR_grid_search.best_score_)
```

```
Best hyperparameters:  {'C': 10, 'penalty': 'l2'}
Best accuracy score:  0.959012894428696
```

```
result = LR_grid_search.predict(test_features)
cm = confusion_matrix(y_test, result)
plt.figure(figsize=(6,4))
ax= plt.subplot()
sns.heatmap(cm, annot=True, fmt='g', ax=ax);
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Logistic Regression Confusion Matrix');
ax.xaxis.set_ticklabels(['0', '1']); ax.yaxis.set_ticklabels(['0', '1']);
```



Comparision Model

Huggingface pre-train sentiment model

Due to hate speech analysis is a sub class of sentiment analysis, hugging face provide an effective pretrained model for use to classify the label for the text using Transformer model.

To begin with, we import pretrained model from hugging face. As it already train model we only need to create pipeline for sentiment analysis and feed our data that we split into this pipeline.

```
!pip install transformers datasets
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/s
Requirement already satisfied: transformers in /usr/local/lib/python3.9/dist-packages (4.28.
Requirement already satisfied: datasets in /usr/local/lib/python3.9/dist-packages (2.11.0)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: filelock in /usr/local/lib/python3.9/dist-packages (from tran
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib/python3.9
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.9/dist-packages (from t
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.9/dist-packages (from tr
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.9/dist-packages (from t
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (from tran
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (fr
Requirement already satisfied: huggingface-hub<1.0,>=0.11.0 in /usr/local/lib/python3.9/dist
Requirement already satisfied: pandas in /usr/local/lib/python3.9/dist-packages (from datase
Requirement already satisfied: multiprocess in /usr/local/lib/python3.9/dist-packages (from
Requirement already satisfied: responses<0.19 in /usr/local/lib/python3.9/dist-packages (fro
Requirement already satisfied: dill<0.3.7,>=0.3.0 in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: fsspec[http]>=2021.11.1 in /usr/local/lib/python3.9/dist-pack
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.9/dist-packages (fro
Requirement already satisfied: aiohttp in /usr/local/lib/python3.9/dist-packages (from datas
Requirement already satisfied: xxhash in /usr/local/lib/python3.9/dist-packages (from datase
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.9/dist-packages (fro
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.9/dist-packages (from
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/python3.9/dist-
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in /usr/local/lib/python3.9/dist
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.9/dist-packages (f
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.9/dist-p
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packag
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packa
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages (from pyth
```

```
from transformers import pipeline
```

```
classifier = pipeline("sentiment-analysis")
```

```
No model was supplied, defaulted to distilbert-base-uncased-finetuned-sst-2-english and revi
```

```
# Example of how to use pipeline for predicting sentiment from text
for i in range(10):
  x = classifier(X_test.iloc[i])
  print(x)
```

```
[{'label': 'NEGATIVE', 'score': 0.9771596789360046}]
[{'label': 'NEGATIVE', 'score': 0.9291220307350159}]
[{'label': 'NEGATIVE', 'score': 0.9944455623626709}]
[{'label': 'POSITIVE', 'score': 0.7282702922821045}]
[{'label': 'POSITIVE', 'score': 0.9997286200523376}]
[{'label': 'POSITIVE', 'score': 0.9972419738769531}]
[{'label': 'NEGATIVE', 'score': 0.9398183226585388}]
[{'label': 'POSITIVE', 'score': 0.7012082934379578}]
[{'label': 'NEGATIVE', 'score': 0.9801750779151917}]
[{'label': 'POSITIVE', 'score': 0.9998643398284912}]
```

```
tran_pred=[]
for i in range(len(X_train)):
  prediction = classifier(X_train.iloc[i])
  if prediction[0]['label'] =='NEGATIVE':
    tran_pred.append(1)
  elif prediction[0]['label'] =='POSITIVE':
    tran_pred.append(0)
```

```
print("Transformers Model Acc: ", accuracy_score(y_train, tran_pred))
```

The pre-trained model from hugging face failed to capture hate speech from the text due to accuracy score being low (50%) equal to random guess prediction. It is because most of the time model capture negative sentences from the texts that just complain about something but that does not mean to be a hate speech or intend to attack or make other feel bad. Thatmean we have to train our specific model that can capture hate speech text in sentence. Hence, we implement BERT along with deep model to create hate speech detection.

BERT Transformer model

Another approach that we implement from huggingface is BERT transformer pre-trained model. Using this algorithm and add the classify layer at the end and observe the performance compare to our baseline model

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

```
if torch.cuda.is_available():
    device = torch.device("cuda")
    print('There are %d GPU(s) available.' % torch.cuda.device_count())
    print('GPU name:', torch.cuda.get_device_name(0))

else:
    print('No GPU available, using the CPU instead.')
    device = torch.device("cpu")
```

```
There are 1 GPU(s) available.
We will use the GPU: Tesla T4
```

```
!pip install transformers
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/s
Requirement already satisfied: transformers in /usr/local/lib/python3.9/dist-packages (4.28.
Requirement already satisfied: filelock in /usr/local/lib/python3.9/dist-packages (from trar
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib/python3.9
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: huggingface-hub<1.0,>=0.11.0 in /usr/local/lib/python3.9/dist
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.9/dist-packages (from t
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (fr
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.9/dist-packages (from t
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (from trar
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.9/dist-packages (from tr
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.9/dist-p
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packag
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-pa
```

```
sentences=df['text_tweet'].values
labels = df['label_tweet'].values
```

```
from transformers import BertTokenizer
print('Loading BERT tokenizer...')
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)
```

```
Loading BERT tokenizer...
```

```
print('Original: ', sentences[0])
print('Tokenized: ', tokenizer.tokenize(sentences[0]))
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.tokenize(sentences[0])))
```

```
Original:  father dysfunctional selfish drags kids dysfunction. #run
Tokenized:  ['father', 'dysfunction', '##al', 'selfish', 'drag', '##s', 'kids', 'dysfunctior
Token IDs:  [2269, 28466, 2389, 14337, 8011, 2015, 4268, 28466, 1012, 1001, 2448]
```

```
max_len = 0
for sent in sentences:

    # Tokenize the text and add `[CLS]` and `[SEP]` tokens.
    input_ids = tokenizer.encode(sent, add_special_tokens=True)

    # Update the maximum sentence length.
    max_len = max(max_len, len(input_ids))

print('Max sentence length: ', max_len)
```

```
Max sentence length:  137
```

```
#''' Credit : BERT Fine-Tuning Tutorial with PyTorch By Chris McCormick and Nick Ryan '''
input_ids = []
attention_masks = []

for sent in sentences:
    encoded_dict = tokenizer.encode_plus(
                        sent,                      # Sentence to encode.
                        add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                        max_length = 140,           # Pad & truncate all sentences.
                        pad_to_max_length = True,
                        return_attention_mask = True,   # Construct attn. masks.
                        return_tensors = 'pt',     # Return pytorch tensors.
                   )
    # Add the encoded sentence to the list.
```

```
        input_ids.append(encoded_dict['input_ids'])

        # And its attention mask (simply differentiates padding from non-padding).
        attention_masks.append(encoded_dict['attention_mask'])

# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)

# Print sentence 0, now as a list of IDs.
print('Original: ', sentences[0])
print('Token IDs:', input_ids[0])
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, pleas
Original:  father dysfunctional selfish drags kids dysfunction. #run
Token IDs: tensor([  101,  2269, 28466,  2389, 14337,  8011,  2015,  4268, 28466,  1012,
         1001,  2448,   102,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0])
```

```
from torch.utils.data import TensorDataset, random_split

# Combine the training inputs into a TensorDataset.
dataset = TensorDataset(input_ids, attention_masks, labels)

# Calculate the number of samples to include in each set.
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size

# Divide the dataset by randomly selecting samples.
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

print('{:>5,} training samples'.format(train_size))
print('{:>5,} validation samples'.format(val_size))
```

```
25,569 training samples
6,393 validation samples
```

```
from transformers import DistilBertModel, DistilBertTokenizer, AdamW
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
model = DistilBertModel.from_pretrained('distilbert-base-uncased')
```

```
Downloading (…)solve/main/vocab.txt: 100%          232k/232k [00:00<00:00, 7.16MB/s
Downloading (…)okenizer_config.json: 100%          28.0/28.0 [00:00<00:00, 1.30kB/s]
Downloading (…)lve/main/config.json: 100%          483/483 [00:00<00:00, 30.9kB/s]
Downloading pytorch_model.bin: 100%                268M/268M [00:01<00:00, 277MB/s]
Some weights of the model checkpoint at distilbert-base-uncased were not used when initializ
- This IS expected if you are initializing DistilBertModel from the checkpoint of a model tr
- This IS NOT expected if you are initializing DistilBertModel from the checkpoint of a mode
```

```
# Define the model architecture
class SentimentClassifier(nn.Module):
    def __init__(self, model):
        super(SentimentClassifier, self).__init__()
        self.model = model
        self.linear = nn.Linear(768, 1) #adding a linear layer at the output hidden state of the

    def forward(self, input_ids, attention_mask):
        outputs = self.model(input_ids=input_ids, attention_mask=attention_mask)
        last_hidden_state = outputs.last_hidden_state[:, 0, :]
        logits = self.linear(last_hidden_state)
        return logits.squeeze(-1)

# Instantiate the model
model = SentimentClassifier(model)



optimizer = AdamW(model.parameters(), lr=1e-5)
loss_fn = nn.BCEWithLogitsLoss()

# Define the training loop
def train(model, train_loader, optimizer, loss_fn, device):
    model.train()
    for batch in train_loader:
        input_ids = batch[0].to(device)
        attention_mask = batch[1].to(device)
        labels = batch[2].to(device)

        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask)
        loss = loss_fn(outputs, labels.float())
        #print(loss)
        loss.backward()
        optimizer.step()

# Define the evaluation loop
def evaluate(model, test_loader, device):
    model.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for batch in test_loader:
            input_ids = batch[0].to(device)
            attention_mask = batch[1].to(device)
            labels = batch[2].to(device)

            outputs = model(input_ids, attention_mask)
            predictions = torch.round(torch.sigmoid(outputs))

            total += labels.size(0)
            correct += (predictions == labels).sum().item()

    return correct / total
```

```
/usr/local/lib/python3.9/dist-packages/transformers/optimization.py:391: FutureWarning: This
  warnings.warn(
```

```
# Train the model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = torch.utils.data.DataLoader(val_dataset, batch_size=16, shuffle=False)
```

```
for epoch in range(3):
    train(model, train_loader, optimizer, loss_fn, device)
    accuracy = evaluate(model, test_loader, device)
    print(f'Epoch {epoch+1} - Test Accuracy: {accuracy:.3f}')
```

```
Epoch 1 - Test Accuracy: 0.967
Epoch 2 - Test Accuracy: 0.970
Epoch 3 - Test Accuracy: 0.967
```

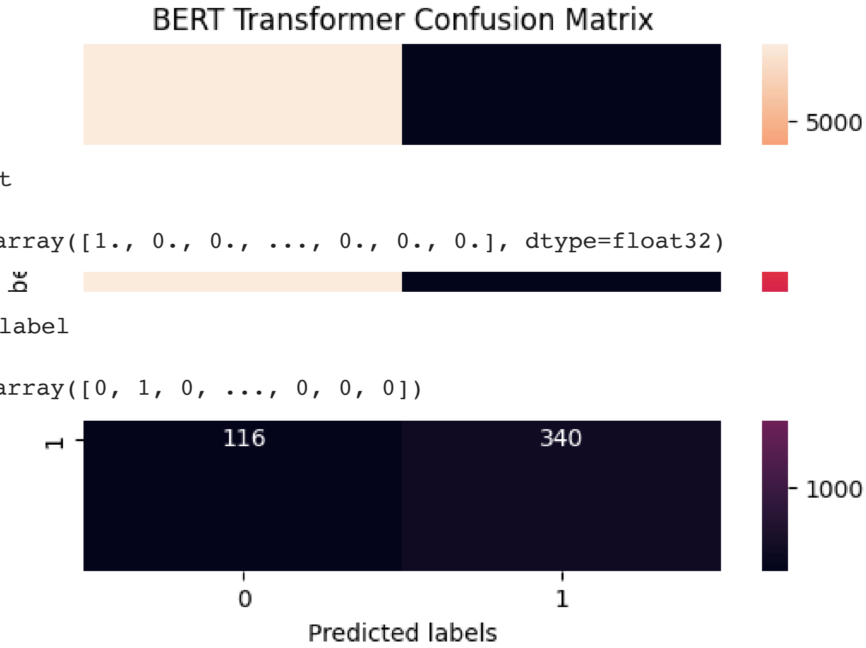
```
# Evaluation testing set performance
model.eval()
correct, total = 0, 0
result = []
true_label=[]
with torch.no_grad():
  for batch in test_loader:
    input_ids = batch[0].to(device)
    attention_mask = batch[1].to(device)
    labels = batch[2].to(device)

    outputs = model(input_ids, attention_mask)
    predictions = torch.round(torch.sigmoid(outputs))
    #print('--------')
    #print(predictions)
    total += labels.size(0)
    #print(total)
    correct += (predictions == labels).sum().item()
    result.append(predictions.detach().cpu().numpy())
    true_label.append(labels.detach().cpu().numpy())

print('Test Accuracy :', correct/total )
print('Error prediction :', total-correct)
result=np.concatenate(result)
true_label=np.concatenate(true_label)
```

```
Test Accuracy : 0.9673079931174723
Error prediction : 209
```

```
cm = confusion_matrix(true_label, result)
plt.figure(figsize=(6,4))
ax= plt.subplot()
sns.heatmap(cm, annot=True, fmt='g', ax=ax);
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('BERT Transformer Confusion Matrix');
ax.xaxis.set_ticklabels(['0', '1']); ax.yaxis.set_ticklabels(['0', '1']);
```




```
result
```

```
array([1., 0., 0., ..., 0., 0., 0.], dtype=float32)
```

```
true_label
```

```
array([0, 1, 0, ..., 0, 0, 0])
```