a3-Pretrained-Language-Models-dan-jang

November 16, 2023

1 CS410: Natural Language Processing, Fall 2023

1.1 A3: Pretrained Language Models, Dan Jang - 11/12/2023

Description of Assignment

Introduction Using the same training & testing datasets from our first & second assignments, in this assignment, *A3: Pretrained Language Models*, we will be exploring & comparing the performance of two, specific ***Pretrained Language Models*** (**PLMs**):

1. *Google AI*'s ***`BERT`*** (*Bidirectional Encoder Representations from Transformers*)

&

2. *OpenAI*'s ***`GPT-2`*** (*Generative Pretrained Transformer 2*).

Like the previous two assignment, this assignment focuses on implementing a text-classification model that predicts sentiment & the same training/testing datasets, where in *A3* specifically, we (where, in comparison to our text-classification model approach in our previous *A2* assignment, we used **pretrained embeddings** from the `Word2Vec` and `GloVe` **models** instead).

Data Preparation Like our previous two assignments, we will use a pair of training & testing datasets containing product customer reviews, which is named the "*Multilingual Amazon Reviews Corpus*", in a `.json` container format, with several columns. The assignment will focus on a smaller subset of the original dataset, where we will focus on **two (2) columns**: 1. "review_title" - self-explanatory 2. "stars" - an integer, either 1 or 5, where the former indicates "negative" and 5 indicates "positive."

There will be a training set & a test set.

We will load the dataset using Python & use respective libraries to implement our text-classification model.

In contrary to the last two assignments, there will be no preprocessing done manually, except in using each *PLM*'s specific tokenizers to prepare our data for each run of our text-classification model implementations.

Implementation of Pretrained Language Models (PLMs) We will use *HuggingFace* libraries (e.g. `transformers`) to access, then correspondingly experiment with the ***`BERT`*** and ***`GPT-2`*** **pretrained language models** (***PLMs***), focusing on these aspects:

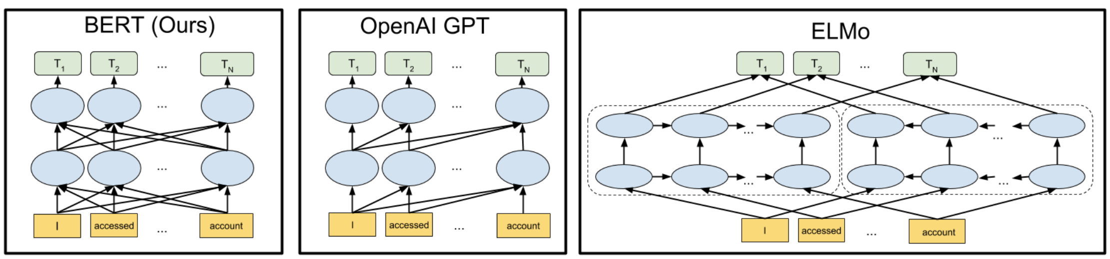
1. **Comparison** of the two pretrained language models.

2. **Model Evaluation, Results, & Analysis** (and comparison) of our two ***PLMs***.

BERT (Bidirectional Encoder Representations from Transformers) Pretrained Language Model (PLM) The first ***PLM*** we will be using is ***`BERT`*** (*Bidirectional Encoder Representations from Transformers*), which was first described in a paper published in May 24th, 2019, by the *Google AI Language* Team, authored by Researchers Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova.

`BERT` (*Bidirectional Encoder Representations from Transformers*) was created-in-mind, to be a model designed particularly for training "*their own state-of-the-art question answering system*" - built upon previous natural language processing research in pretraining contextual representations, e.g. *Semi-Supervised Sequence Learning*, *Generative Pre-Training*, etc. (*Google AI Language*, Devlin & Chang, November 2 2018).

Moreover, ***`BERT`*** (*Bidirectional Encoder Representations from Transformers*), at the time, this model was the "*first* ***deeply directional***, **unsupervised** *language representation, pretrained using only a plain text corpus*", using *Wikipedia* as its training source (2018).



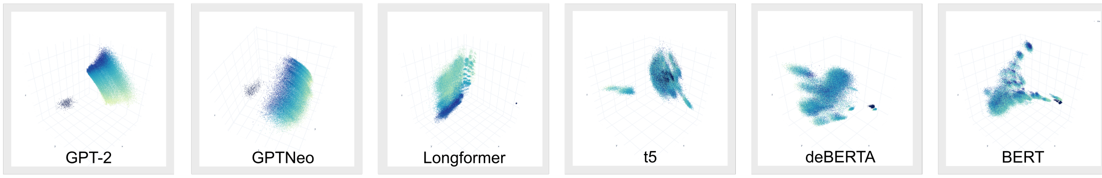
OpenAI's GPT-2 (Generative Pretrained Transformer) Pretrained Language Model (PLM) An almost infamously named ***PLM*** model, ***`GPT-2`*** (*Generative Pretrained Transformer 2*) is the second predecessor of the ***`ChatGPT`*** model that is known popularly today (as of November 10th, 2023, ***`ChatGPT`*** utilizes specifically for its ***PLM*** models, ***`GPT-3.5 Turbo`*** for free & ***`GPT-4+`*** [/w ***`Vision`***] for premium users), our second ***PLM OpenAI***'s ***`GPT-2`*** (*Generative Pretrained Transformer 2*).

On February 14th, 2019, *OpenAI* describes first in a blog post & technical paper, their announcement & description of their new "*large-scale unsupervised language model*", named ***`GPT-2`*** (*OpenAI*, February 14 2019).

In ***`GPT-2`***'s technical paper, *OpenAI* researchers Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever describes the ***`GPT-2`*** model - a ***unidirectional*** model trained on a dataset of *8 million web pages* with **1.5 billion parameters** (*OpenAI*, Radford et al., February 14 2019).

Comparison of the BERT vs. GPT-2 Model Architectures & Designs While both ***`BERT`*** and ***`GPT-2`*** share great similarities in their overall architecture & design, one of the biggest differences between these two ***PLMs*** lies in the *deeply bidirectional* nature of ***`BERT`*** & the *unidirectional* nature of ***`GPT-2`***. Specifically, BERT had been traditionally used preferably over GPT-2 in certain natural language processing (NLP) tasks, as it seemed to be designed with a more efficient model design & architecture (Kehlbeck, et al., July 31 2021).

Furthermore, the nature of embedding spaces in ***`BERT`*** and ***`GPT-2`*** differ greatly, as shown below of the first layers of each respective ***PLM***:



Most importantly, with regards to the strengths of our two ***PLMs***, ***`BERT`*** may be better for tasks surrounding contextual understanding of sentences, while ***`GPT-2`*** may be more geared towards tasks related to generating text & general language tasks (Fhal, January 17 2023).

Text Classification Model To build our text-classification model, we will **follow these steps**:

1. Implementing & setting up our two (2) **Pretrained Language Models** (***PLMs***), ***`BERT`*** and [***`GPT-2`***](C:).
2. Using the specific tokenizer used for each respective ***PLM*** to prepare the training & testing text data.
3. Training of the text-classification model using the specifically-tokenized training dataset for each ***PLM***, based on "sentiment_train.json."
4. Evaluation of our text-classification model using the specifically-tokenized testing dataset for each ***PLM***, based on "sentiment_test.json."

Results & Analysis A detailed analysis of the model's performance by comparing the results from the output of our two algorithms, where we will **include the following**:

1. *F1-score* or other relevant metrics.
2. Any challenges or limitations of the text-classification model/task.

Additionally, we will also try to provide a comparative analysis based on the results from our two previous assignments, our first assignment, *A1: Sentiment Analysis Text Classification* & from our second assignment, *A2: `Word2Vec` and `GloVe` Embeddings.*

Specifically, we recall from our previous two assignments, that we first implemented text-classification models based on two suitable algorithms (*A1*), and in our second, implemented the usage of `Word2Vec` & `GloVe` pretrained embeddings to assist in our text-classification tasks (*A1*).

Requirements

1.1.1 Libraries & Constants Initialization

[1]:
```
##### CS410: Natural Language Processing, Fall 2023 - 11/13/2023
##### A3: Pretrained Language Models (PLMs), Dan Jang - Initializations:␣
↪Libraries, Models, Data, & Constants

### 0.) Libraries
```

```
#from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from transformers import AutoTokenizer, AutoModel
import torch
import json
import pandas
#import huggingface_hub
import numpy as np
import random

## 1.0.) Constants, Variables, & Datasets

percentness = float(100)
# trainfile = str(trainfile)
# testfile = str(testfile)
traindata = []
testdata = []

# Loading the pretrained language models (PLMs), BERT & GPT-2 using HuggingFace␣
↪libraries!
## Bertie -> BERT PLM
## Bumblebee -> GPT(ransformer)-2 PLM
print("Downloading / Loading the BERT Pretrained Language Model (PLM) through␣
↪HuggingFace libraries...")
bertie = AutoModel.from_pretrained("bert-base-uncased")
print("Loading the BERT Specific Tokenizer...")
bertie_tokens = AutoTokenizer.from_pretrained("bert-base-uncased")
print("...the BERT (Base, pretrained, uncased tokenization) Pretrained Language␣
↪Model (PLM) has been downloaded / loaded!\n")

print("Downloading / Loading the GPT-2 Pretrained Language Model (PLM) through␣
↪HuggingFace libraries...")
bumblebee = AutoModel.from_pretrained("gpt2")
print("Loading the GPT-2 Specific Tokenizer...")
bumblebee_tokens = AutoTokenizer.from_pretrained("gpt2")
print("...the GPT-2 Pretrained Language Model (PLM) has been downloaded /␣
↪loaded!\n")
print("Checking for GPT-2 tokenizer padding token...")
if bumblebee_tokens.pad_token is None:
    print("GPT-2 tokenizer has no padding token!")
    print("...setting GPT-2 tokenizer padding token...")
    bumblebee_tokens.pad_token = bumblebee_tokens.eos_token
```

```
    print("...GPT-2 tokenizer padding token set!")
print("\n\nModel initialization all done!\n")
```

```
Downloading / Loading the BERT Pretrained Language Model (PLM) through
HuggingFace libraries…
Loading the BERT Specific Tokenizer…
…the BERT (Base, pretrained, uncased tokenization) Pretrained Language Model
(PLM) has been downloaded / loaded!

Downloading / Loading the GPT-2 Pretrained Language Model (PLM) through
HuggingFace libraries…
Loading the GPT-2 Specific Tokenizer…
…the GPT-2 Pretrained Language Model (PLM) has been downloaded / loaded!

Checking for GPT-2 tokenizer padding token…
GPT-2 tokenizer has no padding token!
…setting GPT-2 tokenizer padding token…
…GPT-2 tokenizer padding token set!


Model initialization all done!
```

1.1.2 Main Implementation: *Text Classification, with data-processed using respective tokenizers from & with Two (2) Pretrained Language Models*, `BERT` and `GPT-2`.

[2]:
```
##### CS410: Natural Language Processing, Fall 2023 - 11/13/2023
##### A3: Pretrained Language Models (PLMs), Dan Jang - Main Implementation
#### Objective: Exploring Natural Language Processing (NLP), by building a␣
↪text-classifier
#### for a text classification task, predicting whether a piece of text is␣
↪"positive" or "negative."
#### ...focusing on two (2) pretrained language models (PLMs),
#### ...BERT (Bidirectional Encoder Representations from Transformers) &␣
↪OpenAI's GPT-2 (Generative Pretrained Transformer),
#### ...and using the respective toenizers to each PLM to perform the␣
↪text-classification task as aforementioned

### 1.1.a) Logistic Regression algorithm using sklearn.linear_model.
↪LogisticRegression
### https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.
↪LogisticRegression.html
### Returns four (4) thingys:
# I.) accuracy_score,
# II.) f1_score,
# III.) confusion_matrix,
# & IV.) classification_report.
```

```
def plm(model, xtrain, ytrain, xtest, ytest):

    lreg = LogisticRegression()

    lreg.fit(xtrain, ytrain)
    predictionresults = lreg.predict(xtest)

    return accuracy_score(ytest, predictionresults), f1_score(ytest,␣
 ↪predictionresults), confusion_matrix(ytest, predictionresults),␣
 ↪classification_report(ytest, predictionresults)

### A3-Specific: Implementation functions for implementing the two pretrained␣
 ↪language models, BERT & GPT-2.
## Pretrained Language Model (PLM) Tokenizer Implementation Function
def plmodel(words, model, tokenizer):
    if tokenizer.pad_token is None:
        raise ValueError("[Debug A3.1 - PLModel()]: Tokenizer has no padding␣
 ↪token for the current model!")
    wordlist = tokenizer(words, return_tensors="pt", truncation=True,␣
 ↪padding=True)

    with torch.no_grad():
        results = model(**wordlist)

    return results.last_hidden_state.mean(dim=1).squeeze().numpy()
    # wordlist = words.split()
    # vecs = [model[w] for w in wordlist if w in model]
    # if vecs:
    #     return sum(vecs) / len(vecs)
    # else:
    #     return [0] * model.vector_size

# ## GPT(ransformer)-2 Model Implementation Function
# def bumblebee(words, model):
#     beds = [avgwordvec(w, model) for w in words]
#     return beds

def main(): #trainfile, testfile):
    print("Welcome, this is the main program for A3: Pretrained Language Models.
 ↪")
    print("Written by Dan J. for CS410: Natural Language Processing, Fall 2023.
 ↪")
    print("\nWe will use two (2) pretrained language models (PLM), BERT & GPT-2.
 ↪\n...to create a text-classifier to guess negative or positive␣
 ↪sentimentiality based on various text-reviews of products.")
```

```python
    # 1.0.I.A) Debug Statements #1a for dataset loading times:
    print("\nLoading the training & testing datasets...")
    # with open(trainfile, "r") as trainfile:
    with open("sentiment_train.json", "r") as trainfile:
        #traindata = json.load(trainfile)
        for row in trainfile:
            traindata.append(json.loads(row))

    trainframe = pandas.DataFrame(traindata)

    # with open(testfile, "r") as testfile:
    with open("sentiment_test.json", "r") as testfile:
        #testdata = json.load(testfile)
        for row in testfile:
            testdata.append(json.loads(row))

    testframe = pandas.DataFrame(testdata)

    # 1.0.I.B) Debug Statements #1b for dataset loading times:
    print("Successfully loaded the training & testing datasets!\n")

    ## 1.0.1.) Initial Preprocessing of the training & testing data
    ## First, we isolate our two (2) columns, "review_title" & "stars."
    ## Second, we will convert values in the "stars" column so that 1␣
↪[negative] = 0 & 5 [positive] = 1.
    ## This will allow us to make the negative or positive sentiment a binary␣
↪value-based thingy.
    trainframe = trainframe[['review_title', 'stars']]
    trainframe['stars'] = trainframe['stars'].apply(lambda x: 1 if x == 5 else␣
↪0)

    testframe = testframe[['review_title', 'stars']]
    testframe['stars'] = testframe['stars'].apply(lambda x: 1 if x == 5 else 0)

    ## A3-Specific: From our Slack channel (#nlp_f23), using tip to only use␣
↪25% of training dataset, evenly split
    ## Credits to Classmate Will McIntosh from the Slack thread started by␣
↪classmate Saurav Kumar Singh
    ## & also, full credits to classmate Will McIntosh for the following code␣
↪for GPU usage:

    #### Credits to Will McIntosh (11/11/2023):
    # Testing
    print(f"Is a GPU available: {torch.cuda.is_available()}")
    #print(f"Is this instance using a GPU?: {next(model.parameters()).is_cuda}")
    #### From Slack, #nlp_f23.
```

```
trainframe = trainframe.sample(frac=1, random_state=69)
trainframe = trainframe.iloc[:int(0.25 * len(trainframe))]

# y2train = trainframe['stars']
# print("[A3 Debug Size-Print #3] y2train", len(y2train))

ytest = testframe['stars']
# print("[A3 Debug Size-Print #4] ytest", len(ytest))

## Evenly split frames
x3train1, x3train2 = train_test_split(trainframe, test_size=0.5,␣
↪random_state=69)

y3train1 = x3train1['stars']
y3train2 = x3train2['stars']

#print("[A3 Debug Size-Print #1] x3train1 & x3train2", len(x3train1),␣
↪len(x3train2))
## A3-Specific: Applying BERT & GPT-2 PLM Specific Tokenization
print("Tokenizing the training & testing datasets for BERT...")
x2train1 = x3train1['review_title'].apply(lambda x: plmodel(x, bertie,␣
↪bertie_tokens))
#x2train1 = trainframe['review_title'].apply(lambda x: plmodel(x, bertie,␣
↪bertie_tokens))
x2test1 = testframe['review_title'].apply(lambda x: plmodel(x, bertie,␣
↪bertie_tokens))
print("BERT Tokenization has been applied to its training & testing␣
↪datasets!")

#print("[A3 Debug Size-Print #2a] x2train1 & x2test1", len(x2train1),␣
↪len(x2test1))

print("Tokenizing the training & testing datasets for GPT-2...")
x2train2 = x3train2['review_title'].apply(lambda x: plmodel(x, bumblebee,␣
↪bumblebee_tokens))
#x2train2 = trainframe['review_title'].apply(lambda x: plmodel(x,␣
↪bumblebee, bumblebee_tokens))
x2test2 = testframe['review_title'].apply(lambda x: plmodel(x, bumblebee,␣
↪bumblebee_tokens))
print("GPT-2's tokenization has been applied to its training & testing␣
↪datasets!")

#print("[A3 Debug Size-Print #2b] x2train2 & x2test2", len(x2train2),␣
↪len(x2test2))
```

```python
    ### 1.0.2b) Run Text-Classification Algorithms & Print the Model Results
    print("-----\n")
    print("Running text-classification model le training & testing datasets␣
↪(with pretrained language models, BERT & GPT-2)...")


    print("Running Logistic Regression algorithm, version A.) BERT...")
    bed1accuracy, bed1f1, bed1cmatrix, bed1creport = plm(bertie, x2train1.
↪tolist(), y3train1, x2test1.tolist(), ytest)
    print("..The data processing from our first PLM, BERT, is done!")

    print("Running Logistic Regression algorithm, version B.) GPT-2...")
    bed2accuracy, bed2f1, bed2cmatrix, bed2creport = plm(bumblebee, x2train2.
↪tolist(), y3train2, x2test2.tolist(), ytest)
    print("..The data processing from our second PLM, GPT-2, is done!")

    print("...All Done!")
    print("-----\n")

    print("Here are le results [Logistic Regression, with comparative results␣
↪between two (2) PLMs, BERT & GPT-2]...\n")
    print("Logistic Regression Algorithm, Version A: BERT Pretrained Language␣
↪Model-based Text-Classification Performance, Metrics, & Results:")
    print("...Accuracy was found to be, ", bed1accuracy * percentness, "%,")
    print("...F1 Score was found to be: ", bed1f1, ",")
    print("...with a Confusion Matrix: \n", bed1cmatrix, ",")
    print("...& lastly, the classification Report: \n", bed1creport)
    print("-----\n")

    print("Logistic Regression Algorithm, Version B: GPT-2 Pretrained Language␣
↪Model-based Text-Classification Performance, Metrics, & Results:")
    print("...Accuracy was found to be, ", bed2accuracy * percentness, "%,")
    print("...F1 Score was found to be: ", bed2f1, ",")
    print("...with a Confusion Matrix: \n", bed2cmatrix, ",")
    print("...& lastly, the classification Report: \n", bed2creport)
    print("-----\n")

if __name__ == "__main__":
    main()
```

```
Welcome, this is the main program for A3: Pretrained Language Models.
Written by Dan J. for CS410: Natural Language Processing, Fall 2023.

We will use two (2) pretrained language models (PLM), BERT & GPT-2.
…to create a text-classifier to guess negative or positive sentimentiality
based on various text-reviews of products.
```

```
Loading the training & testing datasets…
Successfully loaded the training & testing datasets!

Is a GPU available: False
Tokenizing the training & testing datasets for BERT…
BERT Tokenization has been applied to its training & testing datasets!
Tokenizing the training & testing datasets for GPT-2…
GPT-2's tokenization has been applied to its training & testing datasets!
-----

Running text-classification model le training & testing datasets (with
pretrained language models, BERT & GPT-2)…
Running Logistic Regression algorithm, version A.) BERT…

c:\tools\miniconda3\lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

..The data processing from our first PLM, BERT, is done!
Running Logistic Regression algorithm, version B.) GPT-2…
..The data processing from our second PLM, GPT-2, is done!
…All Done!
-----

Here are le results [Logistic Regression, with comparative results between two
(2) PLMs, BERT & GPT-2]…

Logistic Regression Algorithm, Version A: BERT Pretrained Language Model-based
Text-Classification Performance, Metrics, & Results:
…Accuracy was found to be,  91.55 %,
…F1 Score was found to be:  0.9158785465405676 ,
…with a Confusion Matrix:
 [[911  89]
 [ 80 920]] ,
…& lastly, the classification Report:
              precision    recall  f1-score   support

           0       0.92      0.91      0.92      1000
           1       0.91      0.92      0.92      1000

    accuracy                           0.92      2000
   macro avg       0.92      0.92      0.92      2000
```

```
weighted avg       0.92      0.92      0.92      2000

-----

Logistic Regression Algorithm, Version B: GPT-2 Pretrained Language Model-based
Text-Classification Performance, Metrics, & Results:
…Accuracy was found to be,  87.4 %,
…F1 Score was found to be:  0.872598584428716 ,
…with a Confusion Matrix:
 [[885 115]
 [137 863]] ,
…& lastly, the classification Report:
               precision    recall  f1-score   support

           0       0.87      0.89      0.88      1000
           1       0.88      0.86      0.87      1000

    accuracy                           0.87      2000
   macro avg       0.87      0.87      0.87      2000
weighted avg       0.87      0.87      0.87      2000

-----

c:\tools\miniconda3\lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

1.1.3 *A3: Pretrained Language Models (PLMs)*, Text-Classification Model Performance Analysis & Discussion

Initial Data Results, Metrics, & Analysis Using the Logistic Regression Algorithm like previously for *A2*, this algorithm was used to implement our text-classification model, for the text-classification task in both *Version A* & *B*.

As shown below, *Version A* featured *Google AI*'s ***BERT*** model & *Version B* featured *OpenAI*'s ***GPT-2*** model - where we saw the following results:

Version A - *BERT* Pretrained Language Model (PLM) Results:

```
Accuracy:  ~91.6%
F1 Score:  0.9158785465405676
```

```
Confusion Matrix:
[[911  89]
 [ 80 920]]

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.91      0.92      1000
           1       0.91      0.92      0.92      1000

    accuracy                           0.92      2000
   macro avg       0.92      0.92      0.92      2000
weighted avg       0.92      0.92      0.92      2000
```

Version B - *GPT-2* Pretrained Language Model (PLM) Results:

```
Accuracy:  87.4%
F1 Score:  0.872598584428716

Confusion Matrix:
[[885 115]
 [137 863]]

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.89      0.88      1000
           1       0.88      0.86      0.87      1000

    accuracy                           0.87      2000
   macro avg       0.87      0.87      0.87      2000
weighted avg       0.87      0.87      0.87      2000
```

Pretrained Language Model (PLM) Comparative Analysis & Discussion From the results above, we see that both *Google AI*'s ***BERT*** & *OpenAI*'s ***GPT-2*** ***PLMs*** faired very well in accuracy, of ~91.6% & 87.4% respectively.

Between these two ***PLMs***, ***BERT*** & ***GPT-2***, we can see that from our performance results that there is/was a slight, low *difference of accuracy*, where ***BERT*** has a ***higher*** **accuracy** of **~4.2%** in *comparison to* GPT-2.

Also, by looking at the respective Confusion Matrices for our two ***PLMs***, we can see that ***BERT***, of course, was able to classify sentiment more accurately than ***GPT-2***, but we also can observe that ***BERT*** seems to be *a little bit* more efficient in detecting *negative* sentimentality vs. ***GPT-2*** lean towards *positive* sentimentality - however, ***GPT-2***'s decreased accuracy in detecting *negative* sentiment seems to be more significant in magnitude compared to the aforementioned, *negative* over *positive* sentimentality bias, of ***BERT***.

Interestingly, a possible scenario where this supposed sentimentality bias in our text-classification task may present itself as to yield a different result, at least for accuracy & the F1 score, may lie in how the positive & negative sentiment composition during the processing of our training & testing dataset, particularly, at the time of randomized splitting. Specifically, if we had somehow chosen a random seed in our splitting code, that supposedly had much more positive or negative sentimentality in the composition of the reviews, we could have seen variations from our initial performance & results as described above.

Ergo, between *Google AI*'s ***`BERT`*** & *OpenAI*'s ***`GPT-2`*** ***PLMs***, ***`BERT`*** seems to be a clearly more suitable model, at least for the text-classification task & model with our assignment's training & dataset, "*Multilingual Amazon Reviews Corpus*."

Analysis of Current & Previous Text-Classification Models & Performance Results from *A1* & *A2* In comparison to the performance seen from *A2: `Word2Vec` & `GloVe` Embeddings*, both *Google AI*'s ***`BERT`*** & *OpenAI*'s ***`GPT-2`*** ***PLMs*** outperformed the accuracies & F1 scores from either `Word2Vec` or `GloVe` pretrained embeddings - where we recall the *A2* results as follows:

A2: Version A, `Word2Vec` Model:

```
Accuracy:  ~86.3%,
F1 Score:  0.86105476673428

Confusion Matrix:
[[1754  246]
 [ 302 1698]]

Classification Report:
            precision    recall  f1-score   support

        0       0.85      0.88      0.86      2000
        1       0.87      0.85      0.86      2000

accuracy                            0.86      4000
macro avg       0.86      0.86      0.86      4000
weighted avg    0.86      0.86      0.86      4000
```

A2: Version B, `GloVe` Model:

```
Accuracy: ~69.69%,
F1 Score: 0.7313829787234042

Confusion Matrix:
[[1138  862]
 [ 350 1650]]

Classification Report:
            precision    recall  f1-score   support

         0       0.76      0.57      0.65      2000
         1       0.66      0.82      0.73      2000
```

```
accuracy                                0.70      4000
macro avg         0.71       0.70       0.69      4000
weighted avg      0.71       0.70       0.69      4000
```

Although, we do see only a *slight* ~**1.1% accuracy** improvement when we compare results from `Word2Vec` vs. ***`GPT-2`***.

However, this is a neat & expected observation, as we can recall that both `Word2Vec` & ***`GPT-2`*** are suitable or otherwise geared towards the generation of text, rather than the specificity of text-classification required for efficiency & performance with our Amazon review sentimentality prediction task. As expected, contrarywise, as the results from the implementation of `GloVe` pre-trained embeddings & the ***`BERT`*** ***PLM*** show higher performance & accuracy compared to the `Word2Vec` pretrained embeddings & ***`GPT-2`*** ***PLM***, we could also make the observation of this supposed higher efficiency & efficacy by recalling that the `GloVe` pretrained embeddings were also designed & trained to create an unsupervised learning algorithm, with training that "*performed on aggregated global word-word co-occurence statistics*" (Pennington, et al., 2014), which, like the nature of ***`BERT`***'s greater suitability towards text-classification tasks, could explain these higher accuracies & better performance as observed from our previous *A2* & most current, *A3* results.

Recalling further, note that from our first assignment, *A1: Sentiment Analysis Text Classification*, we saw the following results from using *two (2) different algorithms* to implement the text-classification model, specifically, the *Logistic Regression* & the *Gaussian Näive Bayes* Algorithms:

From A1 Results: Version A: *Gaussian Näive Bayes* Algorithm:

```
Accuracy: ~59.2%
F1 Score: 0.3664596273291925

CConfusion Matrix:
[[948  52]
[764 236]]

Classification Report:
            precision    recall  f1-score   support

        0       0.55      0.95      0.70      1000
        1       0.82      0.24      0.37      1000

accuracy                            0.59      2000
macro avg       0.69      0.59      0.53      2000
weighted avg    0.69      0.59      0.53      2000
```

From A1 Results: Version B: *Logistic Regression* Algorithm:

```
Accuracy:  92.7%
F1 Score:  0.9272908366533865

Confusion Matrix:
[[923  77]
[ 69 931]]
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.92      0.93      1000
           1       0.92      0.93      0.93      1000

    accuracy                           0.93      2000
   macro avg       0.93      0.93      0.93      2000
weighted avg       0.93      0.93      0.93      2000
```

This is interesting, as while it appears that using ***PLMs*** over ***pretrained embeddings*** yielded much significant improvements in both accuracy & performance, the results from using only the Logistic Regression Algorithm with no classifiers had yielded a *92.7% accuracy*, which still stands as one of the most accurate results from our three (3) assignments.

However, this observation & other possible variations that can be observed, in performance & results, may be attributed to both the nature of text-tokenization, usage of classifiers, & particularly, that in the implementation of the first text-classification model of our *A1* assignment, I had trained the text-classification model cases with the ***full, eighty-thousand (80,000) rows of training data***, while in our current *A3* implementation, the training data for both the *`BERT`* & *`GPT-2`* ***PLMs*** were truncated randomly (with a shared random seed of `69`) to a quarter-percent (25%) of the original number of rows, or twenty-thousand (20,000) rows of training data.

Text-Classification Challenges & Limitations The initial & biggest challenge to implement these **pretrained language models** (***PLMs***) was the very significantly resource & time-intensive requirements for the computation, tokenization, & training of the two (2) ***PLMs*** used.

With the current version of code & implementation, in average, it appeared that *`BERT`* took around *two-to-eight (2-to-8) minutes*, while *`GPT-2`* took longer, around *five-to-fifteen (5-to-15) minutes* long. Both, on average, usually took around *fifteen-to-thirty (15-to-30) minutes* to fully complete.

However, in the earlier stages of coding & implementation, particularly before truncating the training & testing datasets to the currently-set percentage of twenty-five percent (25%) of the original amount of rows/lines of text, the average varied wildly, with completion durations taking *hours, upon hours* to complete or were stopped prior to completion due to time constraints.

Furthermore, even post-truncation adjustments, the pretraining of the two ***PLMs***, in terms of computational power & resources, were very intensive, which had caused ***multiple***, **full computer crashes**, leading to a lot of processing delays & mental energy to continue implementing.

Discussion for Future Performance & Efficacy Improvements With the issue of computational resources & the prevention of computer crashes, would be, of course, to finally sign-up for the student discounted *Google Cloud* subscription for using high-resource, cloud computing in *Google Colaboratory*, where NLP compatible, GPU-acceleration is available to expedite ***PLM***, model training, or otherwise for running intensive NLP code & tasks. As such, this specific idea for future performance & efficacy improvement will be implemented immediately following this current assignment, *A3*, where I will go ahead & attempt to set-up *Google Colaboratory* for use to hopefully, avoid the aforementioned resource & crash pitfalls for *A4* and our *Group Project*, heh.

Furthermore, I should attempt to start early & try to implement for assignments in smaller code-blocks/pieces.

1.1.4 References & Resources

Libraries & Dependencies

`numpy`
`pandas`
`torch`
`random`

HuggingFace_hub

Google AI's *BERT*

OpenAI's *GPT-2*

HuggingFace's *transformers*

`from transformers import AutoModel`(s)

`from transformers import AutoTokenizer`

`from transformers import AutoTokenizer.from_pretrained`, `AutoModel.from_pretrained`

sklearn.linear_model.LogisticRegression

sklearn.model_selection.train_test_split

sklearn.metrics.f1_score

sklearn.metrics.accuracy_score

sklearn.metrics.confusion_matrix

sklearn.metrics.classification_report

nbconvert

References & Credits *The Multilingual Amazon Reviews Corpus* by Phillip Keung, Yichao Lu, György Szarvas, & Noah A. Smith (October 6th, 2020)

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding by Jacob Devlin, Ming-Wei Chang, Kenton Lee, & Kristina Toutanova (May 24th, 2019)

Language Models are Unsupervised Multitask Learners by Alec Radford, Jeffrey Wu, Rewon Child David Luan, Dario Amodei, & Ilya Sutskever (*OpenAI*; February 14th, 2019)

Comparing BERT, GPT-2, and GPT-3: A Look at the Pros and Cons of Popular Pre-Trained Language Models by Em Fhal (January 17th, 2023)

GloVe: Global Vectors for Word Representation by Jeffrey Pennington, Richard Socher, & Christopher D. Manning (2014)

Efficient Estimation of Word Representations in Vector Space [*Word2Vec* Pre-trained Embeddings] by Tomas Mikolov, Kai Chen, Greg Corrado, & Jeffrey Dean (January 16th, 2013)

Credits to GitHub Copilot & ChatGPT for code implementation assistance.

Special Thanks Thanks to fellow classmate *Will McIntosh* for their helpful tips & tricks for resource management particularly for ***PLMs*** in the current *A3* assignment, from the `#nlp_f23` class-channel on *pdx-cs* Slack, November 11th, 2023.

Extra Stuff

1.1.5 A3 Results: Raw Output from ***BERT*** & ***OpenAI***'s ***GPT-2*** Pretrained Language Models (PLMs), using the Logistic Regression Algorithm

```
Here are le results [Logistic Regression, with comparative results between two (2) PLMs, BERT &

Logistic Regression Algorithm, Version A: BERT Pretrained Language Model-based Text-Classificat
...Accuracy was found to be,  91.55 %,
...F1 Score was found to be:  0.9158785465405676 ,
...with a Confusion Matrix:
[[911  89]
[ 80 920]] ,
...& lastly, the classification Report:
            precision    recall  f1-score   support

        0       0.92      0.91      0.92      1000
        1       0.91      0.92      0.92      1000

    accuracy                           0.92      2000
macro avg       0.92      0.92      0.92      2000
weighted avg       0.92      0.92      0.92      2000

-----

Logistic Regression Algorithm, Version B: GPT-2 Pretrained Language Model-based Text-Classifica
...Accuracy was found to be,  87.4 %,
...F1 Score was found to be:  0.872598584428716 ,
...with a Confusion Matrix:
[[885 115]
[137 863]] ,
...& lastly, the classification Report:
            precision    recall  f1-score   support

        0       0.87      0.89      0.88      1000
        1       0.88      0.86      0.87      1000

    accuracy                           0.87      2000
macro avg       0.87      0.87      0.87      2000
weighted avg       0.87      0.87      0.87      2000
```

1.1.6 A2 Results: Raw Output from Word2Vec & GloVe Embedding Results, using the Logistic Regression Algorithm

```
Logistic Regression Algorithm, Version A: Word2Vec Pretrained Model-based Embeddings Performanc
...Accuracy was found to be,  86.3 %,
...F1 Score was found to be:  0.86105476673428 ,
...with a Confusion Matrix:
[[1754  246]
[ 302 1698]] ,
...& lastly, the classification Report:
            precision    recall  f1-score   support

        0       0.85      0.88      0.86      2000
        1       0.87      0.85      0.86      2000

    accuracy                           0.86      4000
macro avg       0.86      0.86      0.86      4000
weighted avg       0.86      0.86      0.86      4000

-----

Logistic Regression Algorithm, Version B: GloVe Pretrained Model-based Embeddings Performance,
...Accuracy was found to be,  69.69999999999999 %,
...F1 Score was found to be:  0.7313829787234042 ,
...with a Confusion Matrix:
[[1138  862]
[ 350 1650]] ,
...& lastly, the classification Report:
            precision    recall  f1-score   support

        0       0.76      0.57      0.65      2000
        1       0.66      0.82      0.73      2000

    accuracy                           0.70      4000
macro avg       0.71      0.70      0.69      4000
weighted avg       0.71      0.70      0.69      4000

-----
```

From A1 Results for Reference: **Initial Full 80k-Row Processing Results Raw Output**

```
Algorithm #1, Version A: Gaussian Näive Bayes Performance, Metrics, & Results:
...Accuracy was found to be,  59.199999999999996 %,
...F1 Score was found to be:  0.3664596273291925 ,
...with a Confusion Matrix:
[[948  52]
[764 236]] ,
...& lastly, the classification Report:
```

```
            precision    recall  f1-score   support

        0       0.55      0.95      0.70      1000
        1       0.82      0.24      0.37      1000

    accuracy                           0.59      2000
macro avg       0.69      0.59      0.53      2000
weighted avg       0.69      0.59      0.53      2000

-----

Algorithm #2, Version A: Logistic Regression Performance, Metrics, & Results:
...Accuracy was found to be,  92.7 %,
...F1 Score was found to be:  0.9272908366533865 ,
...with a Confusion Matrix:
[[923  77]
[ 69 931]] ,
...& lastly, the classification Report:
            precision    recall  f1-score   support

        0       0.93      0.92      0.93      1000
        1       0.92      0.93      0.93      1000

    accuracy                           0.93      2000
macro avg       0.93      0.93      0.93      2000
weighted avg       0.93      0.93      0.93      2000

-----

Algorithm #1, Version B: Gaussian Näive Bayes Performance, Metrics, & Results:
...Accuracy was found to be,  59.3 %,
...F1 Score was found to be:  0.36899224806201547 ,
...with a Confusion Matrix:
[[948  52]
[762 238]] ,
...& lastly, the classification Report:
            precision    recall  f1-score   support

        0       0.55      0.95      0.70      1000
        1       0.82      0.24      0.37      1000

    accuracy                           0.59      2000
macro avg       0.69      0.59      0.53      2000
weighted avg       0.69      0.59      0.53      2000

-----

Algorithm #2, Version B: Logistic Regression Performance, Metrics, & Results:
```

```
...Accuracy was found to be,  92.80000000000001 %,
...F1 Score was found to be:  0.9281437125748503 ,
...with a Confusion Matrix:
[[926  74]
[ 70 930]] ,
...& lastly, the classification Report:
            precision    recall  f1-score   support

        0       0.93      0.93      0.93      1000
        1       0.93      0.93      0.93      1000

    accuracy                           0.93      2000
macro avg       0.93      0.93      0.93      2000
weighted avg       0.93      0.93      0.93      2000

-----
```

[17]:

```
##### Juypter Notebook -> PDF Conversion thingy

#!pip install nbconvert

!jupyter nbconvert a3-Pretrained-Language-Models-dan-jang.ipynb --to pdf
```

```
[NbConvertApp] Converting notebook a3-Pretrained-Language-Models-dan-jang.ipynb
to pdf
[NbConvertApp] Writing 95660 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | b had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 3735153 bytes to a3-Pretrained-Language-Models-dan-
jang.pdf
```