Toxic Comment Classification Challenge

Thiago Andrade

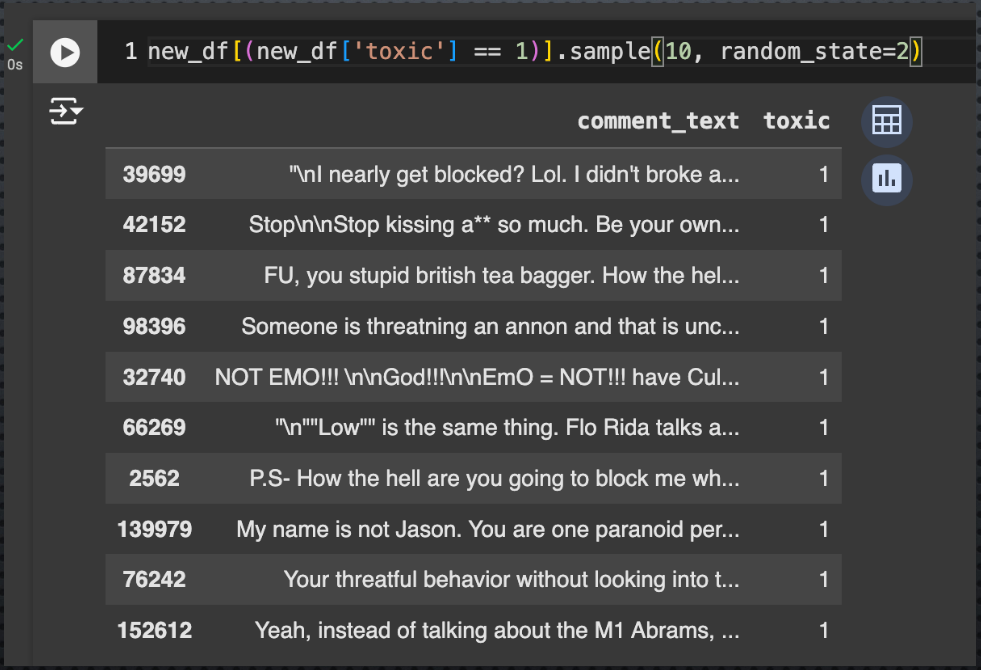
Objetivos

- Identificar e classificar comentários tóxicos
- Utilizar diversas técnicas de processamento de linguagem
- Comparar os resultados das técnicas utilizadas

Dataset

- Comentários em artigos da Wikipedia que foram classificados por humanos como tendo conteúdo tóxico. Source: ref
- Os tipos de classificação tóxica são:
  - *toxic*
  - *severe_toxic*
  - *obscene*
  - *threat*
  - *insult*
  - *identity_hate*
- O dataset foi processado para somente conter a informação de toxicidade, não importando o tipo

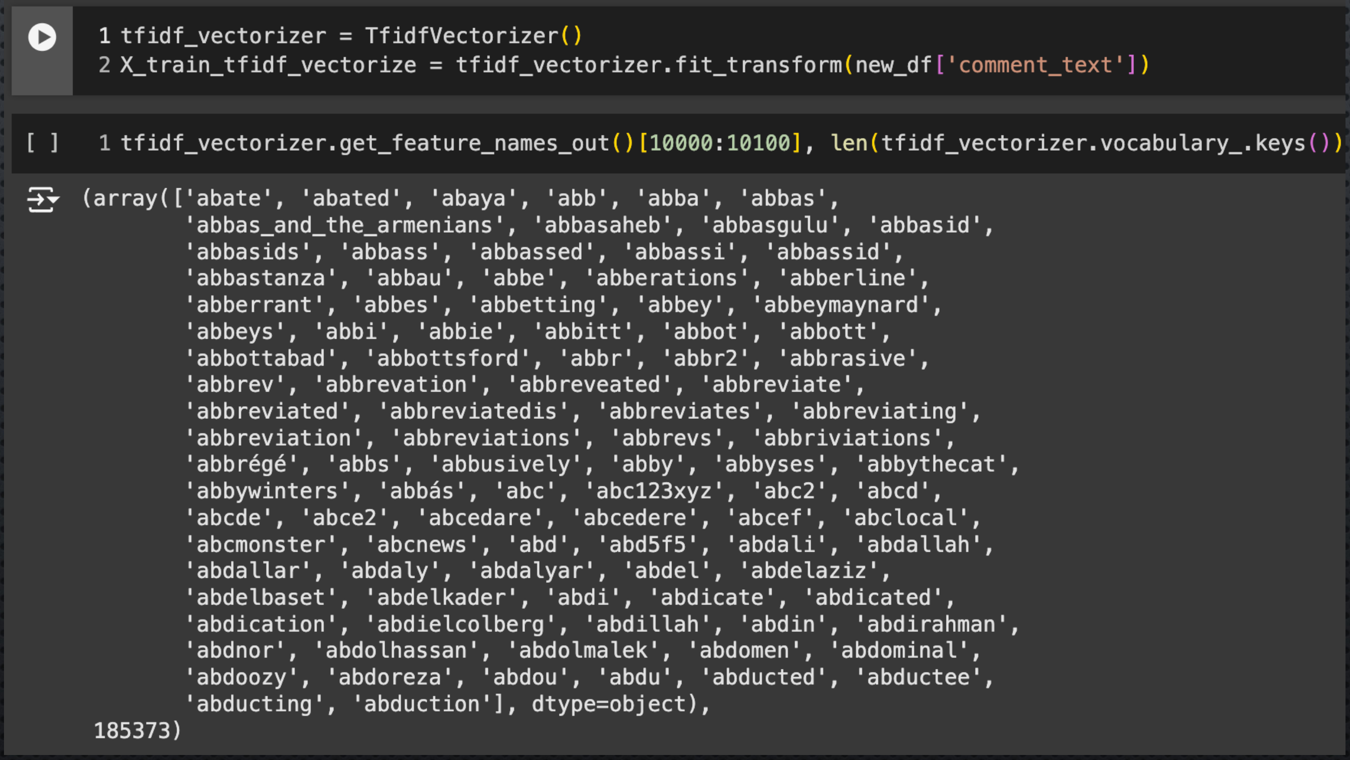
DATASET



```python
new_df[(new_df['toxic'] == 1)].sample(10, random_state=2)
```

	comment_text	toxic
39699	"\nI nearly get blocked? Lol. I didn't broke a...	1
42152	Stop\n\nStop kissing a** so much. Be your own...	1
87834	FU, you stupid british tea bagger. How the hel...	1
98396	Someone is threatning an annon and that is unc...	1
32740	NOT EMO!!! \n\nGod!!!\n\nEmO = NOT!!! have Cul...	1
66269	"\n""Low"" is the same thing. Flo Rida talks a...	1
2562	P.S- How the hell are you going to block me wh...	1
139979	My name is not Jason. You are one paranoid per...	1
76242	Your threatful behavior without looking into t...	1
152612	Yeah, instead of talking about the M1 Abrams, ...	1

BOW & SVM

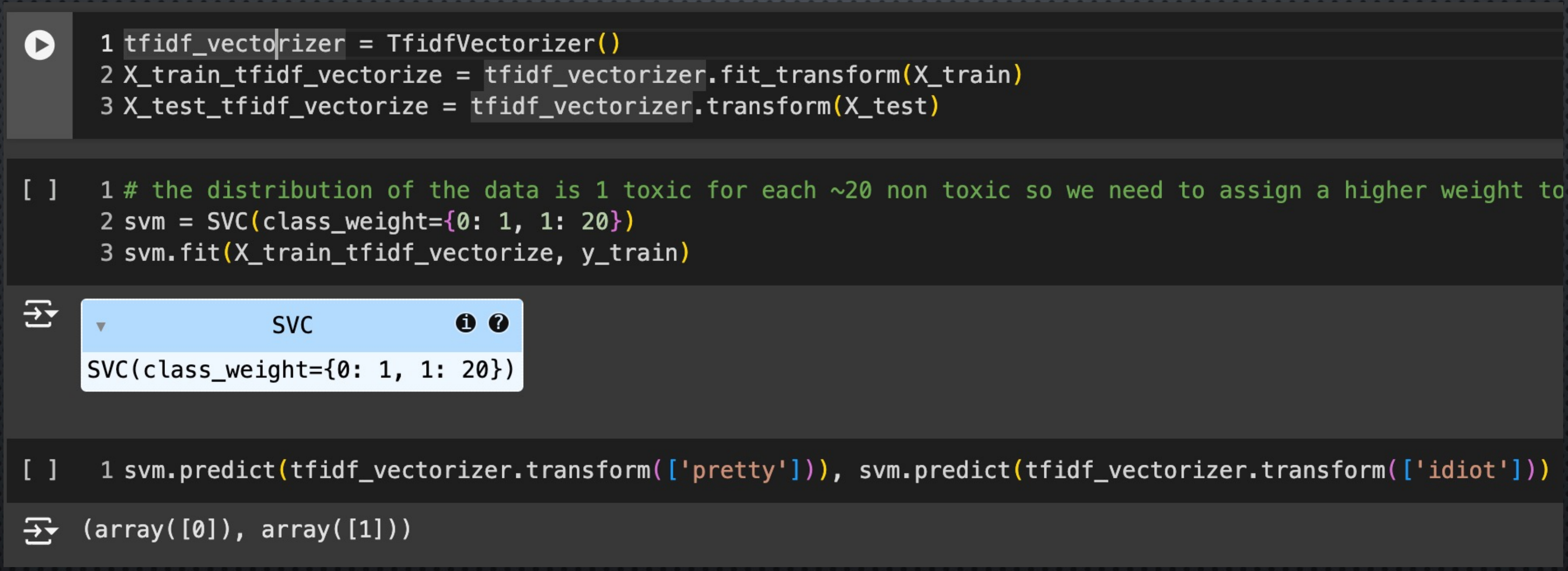


```python
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf_vectorize = tfidf_vectorizer.fit_transform(new_df['comment_text'])
```

```python
tfidf_vectorizer.get_feature_names_out()[10000:10100], len(tfidf_vectorizer.vocabulary_.keys())
```

```
(array(['abate', 'abated', 'abaya', 'abb', 'abba', 'abbas',
        'abbas_and_the_armenians', 'abbasaheb', 'abbasgulu', 'abbasid',
        'abbasids', 'abbass', 'abbassed', 'abbassi', 'abbassid',
        'abbastanza', 'abbau', 'abbe', 'abberations', 'abberline',
        'abberrant', 'abbes', 'abbetting', 'abbey', 'abbeymaynard',
        'abbeys', 'abbi', 'abbie', 'abbitt', 'abbot', 'abbott',
        'abbottabad', 'abbottsford', 'abbr', 'abbr2', 'abbrasive',
        'abbrev', 'abbrevation', 'abbreveated', 'abbreviate',
        'abbreviated', 'abbreviatedis', 'abbreviates', 'abbreviating',
        'abbreviation', 'abbreviations', 'abbrevs', 'abbriviations',
        'abbrégé', 'abbs', 'abbusively', 'abby', 'abbyses', 'abbythecat',
        'abbywinters', 'abbás', 'abc', 'abc123xyz', 'abc2', 'abcd',
        'abcde', 'abce2', 'abcedare', 'abcedere', 'abcef', 'abclocal',
        'abcmonster', 'abcnews', 'abd', 'abd5f5', 'abdali', 'abdallah',
        'abdallar', 'abdaly', 'abdalyar', 'abdel', 'abdelaziz',
        'abdelbaset', 'abdelkader', 'abdi', 'abdicate', 'abdicated',
        'abdication', 'abdielcolberg', 'abdillah', 'abdin', 'abdirahman',
        'abdnor', 'abdolhassan', 'abdolmalek', 'abdomen', 'abdominal',
        'abdoozy', 'abdoreza', 'abdou', 'abdu', 'abducted', 'abductee',
        'abducting', 'abduction'], dtype=object),
 185373)
```

BOW & SVM



```
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf_vectorize = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf_vectorize = tfidf_vectorizer.transform(X_test)
```

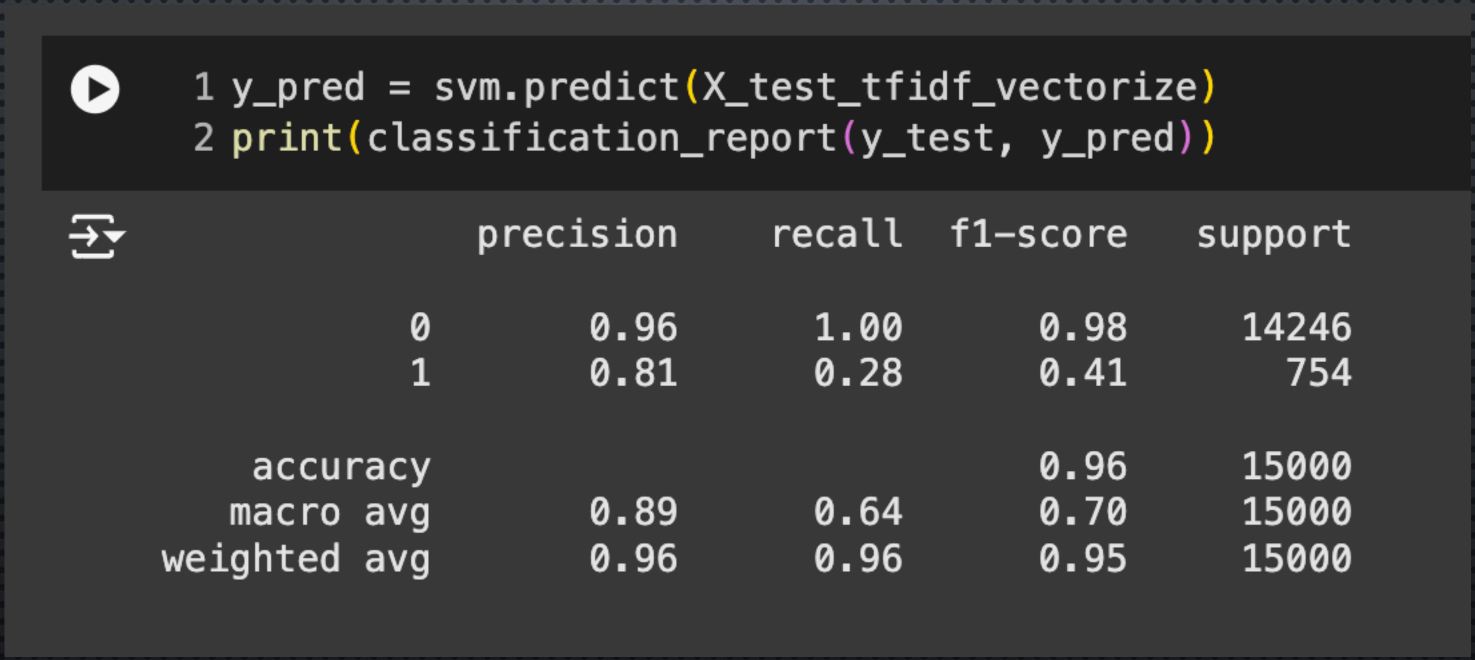
```
# the distribution of the data is 1 toxic for each ~20 non toxic so we need to assign a higher weight to
svm = SVC(class_weight={0: 1, 1: 20})
svm.fit(X_train_tfidf_vectorize, y_train)
```

```
SVC(class_weight={0: 1, 1: 20})
```

```
svm.predict(tfidf_vectorizer.transform(['pretty'])), svm.predict(tfidf_vectorizer.transform(['idiot']))
```

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

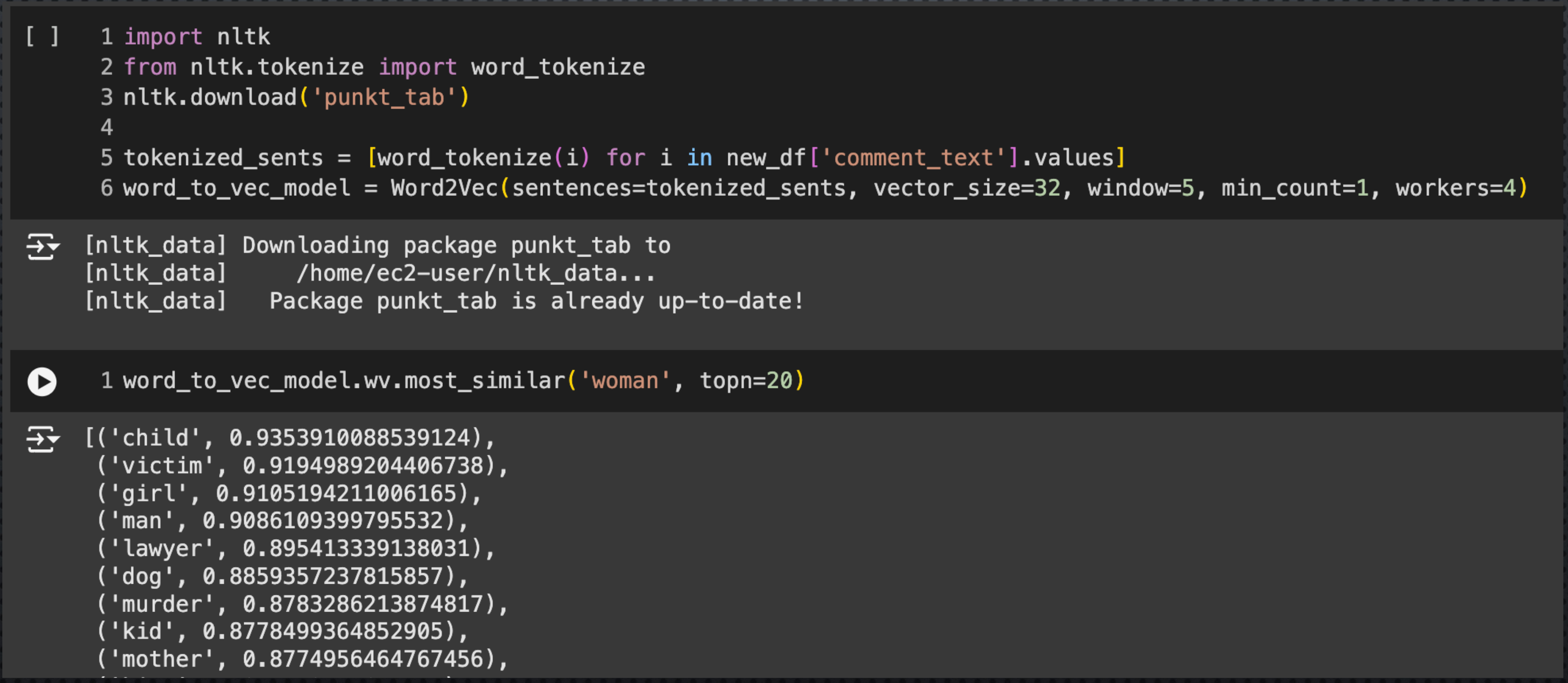
BOW & SVM



```
y_pred = svm.predict(X_test_tfidf_vectorize)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98     14246
           1       0.81      0.28      0.41       754

    accuracy                           0.96     15000
   macro avg       0.89      0.64      0.70     15000
weighted avg       0.96      0.96      0.95     15000
```

EMBEDDINGS & SVM



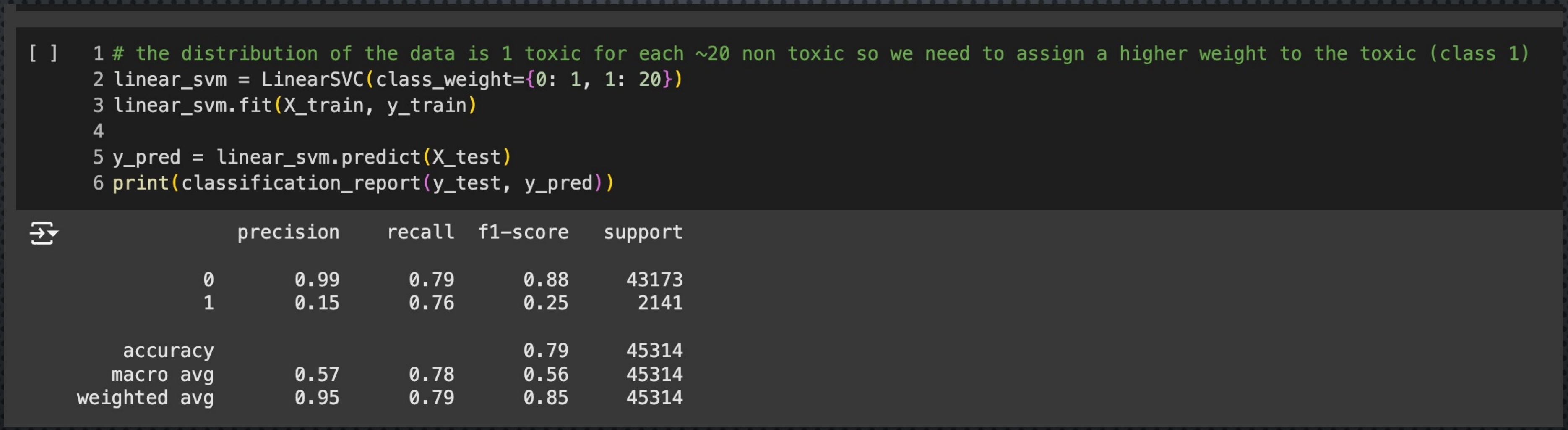
```
[ ]
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt_tab')

tokenized_sents = [word_tokenize(i) for i in new_df['comment_text'].values]
word_to_vec_model = Word2Vec(sentences=tokenized_sents, vector_size=32, window=5, min_count=1, workers=4)
```

```
[nltk_data] Downloading package punkt_tab to
[nltk_data]     /home/ec2-user/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

```
word_to_vec_model.wv.most_similar('woman', topn=20)
```

```
[('child', 0.9353910088539124),
 ('victim', 0.9194989204406738),
 ('girl', 0.9105194211006165),
 ('man', 0.9086109399795532),
 ('lawyer', 0.895413339138031),
 ('dog', 0.8859357237815857),
 ('murder', 0.8783286213874817),
 ('kid', 0.8778499364852905),
 ('mother', 0.8774956464767456),
```

EMBEDDINGS & SVM

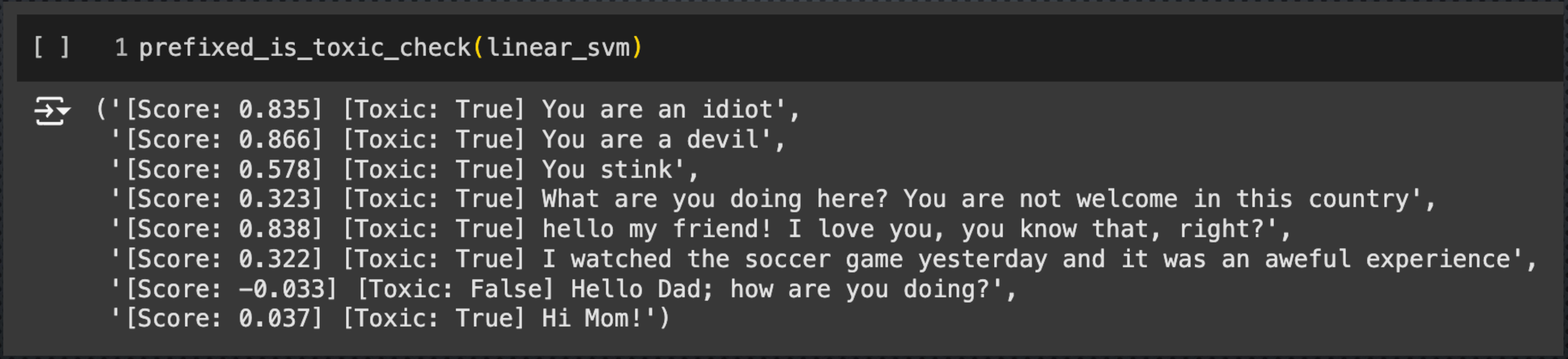


```python
# the distribution of the data is 1 toxic for each ~20 non toxic so we need to assign a higher weight to the toxic (class 1)
linear_svm = LinearSVC(class_weight={0: 1, 1: 20})
linear_svm.fit(X_train, y_train)

y_pred = linear_svm.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.79      0.88     43173
           1       0.15      0.76      0.25      2141

    accuracy                           0.79     45314
   macro avg       0.57      0.78      0.56     45314
weighted avg       0.95      0.79      0.85     45314
```

EMBEDDINGS & SVM

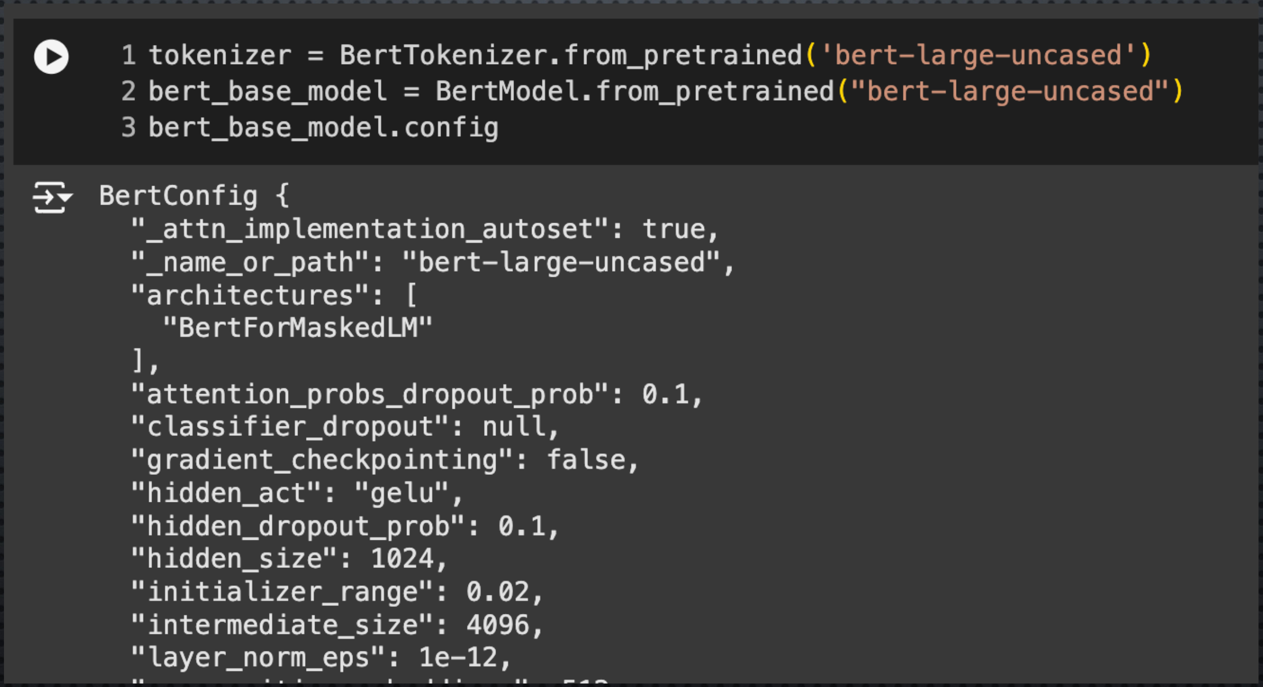
EMBEDDINGS & SVM



```
prefixed_is_toxic_check(linear_svm)
```

```
('[Score: 0.835] [Toxic: True] You are an idiot',
 '[Score: 0.866] [Toxic: True] You are a devil',
 '[Score: 0.578] [Toxic: True] You stink',
 '[Score: 0.323] [Toxic: True] What are you doing here? You are not welcome in this country',
 '[Score: 0.838] [Toxic: True] hello my friend! I love you, you know that, right?',
 '[Score: 0.322] [Toxic: True] I watched the soccer game yesterday and it was an aweful experience',
 '[Score: -0.033] [Toxic: False] Hello Dad; how are you doing?',
 '[Score: 0.037] [Toxic: True] Hi Mom!')
```

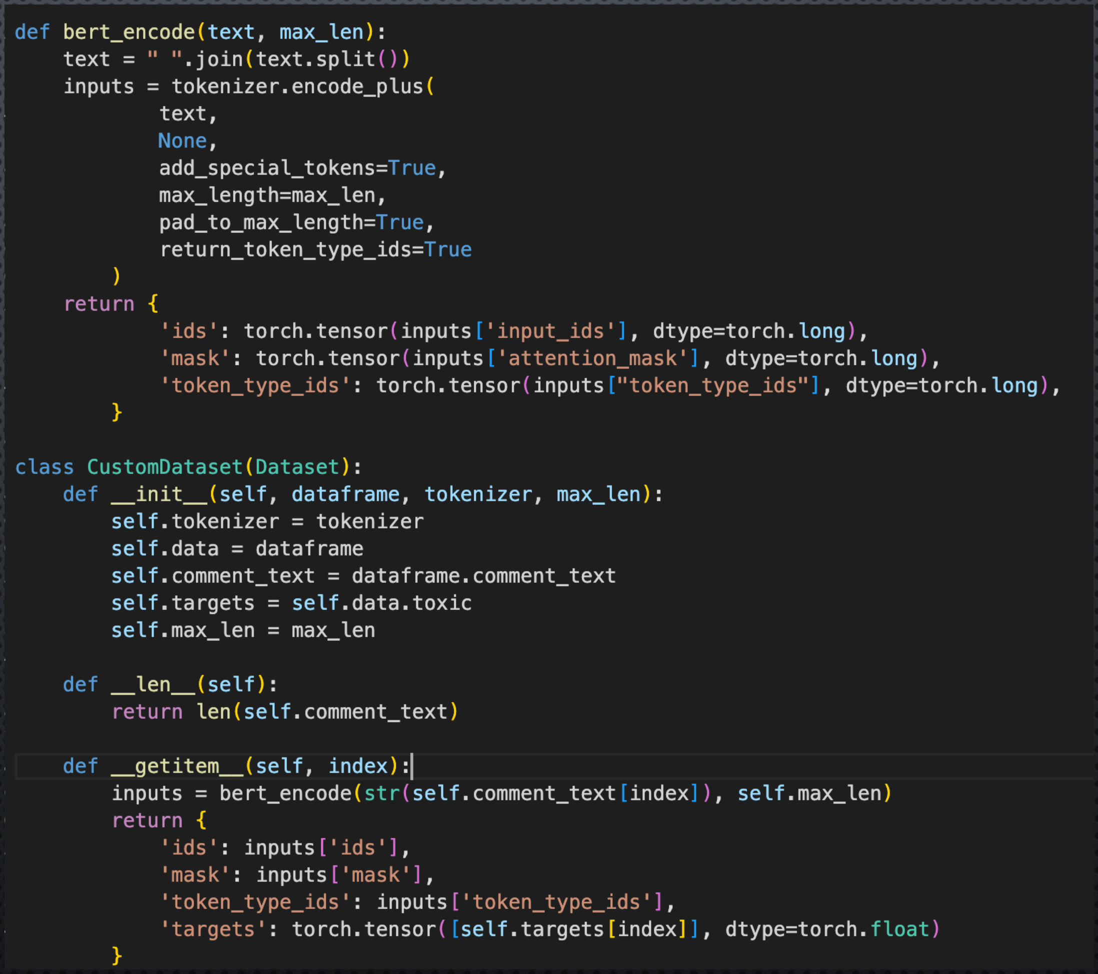
BERT



```python
tokenizer = BertTokenizer.from_pretrained('bert-large-uncased')
bert_base_model = BertModel.from_pretrained("bert-large-uncased")
bert_base_model.config
```

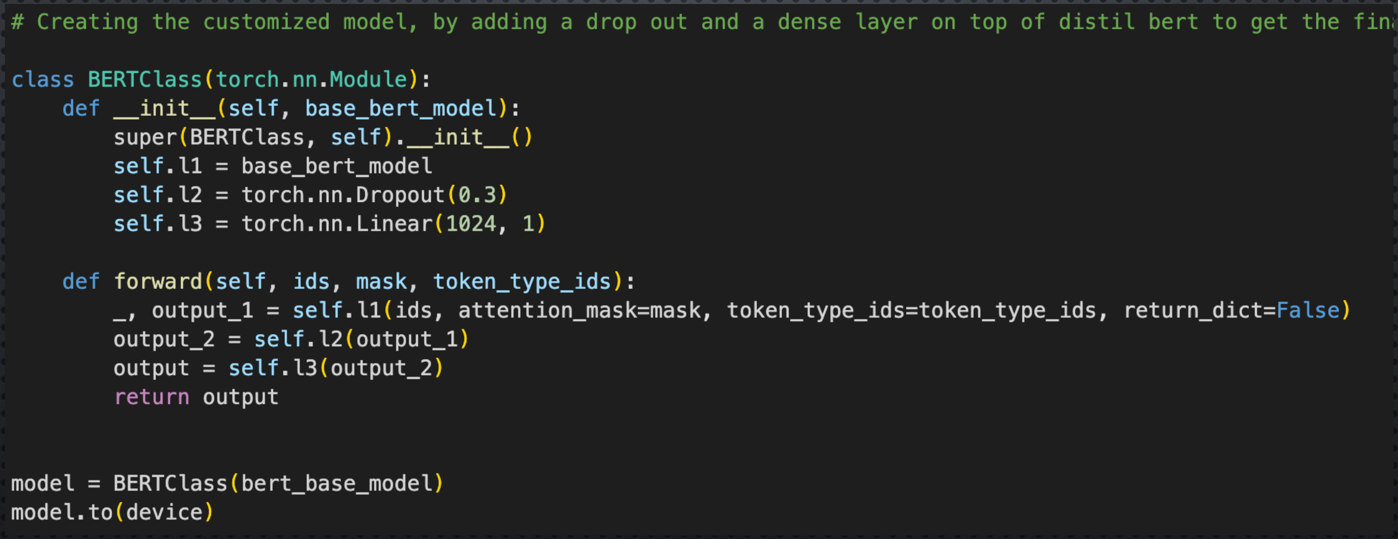
```
BertConfig {
  "_attn_implementation_autoset": true,
  "_name_or_path": "bert-large-uncased",
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 1024,
  "initializer_range": 0.02,
  "intermediate_size": 4096,
  "layer_norm_eps": 1e-12,
```

BERT



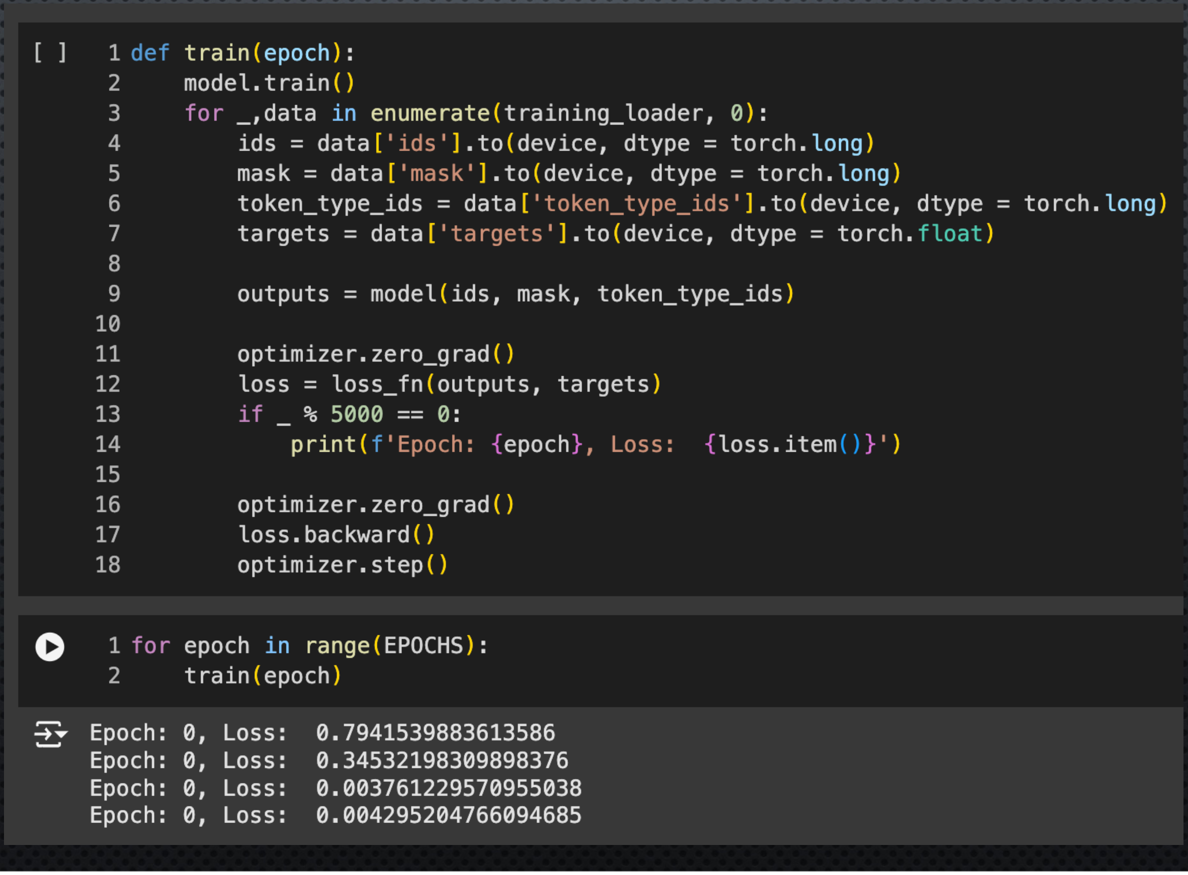
```python
def bert_encode(text, max_len):
    text = " ".join(text.split())
    inputs = tokenizer.encode_plus(
            text,
            None,
            add_special_tokens=True,
            max_length=max_len,
            pad_to_max_length=True,
            return_token_type_ids=True
        )
    return {
            'ids': torch.tensor(inputs['input_ids'], dtype=torch.long),
            'mask': torch.tensor(inputs['attention_mask'], dtype=torch.long),
            'token_type_ids': torch.tensor(inputs["token_type_ids"], dtype=torch.long),
        }

class CustomDataset(Dataset):
    def __init__(self, dataframe, tokenizer, max_len):
        self.tokenizer = tokenizer
        self.data = dataframe
        self.comment_text = dataframe.comment_text
        self.targets = self.data.toxic
        self.max_len = max_len

    def __len__(self):
        return len(self.comment_text)

    def __getitem__(self, index):
        inputs = bert_encode(str(self.comment_text[index]), self.max_len)
        return {
            'ids': inputs['ids'],
            'mask': inputs['mask'],
            'token_type_ids': inputs['token_type_ids'],
            'targets': torch.tensor([self.targets[index]], dtype=torch.float)
        }
```

BERT



```
# Creating the customized model, by adding a drop out and a dense layer on top of distil bert to get the fin

class BERTClass(torch.nn.Module):
    def __init__(self, base_bert_model):
        super(BERTClass, self).__init__()
        self.l1 = base_bert_model
        self.l2 = torch.nn.Dropout(0.3)
        self.l3 = torch.nn.Linear(1024, 1)

    def forward(self, ids, mask, token_type_ids):
        _, output_1 = self.l1(ids, attention_mask=mask, token_type_ids=token_type_ids, return_dict=False)
        output_2 = self.l2(output_1)
        output = self.l3(output_2)
        return output


model = BERTClass(bert_base_model)
model.to(device)
```

BERT

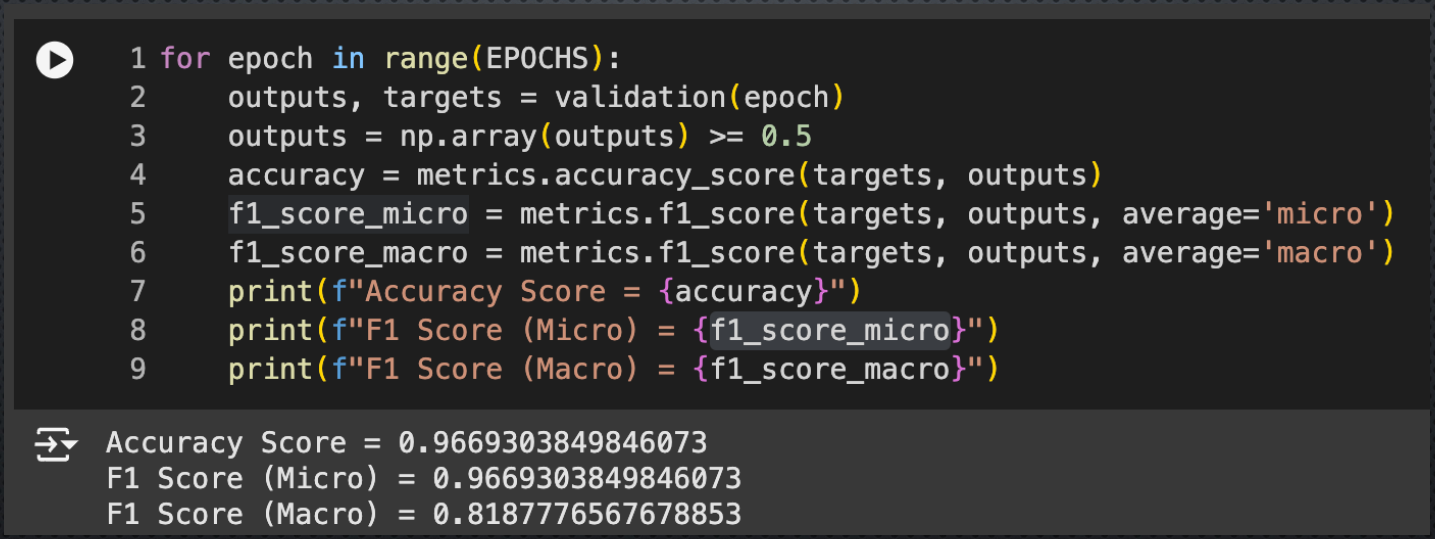


```python
def train(epoch):
    model.train()
    for _,data in enumerate(training_loader, 0):
        ids = data['ids'].to(device, dtype = torch.long)
        mask = data['mask'].to(device, dtype = torch.long)
        token_type_ids = data['token_type_ids'].to(device, dtype = torch.long)
        targets = data['targets'].to(device, dtype = torch.float)

        outputs = model(ids, mask, token_type_ids)

        optimizer.zero_grad()
        loss = loss_fn(outputs, targets)
        if _ % 5000 == 0:
            print(f'Epoch: {epoch}, Loss:  {loss.item()}')

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

```python
for epoch in range(EPOCHS):
    train(epoch)
```

```
Epoch: 0, Loss:  0.7941539883613586
Epoch: 0, Loss:  0.34532198309898376
Epoch: 0, Loss:  0.003761229570955038
Epoch: 0, Loss:  0.004295204766094685
```

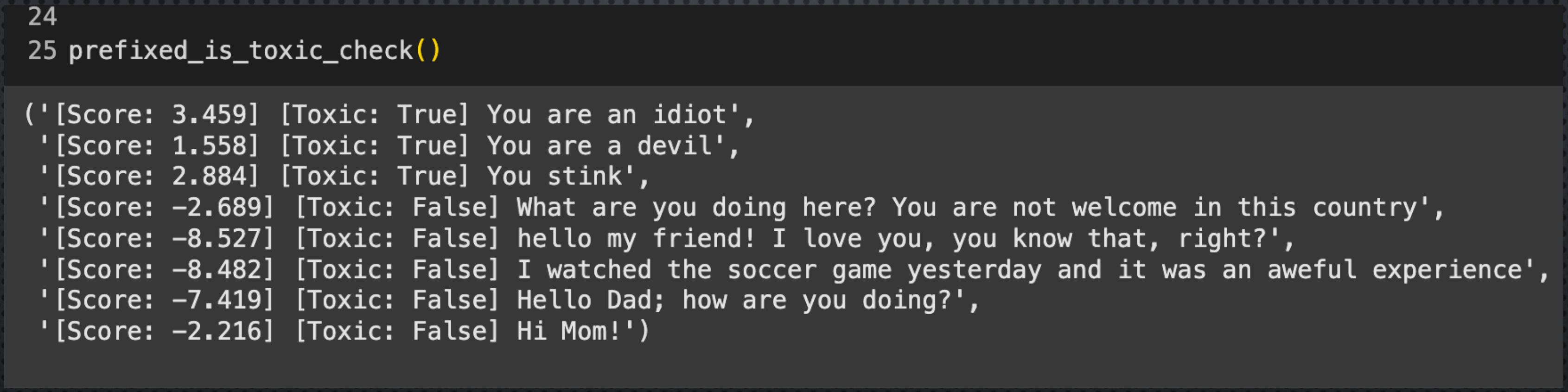
BERT



```python
for epoch in range(EPOCHS):
    outputs, targets = validation(epoch)
    outputs = np.array(outputs) >= 0.5
    accuracy = metrics.accuracy_score(targets, outputs)
    f1_score_micro = metrics.f1_score(targets, outputs, average='micro')
    f1_score_macro = metrics.f1_score(targets, outputs, average='macro')
    print(f"Accuracy Score = {accuracy}")
    print(f"F1 Score (Micro) = {f1_score_micro}")
    print(f"F1 Score (Macro) = {f1_score_macro}")
```

```
Accuracy Score = 0.9669303849846073
F1 Score (Micro) = 0.9669303849846073
F1 Score (Macro) = 0.8187776567678853
```

BERT



```
prefixed_is_toxic_check()
```

```
('[Score: 3.459] [Toxic: True] You are an idiot',
 '[Score: 1.558] [Toxic: True] You are a devil',
 '[Score: 2.884] [Toxic: True] You stink',
 '[Score: -2.689] [Toxic: False] What are you doing here? You are not welcome in this country',
 '[Score: -8.527] [Toxic: False] hello my friend! I love you, you know that, right?',
 '[Score: -8.482] [Toxic: False] I watched the soccer game yesterday and it was an aweful experience',
 '[Score: -7.419] [Toxic: False] Hello Dad; how are you doing?',
 '[Score: -2.216] [Toxic: False] Hi Mom!')
```

CONCLUSÕES

- BERT É O CLARO VENCEDOR DENTRE AS TRÊS ABORDAGENS
- F1 SCORE COM 1 ÚNICA ÉPOCA DE *FINE TUNING* DE 0.96