a4-Multilingual-Large-Language-Models-dan-jang

November 30, 2023

1 CS410: Natural Language Processing, Fall 2023

1.1 A4: Multilingual Large Language Models (LLMs), Dan Jang - 11/27/2023

Description of Assignment

Introduction As the training & testing datasets from our first & second assignments has become defunct recently (*The Multilingual Amazon Reviews Corpus* by Phillip Keung, Yichao Lu, György Szarvas, & Noah A. Smith (October 6th, 2020) [1]), in this assignment, *A4: Multilingual Large Language Models (LLMs)*, we will be using a new dataset, the "*Unified Multilingual Sentiment Analysis Benchmark*" [2] - where we will be exploring & comparing the performance of two, specific ***Large Language Models*** (**LLMs**):

1. *Meta*'s **`LLaMA 2`** (*Large Language Model Meta AI 2*) [3], [4]

&

2. *OpenAI*'s **`ChatGPT`** (As of November 27th, 2023, **`ChatGPT`** is currently running the November 6th, 2023 Update model version of *GPT-3.5 Turbo* **`gpt-3.5-turbo-1106`**) [5]-[7]

Like the previous two assignment, this assignment focuses on using specific NLP models - specifically, *Meta*'s **`LLaMA 2`** [3], [4] and *OpenAI*'s **`ChatGPT`** ***Large Language Models*** (**LLMs**) [5], [6].

As a different approach in comparison to our previous three (3) assignments, in this current assignment, *A4*, instead of implementing a full text-classification model for sentiment-prediction, we will instead be exploring the techniques of *prompt engineering* [8] in creating a text-classifier - in comparison to previous assignment, *A3*, where we had used the monolingual *Pretrained Language Models* (***PLMs***) of **`BERT`** [9] and **`GPT-2`** [10].

Data Preparation In *A4*, we will be using a new dataset, *Unified Multilingual Sentiment Analysis Benchmark* [2], which is a multilingual dataset with Tweets labeled with a `sentiment` value, where the classification labels are as follows:

`0` indicating ***`negative`***,

`1` indicating ***`neutral`***,

...and a `2` indicating ***`positive`***.

The *Unified Multilingual Sentiment Analysis Benchmark* [2] dataset contains sentiment-labeled Tweets in Arabic, English, French, German, Hindi, Italian, Portuguese, and Spanish.

Instead of using several thousands of data-entries, for both a training and a testing dataset to train and test our text-classification model like we've explored in our previous assignments, of assignments *A1* through *A3* - in this assignment, *A4*, we will *instead* be utilizing a **much smaller subset of data** from our ***new dataset*** [2] & only utilizing a testing dataset based on a sampling of Tweet-entries across the different languages, in lieu of using both a testing and training dataset.

Specifically, we will be creating a multilingual, Tweet-entry-based testing dataset, of a recommended amount of ~50 instances (the actual number used in my implementation will be specified below in my ***Text Classification Through Prompt Engineering*** section) of the testing dataset for each language as aforementioned, and sentimentality-wise, we will only look at the Tweet-entries with either the `positive` or `negative` sentiment labels, and will be ignoring Tweet-entries with the `neutral` sentiment label.

Although no training will be required, we will still need to carefully prepare a balanced test set across both classes, and throughout our various languages sampled from the ***new dataset*** [2].

Prompt Engineering Researcher Golam Md. Muktadir of the University of California, Santa Cruz, Computer Science and Engineering (muktadir@ucsc.edu) describes in good detail, the past-to-present history of *prompt engineering* in their relatively recent September 30th, 2023 paper, "*A Brief History of Prompt: Leveraging Language Models*" [8].

Modern day *prompt engineering* can be described by the following description…

"…***researchers and practitioners [who] have explored various techniques to harness the full potential of language models, leveraging the power of prompts to guide, control, and tailor the output of these sophisticated AI systems.***…"

…where this ***very description*** of *prompt engineering* originates from a generated text borne of *prompt engineering* itself, as given by Researcher Muktadir in their paper's introduction, from these two prompts [8]:

"**Prompt #1**: *You are a scholar in machine learning and language models. I am writing a paper on the history of prompt engineering and generation. Can you give me a timeline for prompt engineering evolution? (We used this timeline to create prompts for each section later)*"

&

"**Prompt #2**: *Write the introduction of this paper. Emphasize that this paper focuses on how language prompts and queries have been used so far.*"

While the techniques of *prompt engineering*, seemingly only recently popularized, or rather, been made the spotlight mainstream subject of discussion of recent, in regard to the wide field of *Natural Language Processing* (***NLP***) - this popularization, of course, significantly due to the rise of *OpenAI*'s `ChatGPT` [6], [7] - however, nonetheless are a set of techniques a history long prior to this very novel advent of widely accessible platforms providing services based on *Natural Language Processing* (***NLP***) study & more specifically, through *Large Language Models* (***LLMs***) [8].

As section-titled in their paper, Researcher Muktadir describes an early era of *prompt engineering* in "*Prehistoric Prompting: Pre NN-Era*", which describes among the most preliminary studies into what we know as *prompt engineering* today - as far back as the 1960s through the 1970s with "*Early Natural Language Interfaces*", the 1990s through the 2010s with "*Advances in Natural Language Processing*", e.g. *Neural Networks* (***NNs***) and *Machine Learning* (***ML***), to modern-day, featuring

the rise of **Multilingual Large Language Models** (***MLLMs***) of those, we will exploring of two such ***MLLMs*** [8].

Multilingual Large Language Models (Multilingual LLMs, or 'MLLMs') One way we can/will be accessing our two **Multilingual Large Language Models** (***MLLMs***), *Meta*'s **`LLaMA 2`** [3], [4] and *OpenAI*'s **`ChatGPT`** **Multilingual Large Language Models** (***MLLMs***) [5], [6], would be through the graphical interface chat platform provided through *HuggingFace* Chat [11].

Meta's Open-Source LLaMA 2 (*Large Language Model Meta AI 2*) Multilingual Large Language Model (MLLM) [3], [4] The first ***MLLM*** we will be exploring will be the ***open-source***, **`LLaMA 2`** (*Large Language Model Meta AI 2*) [4] ***MLLM***, which was first co-released by *Meta* and *Microsoft*, back on June 17th, 2023 [14].

This ***MLLM*** was first released & described as ***`LLaMA (Version 1)`*** in the paper, "*LLaMA: Open and Efficient Foundation Language Models*" [13] - which was first published earlier this year in February 27th, 2023, by the *Meta AI* team [3] - authored by Researchers Hugo Touvron, and thirteen (13) other authors.

This first version of ***`LLaMA`*** was described as "*a collection of foundation language models ranging from 7B to 65B parameters*", these models having been trained on "*trillions of tokens*", and importantly, claiming that *it is* and *was possible* to "*train state-of-the-art models using publicly available datasets* **exclusively**" - claiming, that the-then ***`LLaMA-13B`*** model outperformed the ***`GPT-3 (175B)`*** model on "*most benchmarks*" [13].

The second iteration, ***`LLaMA 2`*** has been released as part of various ***fine-tuned LLMs***, e.g. ***`LLaMA 2-Chat`***, ***`Code LLaMA 2`***, etc. - as to represent the latest efforts by *Meta* and *Microsoft* to provide their updated advancements of their open-source ***LLM*** - where, in their **paper**, they claim that the latest ***`LLaMA 2 LLM(s)`*** are capable and *very* comparable in performance to those of other closed-source ***LLMs***, either in regard to a closed-source based set of techniques used to create the ***LLM(s)*** and/or a closed-source service that is provided as *SaaS* (***Software as a Service***), e.g. *OpenAI*'s **`ChatGPT`** [14], [4], [5], [6].

OpenAI's Proprietary ChatGPT (*Chat Generative Pretrained Transformer*) Multilingual Large Language Model (MLLM) [5]-[7] The most famous ***MLLM*** model as of recent mainstream attention, is the proprietary model **`ChatGPT`** (based on *Generative Pretrained Transformer 3.5 Turbo*, currently on the November 6th, 2023 update-iteration, **`gpt-3.5-turbo-1106`**) is based off advancements made by *OpenAI* on the closed-source 'release' of **`GPT-3`**, and of course, follows the progress made from the second predecessor, **`GPT-2`** [6], [7], [5], [15], [10].

Recalling from *A3*: 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 ***uni-directional*** model trained on a dataset of *8 million web pages* with **1.5 billion parameters** (*OpenAI*, Radford et al. [10].

However, if we were to make a quick GPT lineage comparison between ***`gpt-3.5-turbo-1106`*** vs. ***`GPT-2`***, of course, the former, ***`gpt-3.5-turbo-1106`***, has up to ***175 billion parameters*** & the latter, at least the largest-parameter version of ***`GPT-2`***, has still only ***1.5 billion parameters*** - representing an extremely significant increase in parameter-power in ***`gpt-3.5-turbo-1106`*** over its long predecessor ***`GPT-2`*** model.

Comparison of LLaMA-2 vs. ChatGPT Multilingual Large Language Models (*MLLMs*) Architectures & Designs When comparing `LLaMA 2` vs. `ChatGPT` (`gpt-3.5-turbo-1106`), the first difference we can note is that, since `LLaMA 1` was released & now with `LLaMA 2`, `LLaMA` has remained available as a series of ***open-source*** **Multilingual Large Language Models** (***MLLM***), as released & created by *Meta's AI Team* (& *Microsoft* for version 2) - where anyone is able to request & receive access to freely download the model & its fine-tuned derivatives as per the nature open-source vs. the proprietary nature of `ChatGPT`, which is only offered indirectly, as a **Software-as-a-Service** (***SaaS***) through the free version of *OpenAI*'s main *`ChatGPT` web-platform* (chat.openai.com) or as a paid service through *OpenAI*'s *API platform* [4], [6], [7], [3], [13], [14], [6], [5].

From September 1st, 2023, Enterprise AI Strategist Sunil Ramlochan of the *Prompt Engineering Institute* described in the summary of their article, *How Does Llama-2 Compare to GPT-4/3.5 and Other AI Language Models* that `LLaMA 2` "*competes on accuracy despite lower complexity* [vs. the complexity of GPT-3.5]" [16] - where such strong accuracy could be through *Meta*'s various improvements from `LLaMA 1` to `LLaMA 2`, e.g. *Ghost Attention*, which may improve "*dialogue context tracking*" [16], [4], [14].

As cited in [16], Researcher Waleed Kadous of *anyscale* performed an analysis of our two **Multilingual Large Language Models** (***MLLMs***), in their supportively titled (of the claims made in [16]) article, "*Llama 2 is about as factually accurate as GPT-4 for summaries and is 30X cheaper*" [17]:

Model	Input Tokens	Output Tokens	Price MTokens input ($)	Price MTokens output ($)	Total cost
gpt-4	125902	834	30	60	$3.83
gpt-3.5-turbo	125902	770	1.5	2	$0.19
Llama-2-7b	149238	80525	0.25	0.25	$0.06
Llama-2-13b	149238	104947	0.5	0.5	$0.13
Llama-2-70b	149238	61500	1	1	$0.21

Text Classification Through Prompt Engineering To perform our text-classification task on these, we will **follow these steps**:

1. Prepare a balanced test set of a specific, constant chosen-size of test Tweet-instances for each of the eight different languages from our new dataset, *Unified Multilingual Sentiment Analysis Benchmark* [2]. Specifically, we need to pick either 'positive' or 'negative' classes, ergo, ignoring Tweet-instances with the 'neutral' label. In my implementation, I will be using a test set of
2. Using Prompt Engineering, we will have each of the ***Multilingual LLMs*** attempt to predict the sentiment, neither 'positive' nor 'negative', of each of the Tweet-instances in our test set, for each language.

Results & Analysis We will, of course, be comparing the two ***Multilingual LLMs*** by the metrics such as the *F1-score* or other relevant metrics like from our previous assignments we've

explored this lovely Fall term.

Specific to this assignment, *A4*, we will be focusing on **answering the following questions** through our exploration of the two aforementioned ***MLLMs*** - and the results and analysis from our exploration:

1. "*How do the two LLMs perform? Which one is better? Any possible explanation?*"
2. "*Comparing the results across the six different languages, what do you observe? Any possible explanation?*"
3. "*What challenges did you face?*"

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
```

1.1.2 Formatted JSON Data Loading

[ ]:
```
arabic = [
    {
        'prompt': "\u0646\u0648\u0627\u0644␣
 ↪\u0627\u0644\u0632\u063a\u0628\u064a (\u0627\u0644\u0634\u0627\u0628␣
 ↪\u062e\u0627\u0644\u062f \u0644\u064a\u0633 \u0639\u0627\u0644\u0645\u064a)␣
 ↪\u0647\u0647\u0647\u0647\u0647\u0647\u0647␣
 ↪\u0623\u062a\u0641\u0631\u062c\u064a \u0639\u0644\u0649 \u0647\u0627␣
 ↪\u0627\u0644\u0641\u064a\u062f\u064a\u0648 \u064a\u0627␣
 ↪\u0645\u0628\u062a\u062f\u0626\u0629 http v\u00eda @user",
        'actual_rating': "0",
```

```
        'predicted_rating_chatgpt': 0,  # My predicted sentiment rating (0 =␣
↪negative)
        'predicted_rating_llama': None,
    },
    {
        'prompt': "\u0646\u0648\u0627\u0644␣
↪\u0627\u0644\u0632\u063a\u0628\u064a \u0644\u0637\u064a\u0641\u0647␣
↪\u0627\u0644\u0641\u0646\u0627\u0646\u0647␣
↪\u0627\u0644\u0648\u062d\u064a\u062f\u0647 \u0627\u0644\u0644\u064a␣
↪\u0643\u0644 \u0627\u0644\u0641\u064a\u062f\u064a\u0648␣
↪\u0643\u0644\u064a\u0628\u0627\u062a \u062a\u0628\u0639\u0647\u0627␣
↪\u0645\u0627\u062a\u0633\u0628\u0628 \u062a\u0644\u0648\u062b␣
↪\u0628\u0635\u0631\u064a \u0648\u0644\u0627 \u0633\u0645\u0639\u064a␣
↪\u0644\u0648 \u0635\u0648\u062a\u0647\u0627 \u0627\u0642\u0644 \u0645\u0646␣
↪\u0639\u0627\u062f\u064a",
        'actual_rating': "2",
        'predicted_rating_chatgpt': 2,  # My predicted sentiment rating (2 =␣
↪positive)
        'predicted_rating_llama': None,
    },
    {
        'prompt': "\u0644\u0645\u0627 \u0642\u0627\u0644\u062a␣
↪\u0646\u0648\u0627\u0644 \u0627\u0644\u0632\u063a\u0628\u064a␣
↪\u0644\u0627\u0628\u0642\u0644\u0647\u0627␣
↪\u0647\u0627\u0644\u0644\u0642\u0628 \u0641\u0631\u062d\u0648\u0627␣
↪\u0641\u0627\u0646\u0632\u0647\u0627␣
↪\ud83d\ude02\ud83d\ude02\ud83d\ude02\u0643\u0627\u0646␣
↪\u0644\u0627\u0632\u0645 \u064a\u0627\u062e\u062f\u0648\u0647\u0627\u0627␣
↪\u0627\u0647\u0627\u0646\u0629 \u0645\u0634 \u062b\u0646\u0627\u0621 http",
        'actual_rating': "0",
        'predicted_rating_chatgpt': 0,  # My predicted sentiment rating (0 =␣
↪negative)
        'predicted_rating_llama': None,
    },
    {
        'prompt': "@user \u062a\u0630\u0643\u0631\u0646\u064a␣
↪\u0628\u0623\u063a\u0646\u064a\u0629 \u0646\u0648\u0627\u0644␣
↪\u0627\u0644\u0632\u063a\u0628\u064a \"\u0639\u064a\u0646\u064a\u0643␣
↪\u0643\u062f\u0627\u0628\u064a\u0646\"",
        'actual_rating': "2",
        'predicted_rating_chatgpt': 2,  # My predicted sentiment rating (2 =␣
↪positive)
        'predicted_rating_llama': None,
    },
    {
```

```
        'prompt': "\u0628\u0644\u0627 \u062a\u0634\u0641\u064a\u0631-␣
↪\u0623\u0645\u0644 \u062d\u0645\u0627\u062f\u064a␣
↪\u0628\u062a\u0646\u062a\u0642\u062f \u0627\u0644\u0646\u062c\u0645\u0629␣
↪\u0646\u0648\u0627\u0644 \u0627\u0644\u0632\u063a\u0628\u064a:\"␣
↪\u0646\u062a\u064a... http",
        'actual_rating': "0",
        'predicted_rating_chatgpt': 0,  # My predicted sentiment rating (0 =␣
↪negative)
        'predicted_rating_llama': None,
    },
    {
        'prompt': "\u0641\u0646\u0627\u0646\u0629␣
↪\u0644\u0628\u0646\u0627\u0646\u064a\u0629 \u0643\u0628\u064a\u0631\u0629 ␣
↪\u0635\u0648\u062a\u0647\u0627 \u0625\u0633\u0637\u0648\u0631\u064a \u061f!␣
↪#\u0645\u0627\u062c\u062f\u0629_\u0627\u0644\u0631\u0648\u0645\u064a   @user␣
↪#\u0646\u062c\u0648\u0649_\u0643\u0631\u0645  @user␣
↪#\u0646\u0648\u0627\u0644_\u0627\u0644\u0632\u063a\u0628\u064a  @user␣
↪#\u062c\u0648\u0644\u064a\u0627_\u0628\u0637\u0631\u0633",
        'actual_rating': "2",
        'predicted_rating_chatgpt': 2,  # My predicted sentiment rating (2 =␣
↪positive)
        'predicted_rating_llama': None,
    },
    {
        'prompt': "#\u0644\u0628\u0646\u0627\u0646\u064a_\u064a\u
```

1.1.3 Main Implementation: Metrics & Performance

[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.")

  # 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))

    ### 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(
```

1.1.4 *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` pretrained 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.

1.1.5 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 [1] P. Keung, Y. Lu, G. Szarvas, and N. A. Smith, "The Multilingual Amazon Reviews Corpus." arXiv, Oct. 06, 2020. doi: 10.48550/arXiv.2010.02573.

[2] "cardiffnlp/tweet_sentiment_multilingual · Datasets at Hugging Face." [Online]. Available: https://huggingface.co/datasets/cardiffnlp/tweet_sentiment_multilingual

[3] "Introducing LLaMA: A foundational, 65-billion-parameter language model." [Online]. Available: https://ai.meta.com/blog/large-language-model-llama-meta-ai/

[4] "Llama 2: Open Foundation and Fine-Tuned Chat Models | Research - AI at Meta." [Online]. Available: https://ai.meta.com/research/publications/llama-2-open-foundation-and-fine-tuned-chat-models/

[5] "OpenAI Platform." [Online]. Available: https://platform.openai.com

[6] "Introducing ChatGPT." [Online]. Available: https://openai.com/blog/chatgpt

[7] "New models and developer products announced at DevDay." [Online]. Available: https://openai.com/blog/new-models-and-developer-products-announced-at-devday

[8] G. M. Muktadir, "A Brief History of Prompt: Leveraging Language Models. (Through Advanced Prompting)." arXiv, Nov. 28, 2023. doi: 10.48550/arXiv.2310.04438.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." arXiv, May 24, 2019. doi: 10.48550/arXiv.1810.04805.

[10] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners".

[11] "HuggingChat." [Online]. Available: https://huggingface.co/chat

[12] " Transformers." [Online]. Available: https://huggingface.co/docs/transformers/index

[13] H. Touvron et al., "LLaMA: Open and Efficient Foundation Language Models." arXiv, Feb. 27, 2023. doi: 10.48550/arXiv.2302.13971.

[14] "Meta and Microsoft Introduce the Next Generation of Llama." [Online]. Available: https://ai.meta.com/blog/llama-2/

[15] T. B. Brown et al., "Language Models are Few-Shot Learners." arXiv, Jul. 22, 2020. doi: 10.48550/arXiv.2005.14165.

[16] "How Does Llama-2 Compare to GPT-4/3.5 and Other AI Language Models," Prompt Engineering Institute. [Online]. Available: https://promptengineering.org/how-does-llama-2-compare-to-gpt-and-other-ai-language-models/

[17] "Llama 2 vs. GPT-4: Nearly As Accurate and 30X Cheaper," Anyscale. [Online]. Available: https://www.anyscale.com/blog/llama-2-is-about-as-factually-accurate-as-gpt-4-for-summaries-and-is-30x-cheaper?ref=promptengineering.org

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.

[2]:
```
##### Juypter Notebook -> PDF Conversion thingy

#!pip install nbconvert

!jupyter nbconvert a4-Multilingual-Large-Language-Models-dan-jang --to pdf
```

```
[NbConvertApp] Converting notebook a4-Multilingual-Large-Language-Models-dan-
jang.ipynb to pdf
[NbConvertApp] Writing 106548 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] CRITICAL | x failed: xelatex notebook.tex -quiet
```

```
dvipdfmx:fatal: Image inclusion failed. Could not find file:
GPT4-3.5-vs.-LLaMA-2-{[}16-17{]}.png



No output PDF file written.



Sorry, but xelatex did not succeed.



The log file hopefully contains the information to get MiKTeX going again:



  C:\Users\Dan\AppData\Local\MiKTeX\miktex\log\xelatex.log


Traceback (most recent call last):
  File "C:\tools\miniconda3\lib\runpy.py", line 196, in _run_module_as_main
    return _run_code(code, main_globals, None,
  File "C:\tools\miniconda3\lib\runpy.py", line 86, in _run_code
    exec(code, run_globals)
  File "c:\tools\miniconda3\Scripts\jupyter-nbconvert.EXE\__main__.py", line 7,
in <module>
  File "C:\tools\miniconda3\lib\site-packages\jupyter_core\application.py", line
277, in launch_instance
    return super().launch_instance(argv=argv, **kwargs)
  File "C:\tools\miniconda3\lib\site-packages\traitlets\config\application.py",
line 1053, in launch_instance
    app.start()
  File "C:\tools\miniconda3\lib\site-packages\nbconvert\nbconvertapp.py", line
369, in start
    self.convert_notebooks()
  File "C:\tools\miniconda3\lib\site-packages\nbconvert\nbconvertapp.py", line
541, in convert_notebooks
    self.convert_single_notebook(notebook_filename)
  File "C:\tools\miniconda3\lib\site-packages\nbconvert\nbconvertapp.py", line
506, in convert_single_notebook
    output, resources = self.export_single_notebook(notebook_filename,
resources, input_buffer=input_buffer)
  File "C:\tools\miniconda3\lib\site-packages\nbconvert\nbconvertapp.py", line
435, in export_single_notebook
    output, resources = self.exporter.from_filename(notebook_filename,
resources=resources)
```

```
  File "C:\tools\miniconda3\lib\site-packages\nbconvert\exporters\exporter.py",
line 190, in from_filename
    return self.from_file(f, resources=resources, **kw)
  File "C:\tools\miniconda3\lib\site-packages\nbconvert\exporters\exporter.py",
line 208, in from_file
    return self.from_notebook_node(nbformat.read(file_stream, as_version=4),
resources=resources, **kw)
  File "C:\tools\miniconda3\lib\site-packages\nbconvert\exporters\pdf.py", line
183, in from_notebook_node
    self.run_latex(tex_file)
  File "C:\tools\miniconda3\lib\site-packages\nbconvert\exporters\pdf.py", line
153, in run_latex
    return self.run_command(self.latex_command, filename,
  File "C:\tools\miniconda3\lib\site-packages\nbconvert\exporters\pdf.py", line
141, in run_command
    raise raise_on_failure(
nbconvert.exporters.pdf.LatexFailed: PDF creating failed, captured latex output:
Failed to run "xelatex notebook.tex -quiet" command:


dvipdfmx:fatal: Image inclusion failed. Could not find file:
GPT4-3.5-vs.-LLaMA-2-{[}16-17{]}.png



No output PDF file written.



Sorry, but xelatex did not succeed.



The log file hopefully contains the information to get MiKTeX going again:



  C:\Users\Dan\AppData\Local\MiKTeX\miktex\log\xelatex.log
```