Notebook Contents

0. Installations

Return to contents

In [1]:
```
#%pip install sklearn
#%pip install eli5
#%pip install transformers==4.4.1
#%pip install torch torchvision torchaudio
#%pip install ipywidgets
#%pip install sentence-transformers
#%pip install MulticoreTSNE
```

In [2]:
```
# import the necessary libraries
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2' #Trying to reduce tensorflow warnings
import re
import math
```

```
import string
import time
import json
import pickle
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from tqdm import tqdm

# sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from MulticoreTSNE import MulticoreTSNE

# useful structures and functions for experiments
from time import sleep

# specific machine learning functionality
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.python.keras import backend as K

# Transformers
import transformers
from transformers import BertTokenizer, TFBertForSequenceClassification, BertConfig, TF
from transformers import GPT2Tokenizer, TFGPT2LMHeadModel
from transformers import pipeline

# Data structures
from dataclasses import dataclass
from typing import List, Optional, Tuple

# Use pre-trained models and pre-generated data
train_model = False
train_new_data = False
```

1. Data Reading and Preprocessing

Return to contents

In [3]:

```
### Data Reading
df = pd.read_json("data/reviews.json", lines=True)
```

In [4]:

```
### Creating Labels
df["label"] = df["overall"] < 4
df["review"] = df["reviewText"]
```

```
### Removing Other Columns
df = df[["review", "label"]]
```

In [5]:
```
### Checking for maximum length (useful for later)
lengths = []
for s in df["review"]:
    lengths.append(
      len(s.split())
    )
lengths = np.asarray(lengths)
np.quantile(lengths, [0.1, 0.5, 0.75, 0.9, 0.95, 0.98])
```

Out[5]:
```
array([ 24.,  66., 118., 192., 254., 347.])
```

In [6]:
```
### Train/Test split
df_train, df_test = train_test_split(df,
                                     test_size=0.2,
                                     shuffle=True,
                                     random_state=98,
                                     stratify=df["label"])
```

In [7]:
```
### Check the rate of positive labels in each set
temp = df["label"].mean()
print(f"{temp:0.4f}")

temp = df_train["label"].mean()
print(f"{temp:0.4f}")

temp = df_test["label"].mean()
print(f"{temp:0.4f}")
```

```
0.2063
0.2063
0.2063
```

In [8]:
```
### Check the set sizes
print(df.shape)
print(df_train.shape)
print(df_test.shape)
```

```
(151254, 2)
(121003, 2)
(30251, 2)
```

2. Creating Artificially Imbalanced Data

Return to contents

In [9]:
```
def create_imbalanced_set(df, desired_ratio = 0.05):
    pos_neg_ratio = desired_ratio / (1 - desired_ratio)
    df_negative = df[df["label"] == False]
```

```
    df_positive = df[df["label"] == True]
    n_positive = int(pos_neg_ratio * df_negative.shape[0])
    df_positive = df_positive[:n_positive]
    return pd.concat([df_positive, df_negative])
```

In [10]:
```
df_train_imbalanced = create_imbalanced_set(df_train)
df_test_imbalanced = create_imbalanced_set(df_test)
```

In [11]:
```
temp = df_train_imbalanced["label"].mean()
print(f"{temp:0.4f}")

temp = df_test_imbalanced["label"].mean()
print(f"{temp:0.4f}")
```

```
0.0500
0.0500
```

3. Modeling: Full Training Data

Return to contents

3.1: Using Word-Based Model

Return to contents

In [12]:
```
### Vectorization using TF-IDF
tfidf_vectorizer = TfidfVectorizer(min_df=2)
X_train = tfidf_vectorizer.fit_transform(df_train["review"])
Y_train = df_train["label"]
```

In [13]:
```
### Logistic Regression
log_reg = LogisticRegression(random_state=98, max_iter=5000).fit(X_train, Y_train)
```

In [14]:
```
### Tokenizing test sets
X_test = tfidf_vectorizer.transform(df_test["review"])
Y_test = df_test["label"]

X_test_imbalanced = tfidf_vectorizer.transform(df_test_imbalanced["review"])
Y_test_imbalanced = df_test_imbalanced["label"]
```

In [15]:
```
### Evaluation
print(f"The accuracy on the train data: {log_reg.score(X_train, Y_train):.4f}")
print(f"The accuracy on the test data: {log_reg.score(X_test, Y_test):.4f}")
print(f"The accuracy on the imbalanced test data: {log_reg.score(X_test_imbalanced, Y_t

print()
```

```
print(f"The F1 on the train data: {f1_score(Y_train, log_reg.predict(X_train)):.4f}")
print(f"The F1 on the test data: {f1_score(Y_test, log_reg.predict(X_test)):.4f}")
print(f"The F1 on the imbalanced test data: {f1_score(Y_test_imbalanced, log_reg.predic
```

```
The accuracy on the train data: 0.8929
The accuracy on the test data: 0.8799
The accuracy on the imbalanced test data: 0.9409

The F1 on the train data: 0.7016
The F1 on the test data: 0.6620
The F1 on the imbalanced test data: 0.4891
```

3.2: Using Context-Based Model

Return to contents

In [16]:

```
### Tokenization parameters
bert_tokenizer = BertTokenizer.from_pretrained('prajjwal1/bert-tiny',do_lower_case=True
batch_size = 8
AUTOTUNE = tf.data.experimental.AUTOTUNE
```

In [17]:

```
### Tokenization function
def tokenize_for_bert(df):
    # create validation set
    df_train, df_val = train_test_split(df,
                                        test_size=0.2,
                                        shuffle=True,
                                        random_state=98,
                                        stratify=df["label"])

    # Tokenization
    X_train_tokenized = bert_tokenizer.batch_encode_plus(
            df_train["review"],
            return_tensors='tf',
            add_special_tokens = True,
            return_token_type_ids=True,
            padding='max_length',
            max_length=256,
            return_attention_mask = True,
            truncation='longest_first'
    )

    X_val_tokenized = bert_tokenizer.batch_encode_plus(
            df_val["review"],
            return_tensors='tf',
            add_special_tokens = True,
            return_token_type_ids=True,
            padding='max_length',
            max_length=256,
            return_attention_mask = True,
            truncation='longest_first'
    )

    # Creating TF datasets
    buffer_train = len(df_train["review"])
```

```
    training_data = tf.data.Dataset.from_tensor_slices(((X_train_tokenized["input_ids"]
                                                         X_train_tokenized["token_type_
                                                         X_train_tokenized["attention_m
                                                        df_train["label"]))
    training_data = training_data.shuffle(buffer_size=buffer_train)
    training_data = training_data.batch(batch_size)
    training_data = training_data.prefetch(buffer_size=AUTOTUNE)

    validation_data = tf.data.Dataset.from_tensor_slices(((X_val_tokenized["input_ids"]
                                                           X_val_tokenized["token_type_
                                                           X_val_tokenized["attention_m
                                                          df_val["label"]))
    validation_data = validation_data.batch(batch_size)
    validation_data = validation_data.prefetch(buffer_size=AUTOTUNE)

    return training_data, validation_data
```

In [18]:

```
training_data, validation_data = tokenize_for_bert(df_train)
```

In [19]:

```
### BERT Setup
learning_rate = 2e-5
epochs = 5

def get_bert():
    return TFBertForSequenceClassification.from_pretrained("prajjwal1/bert-tiny",
                                                           num_labels=1, from_pt = True
```

In [20]:

```
def get_compiled_bert():
    # Free up memory
    K.clear_session()

    # Build the model
    model = get_bert()

    # Print the model architecture
    print(model.summary())

    # Optimizer
    optimizer = keras.optimizers.Adam(lr=learning_rate, epsilon=1e-08)
    # Loss
    loss = keras.losses.BinaryCrossentropy(from_logits=True)

    # Compile
    model.compile(loss=loss,
                  optimizer=optimizer,
                  metrics=['accuracy'])

    return model
```

In [21]:

```
model = get_compiled_bert()
```

```
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBer
tForSequenceClassification: ['bert.embeddings.position_ids']
- This IS expected if you are initializing TFBertForSequenceClassification from a PyTorc
```

```
h model trained on another task or with another architecture (e.g. initializing a TFBert
ForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing TFBertForSequenceClassification from a Py
Torch model that you expect to be exactly identical (e.g. initializing a TFBertForSequen
ceClassification model from a BertForSequenceClassification model).
Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification were not ini
tialized from the PyTorch model and are newly initialized: ['classifier.weight', 'classi
fier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for pred
ictions and inference.
Model: "tf_bert_for_sequence_classification"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bert (TFBertMainLayer)       multiple                  4385920
_________________________________________________________________
dropout_7 (Dropout)          multiple                  0
_________________________________________________________________
classifier (Dense)           multiple                  129
=================================================================
Total params: 4,386,049
Trainable params: 4,386,049
Non-trainable params: 0
_________________________________________________________________
None
/n/home08/mhaidar/.conda/envs/tf2.5_cuda11/lib/python3.8/site-packages/tensorflow/pytho
n/keras/optimizer_v2/optimizer_v2.py:374: UserWarning: The `lr` argument is deprecated,
use `learning_rate` instead.
  warnings.warn(
```

In [22]:

```
### Train model
if train_model:
    start_time = time.time()
    training_results = model.fit(
            training_data,
            validation_data=validation_data,
            epochs=epochs,
            verbose=1)

    execution_time = (time.time() - start_time)/60.0
    print("Training execution time (mins)",execution_time)
    model.save_pretrained('model_tiny_bert_data_A_1/temp')
else:
    model = model.from_pretrained('model_tiny_bert_data_A_1/temp')
```

```
Some layers from the model checkpoint at model_tiny_bert_data_A_1/temp were not used whe
n initializing TFBertForSequenceClassification: ['dropout_7']
- This IS expected if you are initializing TFBertForSequenceClassification from the chec
kpoint of a model trained on another task or with another architecture (e.g. initializin
g a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing TFBertForSequenceClassification from the
checkpoint of a model that you expect to be exactly identical (initializing a BertForSeq
uenceClassification model from a BertForSequenceClassification model).
All the layers of TFBertForSequenceClassification were initialized from the model checkp
oint at model_tiny_bert_data_A_1/temp.
If your task is similar to the task the model of the checkpoint was trained on, you can
already use TFBertForSequenceClassification for predictions without further training.
```

In [23]:

```
### Evaluation Function
def evaluate_bert(bert_model, dataset, Y_true, only_accuracy = False):
    Y_pred = bert_model.predict(dataset)
```

```
    Y_pred = Y_pred['logits'] > 0
    acc = accuracy_score(Y_true, Y_pred)
    print(f"Accuracy: {acc}")
    if only_accuracy:
        return
    f1 = f1_score(Y_true, Y_pred)
    print(f"F1 score: {f1}")
    recall = recall_score(Y_true, Y_pred)
    print(f"Recall score: {recall}")
    precision = precision_score(Y_true, Y_pred)
    print(f"Precision score: {precision}")

    Y_pred = np.asarray([x[0] for x in Y_pred])

    TN = np.sum((Y_true == Y_pred) & (Y_pred == 0))
    TP = np.sum((Y_true == Y_pred) & (Y_pred == 1))

    FN = np.sum((Y_true != Y_pred) & (Y_pred == 0))
    FP = np.sum((Y_true != Y_pred) & (Y_pred == 1))

    print(f"TN: {TN}, TP:{TP}, FN:{FN}, FP:{FP}")
```

In [24]:

```
### Tokenize test dataset
X_test_tokenized = bert_tokenizer.batch_encode_plus(
        df_test["review"],
        return_tensors='tf',
        add_special_tokens = True,
        return_token_type_ids=True,
        padding='max_length',
        max_length=256,
        return_attention_mask = True,
        truncation='longest_first'
)

testing_data_balanced = tf.data.Dataset.from_tensor_slices(((X_test_tokenized["input_id
                                                             X_test_tokenized["token_ty
                                                             X_test_tokenized["attentio
                                                            df_test["label"]))
testing_data_balanced = testing_data_balanced.batch(batch_size)
testing_data_balanced = testing_data_balanced.prefetch(buffer_size=AUTOTUNE)
```

In [25]:

```
### evaluate on full test data
evaluate_bert(model, testing_data_balanced, df_test["label"])
```

```
WARNING:tensorflow:From /n/home08/mhaidar/.conda/envs/tf2.5_cuda11/lib/python3.8/site-pa
ckages/tensorflow/python/ops/array_ops.py:5043: calling gather (from tensorflow.python.o
ps.array_ops) with validate_indices is deprecated and will be removed in a future versio
n.
Instructions for updating:
The `validate_indices` argument has no effect. Indices are always validated on CPU and n
ever validated on GPU.
Accuracy: 0.8885326104922151
F1 score: 0.7000533712862479
Recall score: 0.6304069208586991
Precision score: 0.787
TN: 22944, TP:3935, FN:2307, FP:1065
```

In [26]:

```
### Tokenize imbalanced test dataset
X_test_imbalanced_tokenized = bert_tokenizer.batch_encode_plus(
        df_test_imbalanced["review"],
        return_tensors='tf',
        add_special_tokens = True,
        return_token_type_ids=True,
        padding='max_length',
        max_length=256,
        return_attention_mask = True,
        truncation='longest_first'
)

testing_data_imbalanced = tf.data.Dataset.from_tensor_slices(((X_test_imbalanced_tokeni
                                                               X_test_imbalanced_tokenize
                                                               X_test_imbalanced_tokenize
                                                              df_test_imbalanced["label"]
testing_data_imbalanced = testing_data_imbalanced.batch(batch_size)
testing_data_imbalanced = testing_data_imbalanced.prefetch(buffer_size=AUTOTUNE)
```

In [27]:

```
### evaluate on imbalanced test data
evaluate_bert(model, testing_data_imbalanced, df_test_imbalanced["label"])
```

```
Accuracy: 0.9399335232668566
F1 score: 0.5162523900573613
Recall score: 0.6413301662707839
Precision score: 0.432
TN: 22944, TP:810, FN:453, FP:1065
```

4. Modeling: Imbalanced Dataset

Return to contents

4.1: Using Word-Based Model

Return to contents

In [28]:

```
### Vectorization using TF-IDF
X_train_imbalanced = tfidf_vectorizer.transform(df_train_imbalanced["review"])
Y_train_imbalanced = df_train_imbalanced["label"]

### Logistic Regression
log_reg_imbalanced = LogisticRegression(random_state=98, max_iter=5000).fit(X_train_imb

### Vectorizing of test data
X_test = tfidf_vectorizer.transform(df_test["review"])
Y_test = df_test["label"]

X_test_imbalanced = tfidf_vectorizer.transform(df_test_imbalanced["review"])
Y_test_imbalanced = df_test_imbalanced["label"]

### Evaluation
print(f"The accuracy on the imbalanced train data: {log_reg_imbalanced.score(X_train_im
print(f"The accuracy on the full test data: {log_reg_imbalanced.score(X_test, Y_test):.
```

```
print(f"The accuracy on the imbalanced test data: {log_reg_imbalanced.score(X_test_imba

print()

print(f"The F1 on the imbalanced train data: {f1_score(Y_train_imbalanced, log_reg_imba
print(f"The F1 on the full test data: {f1_score(Y_test, log_reg_imbalanced.predict(X_te
print(f"The F1 on the imbalanced test data: {f1_score(Y_test_imbalanced, log_reg_imbala
```

```
The accuracy on the imbalanced train data: 0.9575
The accuracy on the full test data: 0.8210
The accuracy on the imbalanced test data: 0.9545

The F1 on the imbalanced train data: 0.3038
The F1 on the full test data: 0.2460
The F1 on the imbalanced test data: 0.2273
```

4.2: Using Context-Based Model

Return to contents

In [29]:
```
### Tokenization
training_data_imbalanced, validation_data_imbalanced = tokenize_for_bert(df_train_imbal
```

In [30]:
```
### BERT
model = get_compiled_bert()
```

```
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBer
tForSequenceClassification: ['bert.embeddings.position_ids']
- This IS expected if you are initializing TFBertForSequenceClassification from a PyTorc
h model trained on another task or with another architecture (e.g. initializing a TFBert
ForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing TFBertForSequenceClassification from a Py
Torch model that you expect to be exactly identical (e.g. initializing a TFBertForSequen
ceClassification model from a BertForSequenceClassification model).
Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification were not ini
tialized from the PyTorch model and are newly initialized: ['classifier.weight', 'classi
fier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for pred
ictions and inference.
Model: "tf_bert_for_sequence_classification"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bert (TFBertMainLayer)       multiple                  4385920
_________________________________________________________________
dropout_7 (Dropout)          multiple                  0
_________________________________________________________________
classifier (Dense)           multiple                  129
=================================================================
Total params: 4,386,049
Trainable params: 4,386,049
Non-trainable params: 0
_________________________________________________________________
None
/n/home08/mhaidar/.conda/envs/tf2.5_cuda11/lib/python3.8/site-packages/tensorflow/pytho
n/keras/optimizer_v2/optimizer_v2.py:374: UserWarning: The `lr` argument is deprecated,
use `learning_rate` instead.
  warnings.warn(
```

In [31]:
```
### Train model
if train_model:
    start_time = time.time()
    training_results = model.fit(
            training_data_imbalanced,
            validation_data=validation_data_imbalanced,
            epochs=epochs,
            verbose=1)

    execution_time = (time.time() - start_time)/60.0
    print("Training execution time (mins)",execution_time)
    model.save_pretrained('model_tiny_bert_data_B_1/temp')
else:
    model = model.from_pretrained('model_tiny_bert_data_B_1/temp')
```

```
Some layers from the model checkpoint at model_tiny_bert_data_B_1/temp were not used whe
n initializing TFBertForSequenceClassification: ['dropout_7']
- This IS expected if you are initializing TFBertForSequenceClassification from the chec
kpoint of a model trained on another task or with another architecture (e.g. initializin
g a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing TFBertForSequenceClassification from the
checkpoint of a model that you expect to be exactly identical (initializing a BertForSeq
uenceClassification model from a BertForSequenceClassification model).
All the layers of TFBertForSequenceClassification were initialized from the model checkp
oint at model_tiny_bert_data_B_1/temp.
If your task is similar to the task the model of the checkpoint was trained on, you can
already use TFBertForSequenceClassification for predictions without further training.
```

In [32]:
```
### evaluate on full test data
evaluate_bert(model, testing_data_balanced, df_test["label"])
```

```
Accuracy: 0.8444018379557701
F1 score: 0.41996303142329017
Recall score: 0.2729894264658763
Precision score: 0.9097704217832354
TN: 23840, TP:1704, FN:4538, FP:169
```

In [33]:
```
### evaluate on imbalanced test data
evaluate_bert(model, testing_data_imbalanced, df_test_imbalanced["label"])
```

```
Accuracy: 0.9573045267489712
F1 score: 0.3955182072829132
Recall score: 0.27949326999208235
Precision score: 0.6762452107279694
TN: 23840, TP:353, FN:910, FP:169
```

5. Data Augmentation Using GPT2

Return to contents

5.1: Fine-Tuning

Return to contents

In [34]:
```
## Tokenization
### Getting positive data points
df_train_positive = df_train_imbalanced[df_train_imbalanced["label"]]

### Tokenizer with a start token
gpt2_tokenizer = GPT2Tokenizer.from_pretrained("distilgpt2")
special_tokens_dict = {
        "additional_special_tokens": [
            '[start]'
        ]
}
gpt2_tokenizer.add_special_tokens(special_tokens_dict)
special_tokens = gpt2_tokenizer.added_tokens_encoder
```

In [35]:
```
### Adding start token
def format_review(row):
    output =  [special_tokens['[start]']] + gpt2_tokenizer.encode(row['review'])
    return output
formatted_reviews = df_train_positive.apply(format_review, axis = 1)
```

```
Token indices sequence length is longer than the specified maximum sequence length for t
his model (1293 > 1024). Running this sequence through the model will result in indexing
errors
```

In [36]:
```
### Padding
def pad_list(lst, value, target_length):
    if len(lst) < target_length:
        additional_pads = [value for _ in range(target_length - len(lst))]
        return lst + additional_pads
    else:
        return lst[:target_length]
pad_value = gpt2_tokenizer.eos_token_id
max_len = 256
blocks = [pad_list(x, pad_value, max_len) for x in formatted_reviews]
```

In [37]:
```
### Get the input and output ids
input_ids = [x[:-1] for x in blocks]
output_ids = [x[1:] for x in blocks]

### Create
train_data_gpt2 = tf.data.Dataset.from_tensor_slices((input_ids, output_ids))

### Shuffle
TRAIN_SHUFFLE_BUFFER_SIZE = 10000
train_data_gpt2 = train_data_gpt2.shuffle(buffer_size=TRAIN_SHUFFLE_BUFFER_SIZE)

### Batch
train_data_gpt2 = train_data_gpt2.batch(12, drop_remainder=True)
```

In [38]:
```
## GPT2
gpt_model = TFGPT2LMHeadModel.from_pretrained("distilgpt2")
gpt_model.resize_token_embeddings(len(gpt2_tokenizer))
```

```
All model checkpoint layers were used when initializing TFGPT2LMHeadModel.
```

```
All the layers of TFGPT2LMHeadModel were initialized from the model checkpoint at distil
gpt2.
If your task is similar to the task the model of the checkpoint was trained on, you can
already use TFGPT2LMHeadModel for predictions without further training.
```

Out[38]:
```
<transformers.modeling_tf_utils.TFSharedEmbeddings at 0x2ab664791f10>
```

In [39]:
```
## Training
### parameters
learning_rate = 3e-5
epsilon=1e-08
clipnorm=1.0
epochs = 30
```

In [40]:
```
### Optimizer, Loss function, and metrics
optimizer = keras.optimizers.Adam(learning_rate=learning_rate, epsilon=epsilon, clipnor
loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metric = keras.metrics.SparseCategoricalAccuracy('accuracy')

### Compile
gpt_model.compile(loss=[loss, *[None] * gpt_model.config.n_layer],
                  optimizer=optimizer,
                  metrics=[metric])
```

In [41]:
```
### Train model
if train_model:
    start_time = time.time()
    training_results = gpt_model.fit(
            train_data_gpt2,
            epochs=epochs,
            verbose=1)
    execution_time = (time.time() - start_time)/60.0
    print("Training execution time (mins)",execution_time)
    gpt_model.save_pretrained('gpt2_1/temp')
else:
    gpt_model = TFGPT2LMHeadModel.from_pretrained('gpt2_1/temp')
```

```
All model checkpoint layers were used when initializing TFGPT2LMHeadModel.

All the layers of TFGPT2LMHeadModel were initialized from the model checkpoint at gpt2_
1/temp.
If your task is similar to the task the model of the checkpoint was trained on, you can
already use TFGPT2LMHeadModel for predictions without further training.
```

5.2: Generating

Return to contents

In [42]:
```
## Generation Function
empty_row = {
    'review': ''
}
def generate_from_gpt2(model = gpt_model, input_ids = [format_review(empty_row)], n_ret
```

```
    input_ids = tf.constant(input_ids)
    outputs = model.generate(
        input_ids,
        do_sample=True,
        max_length=256,
        top_k=10,
        num_return_sequences= n_return
    )
    if n_return == 1:
        # return string
        generated_text = gpt2_tokenizer.decode(outputs[0,8:], skip_special_tokens=True)
        return generated_text
    # return list of strings
    generated_list = [
        gpt2_tokenizer.decode(outputs[x,8:], skip_special_tokens=True) for x in range(n
    ]
    return generated_list
```

In [43]:
```
generate_from_gpt2(gpt_model)
```

```
Setting `pad_token_id` to 50256 (first `eos_token_id`) to generate sequence
```

Out[43]:
```
' the convenience of the Keurig for when I need to travel or make food.  They are a big,
convenient way to get some veggies for my family.  These have a nice taste that is very
natural.  I think the main reason I bought these is because they do NOT have as many veg
gies per packet.  I was hoping for more and they are not disappointed.'
```

In [44]:
```
generate_from_gpt2(gpt_model)
```

```
Setting `pad_token_id` to 50256 (first `eos_token_id`) to generate sequence
```

Out[44]:
```
".  They are small and convenient for those with digestive issues but the taste is not a
s sweet and tasty as I expected. I would rather have a bowl of plain ol' Cheerios.  I wa
s hoping these would be like the ones I get at the grocery store.  I did not find them t
o be very filling.  The flavor is very fruity, though somewhat tart."
```

In [45]:
```
generate_from_gpt2(gpt_model)
```

```
Setting `pad_token_id` to 50256 (first `eos_token_id`) to generate sequence
```

Out[45]:
```
" I don't have the taste.  I'm not impressed.  The texture was fine too."
```

In [46]:
```
generate_from_gpt2(gpt_model, n_return=3)
```

```
Setting `pad_token_id` to 50256 (first `eos_token_id`) to generate sequence
```

Out[46]:
```
[' version of this bar which was ok but was too sweet for the health benefit.  The bar w
as good but not great either.  I like dark chocolate and dark chocolate but this is not
my favorite.  It is very filling, just not my favorite chocolate bar I have tried.',
 " aren't good either.  The taste is good but not great, either.  I am not much-loved ci
nnamon, but the flavor is ok at best, but not great.  They were a bit on-the-go for my t
aste.",
 ' a huge fan of the flavor of this candy, but I really did not like this product.  I th
ought that if I were to buy it I would at least have liked it, but I have never liked i
t.  I was a bit skeptical when it said it would be made, but it certainly did not.  I am
not sure if I will buy it again, but if it is, I will have to give it away to someone el
se to make sure it is good for them.']
```

In [47]:
```
## Creating Augmented Dataset
```

```
def create_augmented_dataset(generation_function, df = df_train_imbalanced, desired_rat
    pos_neg_desired_ratio = desired_ratio / (1 - desired_ratio)
    n_positive = df[df["label"] == True ].shape[0]
    n_negative = df[df["label"] == False].shape[0]
    n_desired_positive = int(pos_neg_desired_ratio * n_negative)
    n_new_positive_examples = n_desired_positive - n_positive
    print(n_new_positive_examples)
    if n_new_positive_examples <= 0:
        return df
    new_reviews = []
    new_labels = []
    for _ in tqdm(range(int(n_new_positive_examples/n_batch))):
        review_batch = generation_function(n_return = n_batch)
        if n_batch == 1:
            new_reviews.append(review_batch)
            new_labels.append(True)
        else:
            for review in review_batch:
                new_reviews.append(review)
                new_labels.append(True)
    print(new_reviews)
    print(len(new_reviews), len(new_labels))
    new_reviews_df = pd.DataFrame({
        "label": new_labels,
        "review": new_reviews
    })
    return new_reviews_df
```

In [48]:

```
if train_new_data:
    gpt_augmented_df = create_augmented_dataset(generate_from_gpt2, desired_ratio=0.2,
    gpt_augmented_df.to_csv("data/gpt2_augmentation_80_to_20_v1.csv")
else:
    gpt_augmented_df = pd.read_csv("data/gpt2_augmentation_80_to_20_v1.csv")
```

In [49]:

```
gpt_augmented_df.head()
```

Out[49]:

	label	review
0	True	thought it would be, but I didn't really like...
1	True	MIO Liquid Peach Mio. I have tried other fla...
2	True	, I really like it, I was so excited. Then, a...
3	True	MIO. It's an energy drink, with the added caf...
4	True	little sweet, but not for me. It's a lot str...

In [50]:

```
df_train_gpt = pd.concat([df_train_imbalanced, gpt_augmented_df])
```

In [51]:

```
np.mean(df_train_gpt["label"])
```

Out[51]:

```
0.19992835303624837
```

6. Modeling: GPT2 Augmented Dataset

Return to contents

6.1: Using Word-Based Model

Return to contents

In [52]:
```
### Tokenization
X_train_gpt = tfidf_vectorizer.fit_transform(df_train_gpt["review"])
Y_train_gpt = df_train_gpt["label"]

### Training
log_reg_gpt = LogisticRegression(random_state=98, max_iter=5000).fit(X_train_gpt, Y_tra

### Evaluation
X_test = tfidf_vectorizer.transform(df_test["review"])
Y_test = df_test["label"]

X_test_imbalanced = tfidf_vectorizer.transform(df_test_imbalanced["review"])
Y_test_imbalanced = df_test_imbalanced["label"]

print(f"The accuracy on the train data: {log_reg_gpt.score(X_train_gpt, Y_train_gpt):.4
print(f"The accuracy on the test data: {log_reg_gpt.score(X_test, Y_test):.4f}")
print(f"The accuracy on the imbalanced test data: {log_reg_gpt.score(X_test_imbalanced,

print()

print(f"The F1 on the train data: {f1_score(Y_train_gpt, log_reg_gpt.predict(X_train_gp
print(f"The F1 on the test data: {f1_score(Y_test, log_reg_gpt.predict(X_test)):.4f}")
print(f"The F1 on the imbalanced test data: {f1_score(Y_test_imbalanced, log_reg_gpt.pr
```

```
The accuracy on the train data: 0.9446
The accuracy on the test data: 0.8366
The accuracy on the imbalanced test data: 0.9452

The F1 on the train data: 0.8502
The F1 on the test data: 0.4209
The F1 on the imbalanced test data: 0.3515
```

6.2: Using Context-Based Model

Return to contents

In [53]:
```
### Tokenization
training_data_aug_gpt2, val_data_aug_gpt2 = tokenize_for_bert(df_train_gpt)
```

In [54]:
```
### BERT
model = get_compiled_bert()
```

```
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBer
```

```
tForSequenceClassification: ['bert.embeddings.position_ids']
- This IS expected if you are initializing TFBertForSequenceClassification from a PyTorc
h model trained on another task or with another architecture (e.g. initializing a TFBert
ForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing TFBertForSequenceClassification from a Py
Torch model that you expect to be exactly identical (e.g. initializing a TFBertForSequen
ceClassification model from a BertForSequenceClassification model).
Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification were not ini
tialized from the PyTorch model and are newly initialized: ['classifier.weight', 'classi
fier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for pred
ictions and inference.
Model: "tf_bert_for_sequence_classification"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bert (TFBertMainLayer)       multiple                  4385920
_________________________________________________________________
dropout_7 (Dropout)          multiple                  0
_________________________________________________________________
classifier (Dense)           multiple                  129
=================================================================
Total params: 4,386,049
Trainable params: 4,386,049
Non-trainable params: 0
_________________________________________________________________
None
/n/home08/mhaidar/.conda/envs/tf2.5_cuda11/lib/python3.8/site-packages/tensorflow/pytho
n/keras/optimizer_v2/optimizer_v2.py:374: UserWarning: The `lr` argument is deprecated,
use `learning_rate` instead.
  warnings.warn(
```

In [55]:

```
### Train model
if train_model:
    start_time = time.time()
    training_results = model.fit(
            training_data_aug_gpt2,
            validation_data=val_data_aug_gpt2,
            epochs=epochs,
            verbose=1)

    execution_time = (time.time() - start_time)/60.0
    print("Training execution time (mins)",execution_time)
    model.save_pretrained('model_tiny_bert_data_C_2/temp')
else:
    model = model.from_pretrained('model_tiny_bert_data_C_2/temp')
```

```
Some layers from the model checkpoint at model_tiny_bert_data_C_2/temp were not used whe
n initializing TFBertForSequenceClassification: ['dropout_7']
- This IS expected if you are initializing TFBertForSequenceClassification from the chec
kpoint of a model trained on another task or with another architecture (e.g. initializin
g a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing TFBertForSequenceClassification from the
checkpoint of a model that you expect to be exactly identical (initializing a BertForSeq
uenceClassification model from a BertForSequenceClassification model).
All the layers of TFBertForSequenceClassification were initialized from the model checkp
oint at model_tiny_bert_data_C_2/temp.
If your task is similar to the task the model of the checkpoint was trained on, you can
already use TFBertForSequenceClassification for predictions without further training.
```

In [56]:

```
### evaluate on full test data
```

```
evaluate_bert(model, testing_data_balanced, df_test["label"])
```

```
Accuracy: 0.8403358566658954
F1 score: 0.4289430125325136
Recall score: 0.2906119833386735
Precision score: 0.8185920577617328
TN: 23607, TP:1814, FN:4428, FP:402
```

In [57]:
```
### evaluate on imbalanced test data
evaluate_bert(model, testing_data_imbalanced, df_test_imbalanced["label"])
```

```
Accuracy: 0.9489949351060463
F1 score: 0.36844683978441944
Recall score: 0.2977038796516231
Precision score: 0.4832904884318766
TN: 23607, TP:376, FN:887, FP:402
```

7. Data Augmentation Using EDA

Return to contents

This is done on a separate script using the instructions and package at:

https://github.com/jasonwei20/eda_nlp

In [58]:
```
with open('data/input_eda.txt', 'w') as f:
    for (line, label) in zip(df_train_imbalanced["review"], df_train_imbalanced["label"
        if label:
            f.write(f"1\t{line}\n")
```

In [59]:
```
### Read and Process EDA Generated Data
with open('data/eda_augmentation_80_20_v3.txt') as f:
    lines = f.readlines()

lines = [line[2:] for line in lines]
labels = [True for line in lines]

eda_augmented_df = pd.DataFrame({
    "label": labels,
    "review": lines
})
```

In [60]:
```
df_train_eda = pd.concat([df_train_imbalanced[df_train_imbalanced['label'] == 0], eda_a
```

In [61]:
```
np.mean(df_train_eda['label'])
```

Out[61]:
```
0.20831787642718766
```

8. Modeling: EDA Augmented Dataset

Return to contents

8.1: Using Word-Based Model

Return to contents

In [62]:
```
### Tokenization
X_train_eda = tfidf_vectorizer.fit_transform(df_train_eda["review"])
Y_train_eda = df_train_eda["label"]

## Training
log_reg_eda = LogisticRegression(random_state=98, max_iter=5000).fit(X_train_eda, Y_tra

# Evaluation
X_test = tfidf_vectorizer.transform(df_test["review"])
Y_test = df_test["label"]

X_test_imbalanced = tfidf_vectorizer.transform(df_test_imbalanced["review"])
Y_test_imbalanced = df_test_imbalanced["label"]

print(f"The accuracy on the train data: {log_reg_eda.score(X_train_eda, Y_train_eda):.4
print(f"The accuracy on the test data: {log_reg_eda.score(X_test, Y_test):.4f}")
print(f"The accuracy on the imbalanced test data: {log_reg_eda.score(X_test_imbalanced,

print()

print(f"The F1 on the train data: {f1_score(Y_train_eda, log_reg_eda.predict(X_train_ed
print(f"The F1 on the test data: {f1_score(Y_test, log_reg_eda.predict(X_test)):.4f}")
print(f"The F1 on the imbalanced test data: {f1_score(Y_test_imbalanced, log_reg_eda.pr
```

```
The accuracy on the train data: 0.9682
The accuracy on the test data: 0.8189
The accuracy on the imbalanced test data: 0.9475

The F1 on the train data: 0.9203
The F1 on the test data: 0.2715
The F1 on the imbalanced test data: 0.2264
```

8.2: Using Context-Based Model

Return to contents

In [63]:
```
### Tokenization
training_data_aug_eda, val_data_aug_eda = tokenize_for_bert(df_train_eda)
```

In [64]:
```
### BERT
model = get_compiled_bert()
```

```
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBer
tForSequenceClassification: ['bert.embeddings.position_ids']
- This IS expected if you are initializing TFBertForSequenceClassification from a PyTorc
h model trained on another task or with another architecture (e.g. initializing a TFBert
ForSequenceClassification model from a BertForPreTraining model).
```

```
- This IS NOT expected if you are initializing TFBertForSequenceClassification from a Py
Torch model that you expect to be exactly identical (e.g. initializing a TFBertForSequen
ceClassification model from a BertForSequenceClassification model).
Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification were not ini
tialized from the PyTorch model and are newly initialized: ['classifier.weight', 'classi
fier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for pred
ictions and inference.
Model: "tf_bert_for_sequence_classification"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bert (TFBertMainLayer)       multiple                  4385920
_________________________________________________________________
dropout_7 (Dropout)          multiple                  0
_________________________________________________________________
classifier (Dense)           multiple                  129
=================================================================
Total params: 4,386,049
Trainable params: 4,386,049
Non-trainable params: 0
_________________________________________________________________
None
/n/home08/mhaidar/.conda/envs/tf2.5_cuda11/lib/python3.8/site-packages/tensorflow/pytho
n/keras/optimizer_v2/optimizer_v2.py:374: UserWarning: The `lr` argument is deprecated,
use `learning_rate` instead.
  warnings.warn(
```

In [65]:

```
### Train model
if train_model:
    start_time = time.time()
    training_results = model.fit(
            training_data_aug_eda,
            validation_data=val_data_aug_eda,
            epochs=epochs,
            verbose=1)

    execution_time = (time.time() - start_time)/60.0
    print("Training execution time (mins)",execution_time)
    model.save_pretrained('model_tiny_bert_data_E_2/temp')
else:
    model = model.from_pretrained('model_tiny_bert_data_E_2/temp')
```

```
Some layers from the model checkpoint at model_tiny_bert_data_E_2/temp were not used whe
n initializing TFBertForSequenceClassification: ['dropout_7']
- This IS expected if you are initializing TFBertForSequenceClassification from the chec
kpoint of a model trained on another task or with another architecture (e.g. initializin
g a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing TFBertForSequenceClassification from the
checkpoint of a model that you expect to be exactly identical (initializing a BertForSeq
uenceClassification model from a BertForSequenceClassification model).
All the layers of TFBertForSequenceClassification were initialized from the model checkp
oint at model_tiny_bert_data_E_2/temp.
If your task is similar to the task the model of the checkpoint was trained on, you can
already use TFBertForSequenceClassification for predictions without further training.
```

In [66]:

```
### evaluate on full test data
evaluate_bert(model, testing_data_balanced, df_test["label"])
```

```
Accuracy: 0.7930977488347493
F1 score: 0.003819831290784657
```

```
Recall score: 0.0019224607497596924
Precision score: 0.2926829268292683
TN: 23980, TP:12, FN:6230, FP:29
```

In [67]:
```
### evaluate on imbalanced test data
evaluate_bert(model, testing_data_imbalanced, df_test_imbalanced["label"])
```

```
Accuracy: 0.9489949351060463
F1 score: 0.004633204633204634
Recall score: 0.0023752969121140144
Precision score: 0.09375
TN: 23980, TP:3, FN:1260, FP:29
```

Part 9: The Distribution of the New Samples

Return to contents

9.1: TSNE Plots

Return to contents

In [68]:
```
### Get the largest trained model
bert_base_tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',do_lower_case=T
model = TFBertModel.from_pretrained('model_bert_data_A_1/temp')
```

```
Some layers from the model checkpoint at model_bert_data_A_1/temp were not used when ini
tializing TFBertModel: ['dropout_37', 'classifier']
- This IS expected if you are initializing TFBertModel from the checkpoint of a model tr
ained on another task or with another architecture (e.g. initializing a BertForSequenceC
lassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing TFBertModel from the checkpoint of a mode
l that you expect to be exactly identical (initializing a BertForSequenceClassification
model from a BertForSequenceClassification model).
All the layers of TFBertModel were initialized from the model checkpoint at model_bert_d
ata_A_1/temp.
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.
```

In [69]:
```
### Mark each dataset
df_train['source'] = 'original'
df_train_positive = df_train[df_train['label'] == True]
df_train_negative = df_train[df_train['label'] == False]

non_eda_indices = [i for i in range(len(eda_augmented_df)) if i%5 == 4]
eda_only_augmented_df = eda_augmented_df.drop(non_eda_indices)
eda_only_augmented_df['source'] = 'EDA'

gpt_augmented_df['source'] = 'GPT2'
```

In [70]:
```
### Sample 500 data points from each one
n_samples = 500
np.random.seed(98)
```

```
df_train_full = pd.concat([df_train_positive.iloc[     np.random.choice(len(df_train_po
                           df_train_negative.iloc[     np.random.choice(len(df_train_ne
                           eda_only_augmented_df.iloc[ np.random.choice(len(eda_only_au
                           gpt_augmented_df.iloc[      np.random.choice(len(gpt_augment
                          ], ignore_index=True)
```

In [71]:

```
### Tokenize all the data
text_encoding = bert_base_tokenizer.batch_encode_plus(df_train_full['review'],
                                                      return_tensors='tf',
                                                      add_special_tokens = True,
                                                      return_token_type_ids=True,
                                                      padding='max_length',
                                                      max_length=256,
                                                      return_attention_mask = True,
                                                      truncation='longest_first')
text_encoding_dataset = tf.data.Dataset.from_tensor_slices((text_encoding["input_ids"],
                                                            text_encoding["token_type_
                                                            text_encoding["attention_m
text_encoding_dataset = text_encoding_dataset.batch(batch_size)
text_encoding_dataset = text_encoding_dataset.prefetch(buffer_size=AUTOTUNE)
```

In [72]:

```
### Get pooled outputs, which are the outputs of the last layer of Bert
embedding_to_concat = []
for batch in tqdm(text_encoding_dataset):
    batch_embedding = model(batch)
    embedding_to_concat.append(batch_embedding['pooler_output'])
text_hidden_layer = tf.concat(embedding_to_concat, axis = 0).numpy()
```

```
100%|██████████| 250/250 [00:20<00:00, 12.21it/s]
```

In [73]:

```
### Get TSNE components
text_tsne_representation = TSNE(n_components=2, random_state = 98).fit_transform(text_h
```

```
/n/home08/mhaidar/.conda/envs/tf2.5_cuda11/lib/python3.8/site-packages/sklearn/manifold/
_t_sne.py:780: FutureWarning: The default initialization in TSNE will change from 'rando
m' to 'pca' in 1.2.
  warnings.warn(
/n/home08/mhaidar/.conda/envs/tf2.5_cuda11/lib/python3.8/site-packages/sklearn/manifold/
_t_sne.py:790: FutureWarning: The default learning rate in TSNE will change from 200.0 t
o 'auto' in 1.2.
  warnings.warn(
```

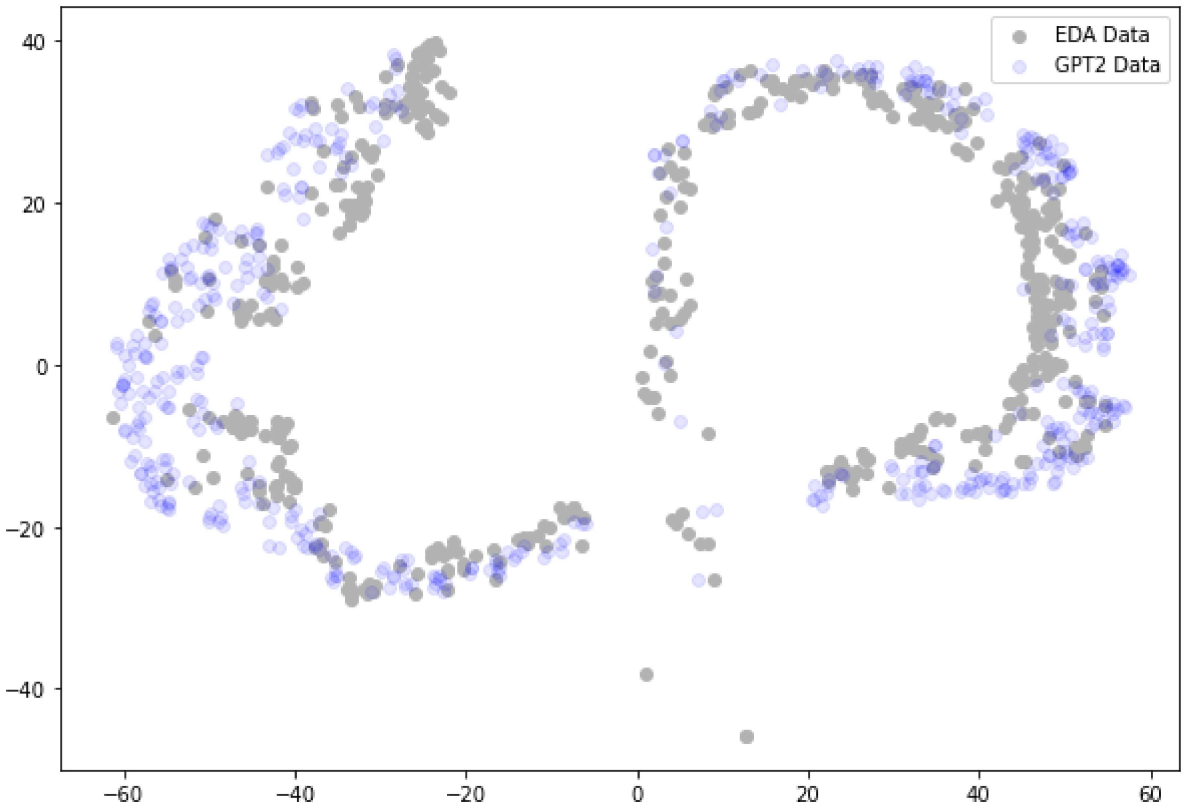
In [74]:

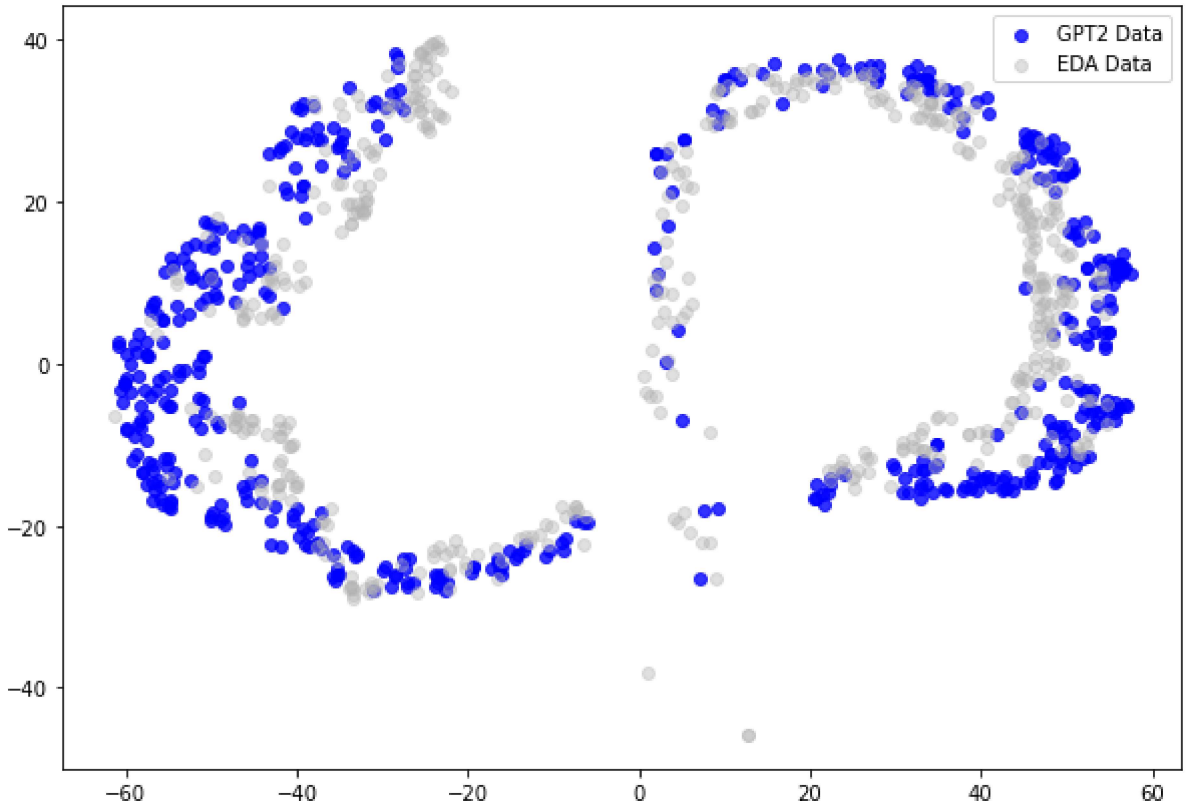
```
### Get the indices of each df
original_positive = (df_train_full['label'] == True) & (df_train_full['source'] == 'ori
original_negative = (df_train_full['label'] == False) & (df_train_full['source'] == 'or
eda_positive = (df_train_full['source'] == 'EDA')
gpt_positive = (df_train_full['source'] == 'GPT2')
```

In [75]:

```
### Separate the TSNE values for each df
original_positive_tsne = text_tsne_representation[original_positive]
original_negative_tsne = text_tsne_representation[original_negative]
eda_tsne = text_tsne_representation[eda_positive]
gpt_tsne = text_tsne_representation[gpt_positive]
```

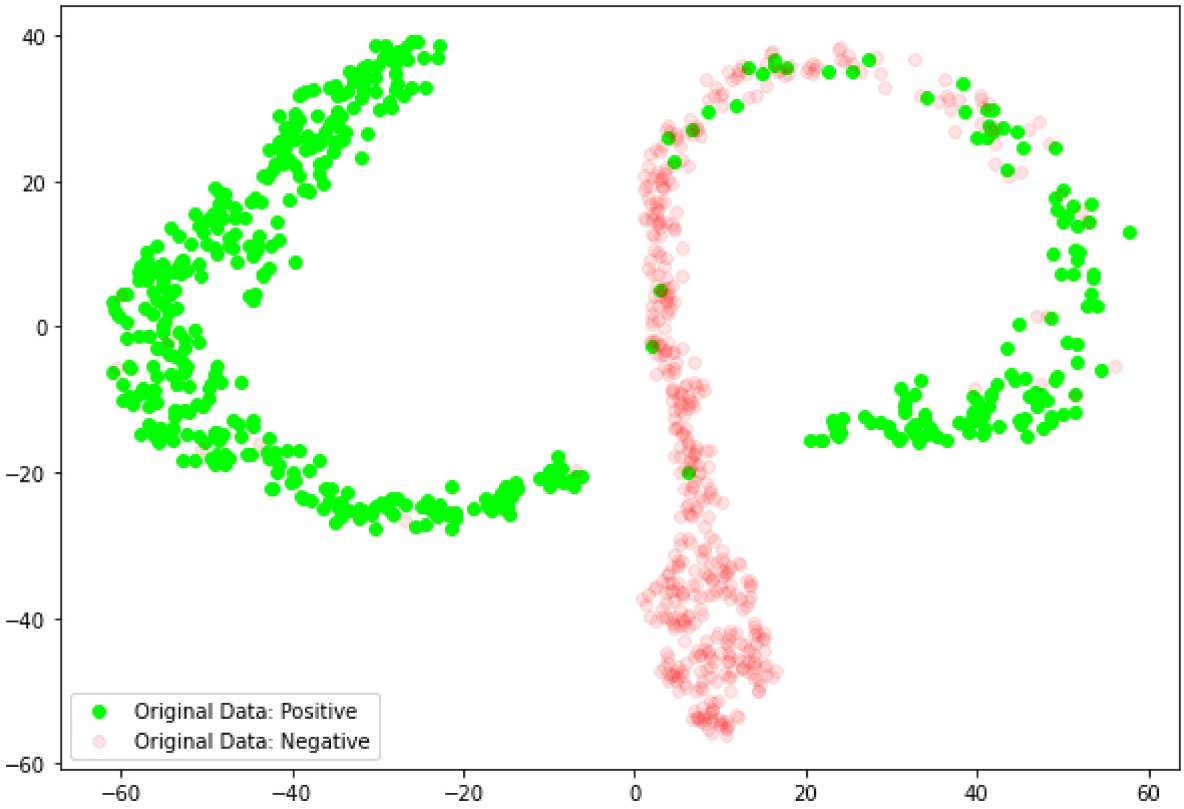
In [76]:
```
### Plot
plt.figure(figsize=(10, 7))
plt.scatter(eda_tsne[:,0], eda_tsne[:,1],
            color = (0.7, 0.7, 0.7, 1),
           label = "EDA Data")
plt.scatter(gpt_tsne[:,0], gpt_tsne[:,1],
            color = (0, 0, 1, 0.1),
           label = "GPT2 Data")

plt.legend()
plt.show()

plt.figure(figsize=(10, 7))
plt.scatter(gpt_tsne[:,0], gpt_tsne[:,1],
            color = (0, 0, 1, 0.8),
           label = "GPT2 Data")
plt.scatter(eda_tsne[:,0], eda_tsne[:,1],
            color = (0.7, 0.7, 0.7, 0.4),
           label = "EDA Data")
plt.legend()
plt.show()
```

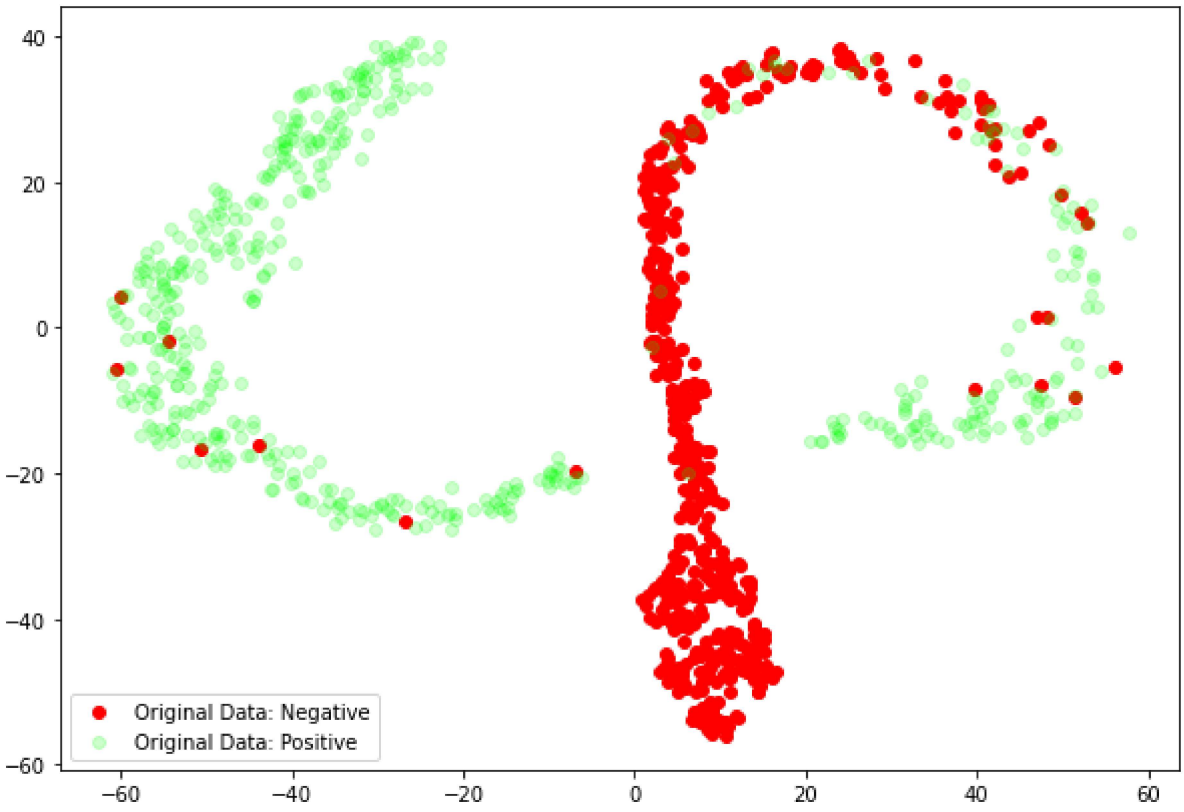




In [77]:

```
plt.figure(figsize=(10, 7))
plt.scatter(original_positive_tsne[:,0], original_positive_tsne[:,1],
            color = (0,1,0,1),
            label = "Original Data: Positive")
plt.scatter(original_negative_tsne[:,0], original_negative_tsne[:,1],
            color = (1, 0, 0, 0.1),
           label = "Original Data: Negative")
plt.legend()
plt.show()

plt.figure(figsize=(10, 7))
plt.scatter(original_negative_tsne[:,0], original_negative_tsne[:,1],
            color = (1, 0, 0, 1),
           label = "Original Data: Negative")
plt.scatter(original_positive_tsne[:,0], original_positive_tsne[:,1],
            color = (0,1,0,0.2),
            label = "Original Data: Positive")
plt.legend()
plt.show()
```

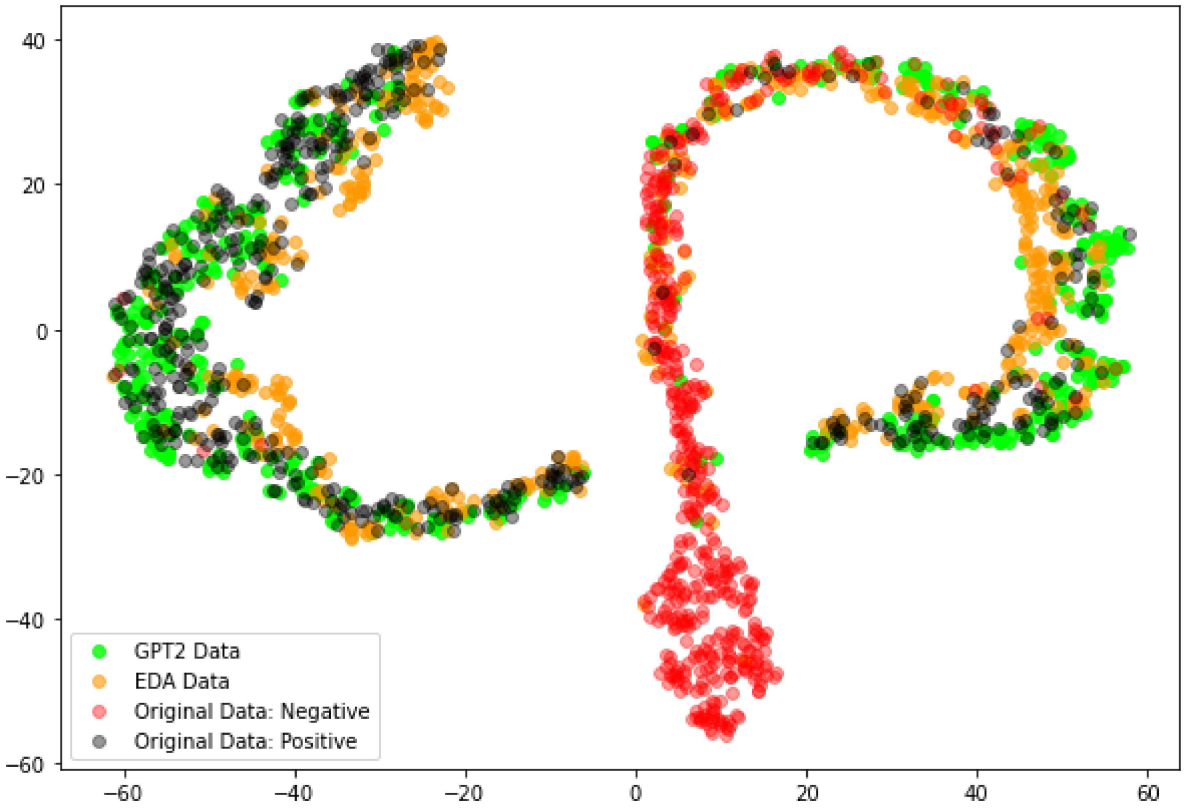




In [78]:

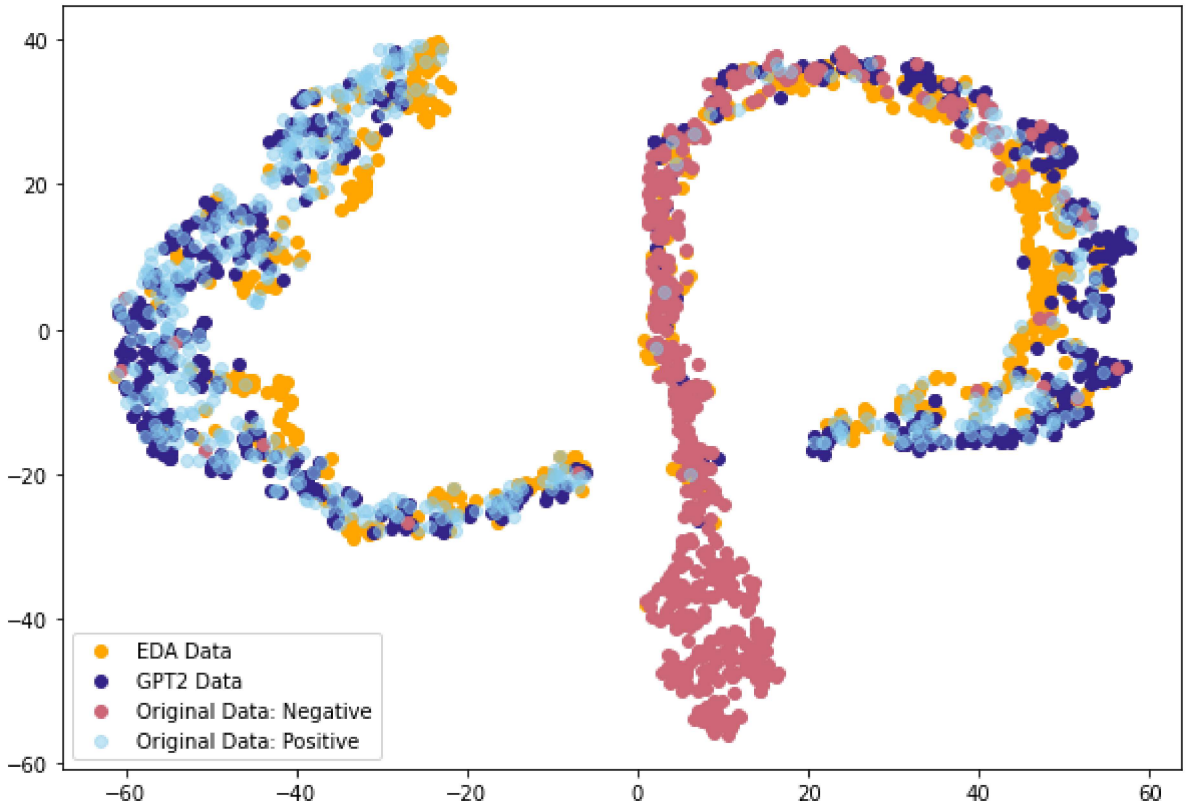
```
plt.figure(figsize=(10, 7))
plt.scatter(gpt_tsne[:,0], gpt_tsne[:,1],
            color = (0, 1, 0, 0.8),
           label = "GPT2 Data")
plt.scatter(eda_tsne[:,0], eda_tsne[:,1],
            color = (1, 0.6, 0, 0.6),
           label = "EDA Data")
```

```
plt.scatter(original_negative_tsne[:,0], original_negative_tsne[:,1],
            color = (1, 0, 0, 0.4),
           label = "Original Data: Negative")
plt.scatter(original_positive_tsne[:,0], original_positive_tsne[:,1],
            color = (0, 0, 0, 0.4),
            label = "Original Data: Positive")
plt.legend()
plt.show()
```



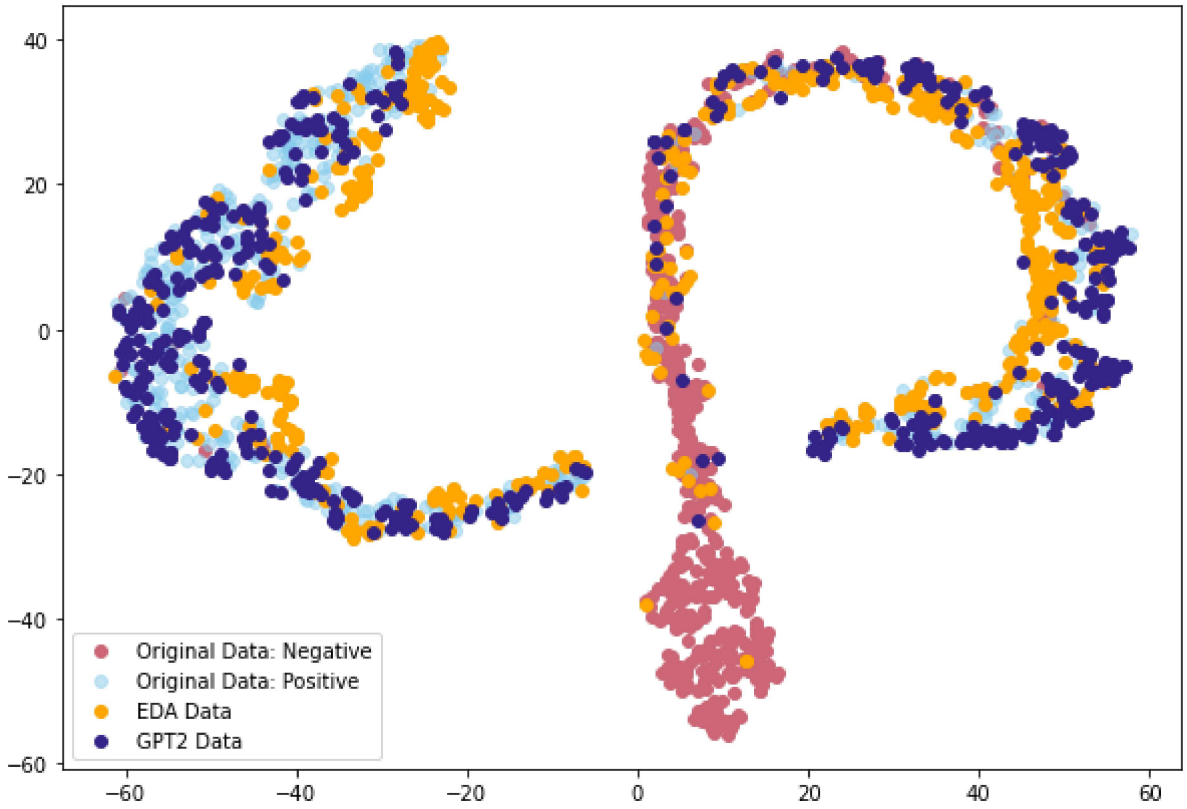
In [79]:

```
plt.figure(figsize=(10, 7))
plt.scatter(eda_tsne[:,0], eda_tsne[:,1],
            color = "orange",
           label = "EDA Data")
plt.scatter(gpt_tsne[:,0], gpt_tsne[:,1],
            color = "#332288",
           label = "GPT2 Data")
plt.scatter(original_negative_tsne[:,0], original_negative_tsne[:,1],
            color = "#CC6677",
           label = "Original Data: Negative")
plt.scatter(original_positive_tsne[:,0], original_positive_tsne[:,1],
            color = (0.53, 0.8, 0.93, 0.5),
            label = "Original Data: Positive")

plt.legend()
plt.show()
```



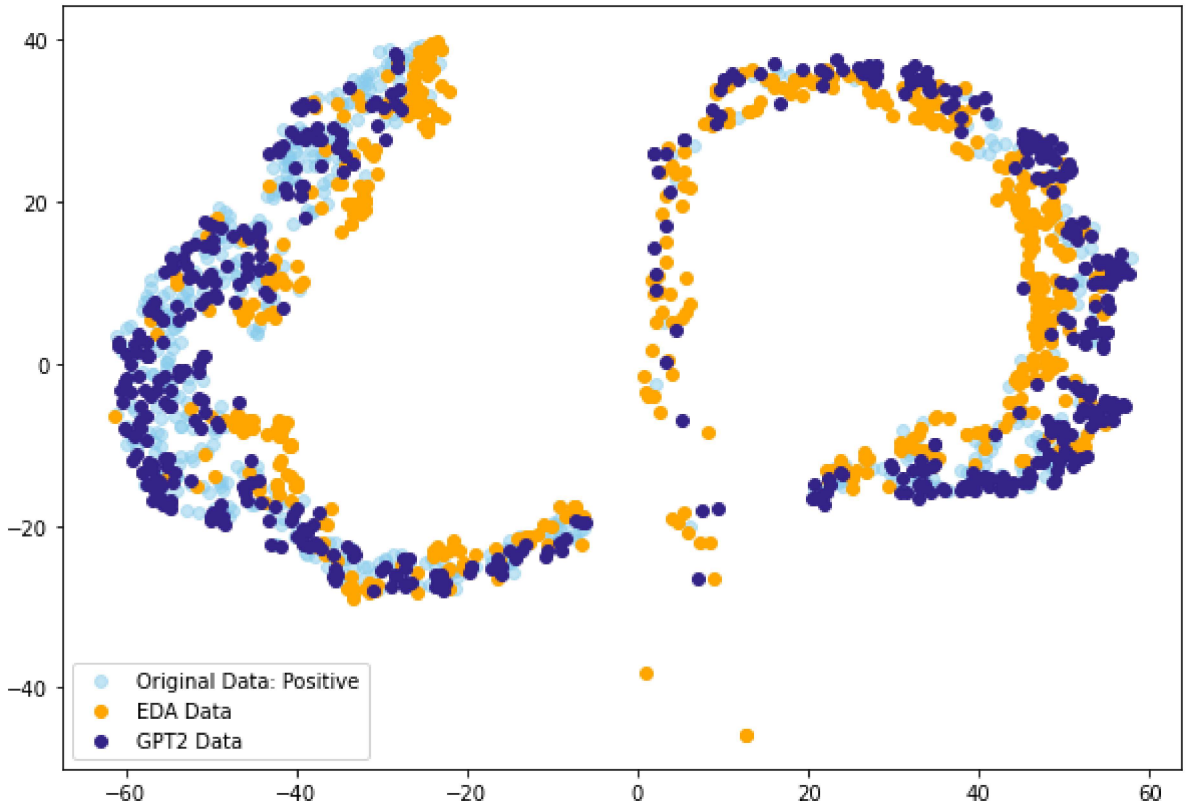
In [80]:

```
plt.figure(figsize=(10, 7))
plt.scatter(original_negative_tsne[:,0], original_negative_tsne[:,1],
            color = "#CC6677",
           label = "Original Data: Negative")
plt.scatter(original_positive_tsne[:,0], original_positive_tsne[:,1],
            color = (0.53, 0.8, 0.93, 0.5),
            label = "Original Data: Positive")
plt.scatter(eda_tsne[:,0], eda_tsne[:,1],
            color = "orange",
           label = "EDA Data")
plt.scatter(gpt_tsne[:,0], gpt_tsne[:,1],
            color = "#332288",
           label = "GPT2 Data")

plt.legend()
plt.show()
```



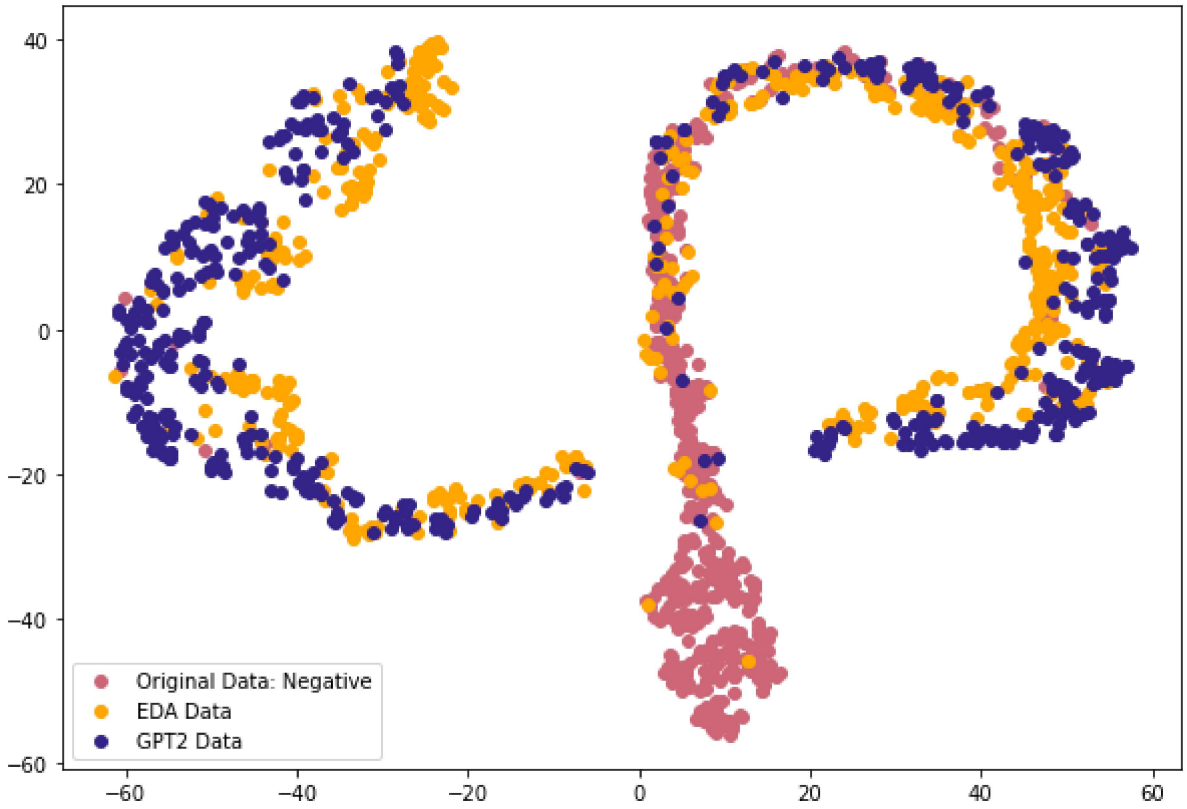
In [81]:

```
plt.figure(figsize=(10, 7))
plt.scatter(original_positive_tsne[:,0], original_positive_tsne[:,1],
            color = (0.53, 0.8, 0.93, 0.5),
            label = "Original Data: Positive")
plt.scatter(eda_tsne[:,0], eda_tsne[:,1],
            color = "orange",
           label = "EDA Data")
plt.scatter(gpt_tsne[:,0], gpt_tsne[:,1],
            color = "#332288",
           label = "GPT2 Data")

plt.legend()
plt.show()
```



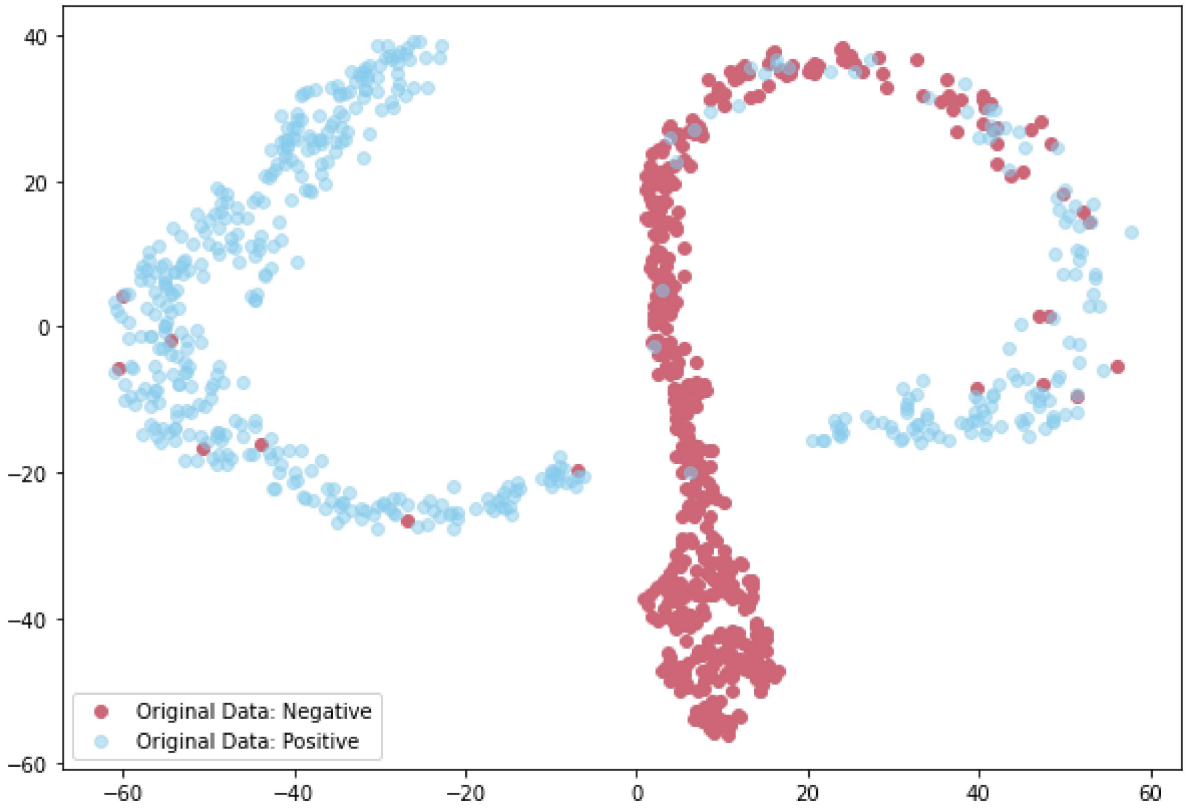
In [82]:

```
plt.figure(figsize=(10, 7))
plt.scatter(original_negative_tsne[:,0], original_negative_tsne[:,1],
            color = "#CC6677",
           label = "Original Data: Negative")
plt.scatter(eda_tsne[:,0], eda_tsne[:,1],
            color = "orange",
           label = "EDA Data")
plt.scatter(gpt_tsne[:,0], gpt_tsne[:,1],
            color = "#332288",
           label = "GPT2 Data")

plt.legend()
plt.show()
```



In [83]:

```
plt.figure(figsize=(10, 7))
plt.scatter(original_negative_tsne[:,0], original_negative_tsne[:,1],
            color = "#CC6677",
           label = "Original Data: Negative")
plt.scatter(original_positive_tsne[:,0], original_positive_tsne[:,1],
            color = (0.53, 0.8, 0.93, 0.5),
            label = "Original Data: Positive")
plt.legend()
plt.show()
```



9.2: Preserving Labels

Return to contents

In [84]:
```
### Get the largest trained model
model = TFBertForSequenceClassification.from_pretrained('model_bert_data_A_1/temp')

### Optimizer
optimizer = keras.optimizers.Adam(lr=learning_rate, epsilon=1e-08)
### Loss
loss = keras.losses.BinaryCrossentropy(from_logits=True)

### Compile
model.compile(loss=loss,
                  optimizer=optimizer,
                  metrics=['accuracy'])
```

```
Some layers from the model checkpoint at model_bert_data_A_1/temp were not used when ini
tializing TFBertForSequenceClassification: ['dropout_37']
- This IS expected if you are initializing TFBertForSequenceClassification from the chec
kpoint of a model trained on another task or with another architecture (e.g. initializin
g a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing TFBertForSequenceClassification from the
checkpoint of a model that you expect to be exactly identical (initializing a BertForSeq
uenceClassification model from a BertForSequenceClassification model).
All the layers of TFBertForSequenceClassification were initialized from the model checkp
oint at model_bert_data_A_1/temp.
If your task is similar to the task the model of the checkpoint was trained on, you can
already use TFBertForSequenceClassification for predictions without further training.
/n/home08/mhaidar/.conda/envs/tf2.5_cuda11/lib/python3.8/site-packages/tensorflow/pytho
n/keras/optimizer_v2/optimizer_v2.py:374: UserWarning: The `lr` argument is deprecated,
```

```
use `learning_rate` instead.
  warnings.warn(
```

In [85]:
```
### Evaluation Function on positive datasets
def evaluate_on_positive(model, df):
    text_encoding_df = bert_base_tokenizer.batch_encode_plus(df['review'],
                                                             return_tensors='tf',
                                                             add_special_tokens = True,
                                                             return_token_type_ids=True,
                                                             padding='max_length',
                                                             max_length=256,
                                                             return_attention_mask = Tru
                                                             truncation='longest_first')

    text_encoding_dataset_df = tf.data.Dataset.from_tensor_slices(((text_encoding_df["i
                                                                    text_encoding_df["
                                                                    text_encoding_df["
                                                                   df['label']))
    text_encoding_dataset_df = text_encoding_dataset_df.batch(batch_size)
    text_encoding_dataset_df = text_encoding_dataset_df.prefetch(buffer_size=AUTOTUNE)

    model.evaluate(text_encoding_dataset_df)
```

In [86]:
```
evaluate_on_positive(model, gpt_augmented_df)
```

```
2368/2368 [==============================] - 171s 71ms/step - loss: 0.9596 - accuracy:
0.7632
```

In [87]:
```
evaluate_on_positive(model, eda_only_augmented_df)
```

```
2527/2527 [==============================] - 179s 71ms/step - loss: 1.4906 - accuracy:
0.6416
```

In [89]:
```
evaluate_on_positive(model, df_train_positive)
```

```
3121/3121 [==============================] - 221s 71ms/step - loss: 0.3316 - accuracy:
0.9090
```

In [ ]:
```

```