Problem 1

```
!pip install -q transformers
```

```
from transformers import GPT2LMHeadModel, GPT2TokenizerFast
import torch
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remoun
```

```
test = pd.read_csv('/content/drive/MyDrive/CS6983_GenAI_Folder/Homework_1/Problem1_dataset.csv')
```

```
print('First 5 test samples:')

for i, example in enumerate(test.text.values[:5]):
  print(f'{i+1}. {example}')
```

```
 a starring role in the play Herons written by Simon Stephens , which was performed in 2001 at the Royal Court Theatre .

ollowed by a role in the 2007 theatre production of How to Curse directed by Josie Rourke . How to Curse was performed a
```

```
#Wiki text samples converted to a list of strings:

wiki_list_of_strings = []
for i, example in enumerate(test.text.values):
  #print(f'{i+1}. {example}')
  wiki_list_of_strings.append(example)

cleaned_wiki_list_of_strings = [string.replace('\n', '') for string in wiki_list_of_strings] #List[str]
#print(len(cleaned_wiki_list_of_strings))

```

```
 'Theatre in the London Borough of Hammersmith and Fulham . In a '
 'review of the production for The Daily Telegraph , theatre critic '
 'Charles Spencer noted , " Robert Boulter brings a touching '
 'vulnerability to the stage as William . " ',
 ' Boulter starred in two films in 2008 , Daylight Robbery by '
 'filmmaker Paris Leonti , and Donkey Punch directed by Olly '
 'Blackburn . Boulter portrayed a character named " Sean " in Donkey '
 'Punch , who tags along with character " Josh " as the " quiet '
 'brother ... who hits it off with Tammi " . Boulter guest starred '
 'on a two @-@ part episode arc " Wounds " in May 2008 of the '
 'television series Waking the Dead as character " Jimmy Dearden " . '
 'He appeared on the television series Survivors as " Neil " in '
 'November 2008 . He had a recurring role in ten episodes of the '
 'television series Casualty in 2010 , as " Kieron Fletcher " . He '
 'portrayed an emergency physician applying for a medical fellowship '
 '. He commented on the inherent difficulties in portraying a '
 'physician on television : " Playing a doctor is a strange '
 "experience . Pretending you know what you 're talking about when "
 "you don 't is very bizarre but there are advisers on set who are "
 'fantastic at taking you through procedures and giving you the '
 "confidence to stand there and look like you know what you 're "
 'doing . " Boulter starred in the 2011 film Mercenaries directed by '
 'Paris Leonti . ']
```

```python
# Print dataset statistics:

# The TOTAL number of samples in dataset:
count = len(test.text.values)
print(f'Total number of samples: {count}')

# The AVERAGE number of characters per sample:
num_char, count = 0, 0
for example in (test.text.values):
    num_char += len(example)
    count += 1
print(f'The average number of characters per sample {(num_char/count)}')

```

```
Total number of samples: 187
The average number of characters per sample 483.80213903743316
```

```python
# Complete function to Calculate Perplexity

def calculate_perplexity(text, model, tokenizer, window_length=1024, step_size=256):
  model_inputs = tokenizer.encode(text, return_tensors='pt')
  perplexities = []
  DIMENSION = 1
  for start in range(0, model_inputs.size(DIMENSION), step_size):
        end = min(start + window_length, model_inputs.size(DIMENSION))
        window = model_inputs[:, start:end]

        with torch.no_grad():
            outputs = model(window, labels=window)

        neg_log_likelihood = outputs.loss

        perplexity = torch.exp(neg_log_likelihood)
        perplexities.append(perplexity.item())

        average_perplexity = sum(perplexities) / len(perplexities) if perplexities else float('inf')
  return average_perplexity

```

```python
# Complete code for HuggingFace GPT2 model (small, medium, and large) evaluation, according to the perplexity metric

# HuggingFace GPT2 model - Small
small_tokenizer = GPT2TokenizerFast.from_pretrained('gpt2')
small_model = GPT2LMHeadModel.from_pretrained('gpt2')

perplexity_dicitonary_gpt2 = dict()
for i in range(len(cleaned_wiki_list_of_strings)):
  perplexity_dicitonary_gpt2[i]=calculate_perplexity(cleaned_wiki_list_of_strings[i],small_model,small_tokenizer)
print(perplexity_dicitonary_gpt2)


# # HuggingFace GPT2 model - Medium

medium_tokenizer = GPT2TokenizerFast.from_pretrained('gpt2-medium')
medium_model = GPT2LMHeadModel.from_pretrained('gpt2-medium')
perplexity_dicitonary_gpt2_medium = dict()
# for i in range(len(cleaned_wiki_list_of_strings)):
#   perplexity_dicitonary_gpt2_medium[i]=calculate_perplexity(cleaned_wiki_list_of_strings[i],medium_model,medium_tokeniz
print(perplexity_dicitonary_gpt2_medium)


# HuggingFace GPT2 model - Large
large_tokenizer = GPT2TokenizerFast.from_pretrained('gpt2-large')
large_model = GPT2LMHeadModel.from_pretrained('gpt2-large')

perplexity_dicitonary_gpt2_large = dict()
for i in range(len(cleaned_wiki_list_of_strings)):
  perplexity_dicitonary_gpt2_large[i]=calculate_perplexity(cleaned_wiki_list_of_strings[i],large_model,large_tokenizer)
print(perplexity_dicitonary_gpt2_large)



```

```
items = list(perplexity_dicitonary_gpt2_large.items())
items.sort(key=lambda x: x[1])

items.sort(key=lambda x: x[1], reverse=True)




#Lowest perplexity samples
pprint.pp(cleaned_wiki_list_of_strings[127], depth=1, width=90)
pprint.pp(cleaned_wiki_list_of_strings[83], depth=1, width=90)
pprint.pp(cleaned_wiki_list_of_strings[84], depth=1, width=90)

#Highest perplexity samples
pprint.pp(cleaned_wiki_list_of_strings[93], depth=1, width=90)
pprint.pp(cleaned_wiki_list_of_strings[183], depth=1, width=90)
pprint.pp(cleaned_wiki_list_of_strings[14], depth=1, width=90)

```

```
Sorted in ascending order: [(2, 27.221298217773438), (1, 29.44189453125), (5, 30.019569396972656), (4, 88.4051513671875)
Sorted in descending order: [(0, 1270.773193359375), (3, 360.9991149902344), (4, 88.4051513671875), (5, 30.0195693969726
(" The ships ' secondary armament consisted of twenty 50 @-@ calibre 14 @-@ centimetre "
 'Type 3 . Eighteen of these were mounted in casemates in the forecastle and '
 'superstructure and the remaining pair were mounted on the deck above them and '
 'protected by gun shields . They had a maximum elevation of + 20 degrees which gave '
 'them ranges of 16 @,@ 300 metres ( 17 @,@ 800 yd ) . Each gun had a rate of fire of up '
 'to 10 rounds per minute . Anti @-@ aircraft defence was provided by four 40 @-@ '
 'calibre 3rd Year Type 8 @-@ centimetre AA guns in single mounts . The 7 @.@ 62 @-@ '
 'centimetre ( 3 in ) high @-@ angle guns had a maximum elevation of + 75 degrees , and '
 'had a rate of fire of 13 to 20 rounds per minute . They fired a 6 kg ( 13 lb ) '
 'projectile with a muzzle velocity of 680 m / s ( 2 @,@ 200 ft / s ) to a maximum '
 'height of 7 @,@ 500 metres ( 24 @,@ 600 ft ) . The ships were also fitted with six '
 'submerged 53 @.@ 3 @-@ centimetre ( 21 @.@ 0 in ) torpedo tubes , three on each '
 'broadside . They carried twelve to eighteen 6th Year Type torpedoes which had a 200 '
 '@-@ kilogram ( 440 lb ) warhead . They had three settings for range and speed : 15 @,@ '
 '000 metres ( 16 @,@ 000 yd ) at 26 knots ( 48 km / h ; 30 mph ) , 10 @,@ 000 metres ( '
 '11 @,@ 000 yd ) at 32 knots ( 59 km / h ; 37 mph ) , or 7 @,@ 000 metres ( 7 @,@ 700 '
 'yd ) at 37 knots ( 69 km / h ; 43 mph ) . ')
(' The single made its Irish Singles Chart debut at number 24 on the week ending 13 '
 'December 2012 . It peaked at number seven on the week ending 17 January 2013 , marking '
 'their sixth top ten appearance in Ireland . " Kiss You " entered at number 152 in the '
 'UK Singles Chart on 24 November 2012 . It peaked at number nine on the UK Singles '
 "Chart on 26 January 2013 , becoming One Direction 's sixth top ten hit in the United "
 'Kingdom . On the week ending 18 November 2012 , " Kiss You " debuted at number 90 on '
 'the United States Billboard Hot 100 due to digital download sales from its parent '
 'album . As a result of an " end @-@ of @-@ year download rush " on the week ending 30 '
 'December 2012 , the track re @-@ entered the Hot 100 at number 83 . After the '
 'accompanying music video was released , the song re @-@ entered the Hot 100 at number '
 '65 . " Kiss You " had sold 207 @,@ 000 digital downloads in the US by 18 January 2013 '
 '. The single ultimately peaked at number 46 on the Hot 100 and was certified gold by '
 'the Recording Industry Association of America ( RIAA ) on 25 April 2013 , denoting '
 'shipments of 500 @,@ 000 copies . ')
(" The song became One Direction 's fourth top @-@ forty hit on the Canadian Hot 100 , "
 'peaking at number 30 . The single bowed at number 13 on the Australian Singles Chart '
 "on 27 January 2013 , marking its peak position and the group 's fourth top twenty hit "
 'in Australia . The song has been certified platinum by the Australian Recording '
 'Industry Association ( ARIA ) for shipments of 70 @,@ 000 copies . The track entered '
 'the New Zealand Singles Chart at number 17 on 11 January 2013 . It peaked at number 13 '
 "in its third and fourth charting weeks , becominh the group 's sixth top @-@ forty "
 'appearance in New Zealand . " Kiss You " has received a gold certification from the '
 'Recording Industry Association of New Zealand ( RIANZ ) , indicating sales of 7 @,@ '
 '500 copies . The track also reached the top 40 in both Belgian territories ( Flanders '
 'and Wallonia ) , as well as in the Czech Republic , Denmark , France , the Netherlands '
 ', and South Korea . In addition , " Kiss You " received gold certifications from the '
 'IFPI Norway and Denmark associations , signifying collective shipments of 20 @,@ 000 '
 'units . ')
' CD single '
' = 1933 Treasure Coast hurricane = '
' = Du Fu = '
```

```python
# Print results:

#GPT-2 Model Results
gpt2_values = list(perplexity_dicitonary_gpt2.values())

#25TH AND 75TH PERCENTILES
percentile_25 = np.percentile(gpt2_values, 25)
print("25th percentiles of the perplexity: ", percentile_25)
percentile_75 = np.percentile(gpt2_values, 75)
print("75th percentiles of the perplexity: ", percentile_75)

gpt2_values.sort()
mid = len(gpt2_values) // 2
res = (gpt2_values[mid] + gpt2_values[~mid]) / 2
print("median perplexity: " + str(res))

midhinge = (percentile_25 + percentile_75) / 2
print("midhinge perplexity:",midhinge)

```

```python
# Comments:
# The larger the HuggingFace GPT2 model size, the smaller the perplexity.
# The larger the HuggingFace GPT2 model size, the higher the upperbound of the sentence perplexity perplexity calculated
```

```python
# #Create GPT2-Histogram
gpt2_values = list(perplexity_dicitonary_gpt2.values())
plt.hist(gpt2_values, edgecolor="red", bins=30)

# Adding labels and title
plt.xlabel('Perplexity GPT-2 Small Values')
plt.ylabel('Frequency')
plt.title('GPT-2 Histogram (Small)')

plt.show()

```

```python
#Create a GPT2-Medium Model Histogram
gpt2_medium_values = list(perplexity_dicitonary_gpt2_medium.values())
plt.hist(gpt2_medium_values, edgecolor="red", bins=30)

# Adding labels and title
plt.xlabel('Perplexity GPT-2 Medium Values')
plt.ylabel('Frequency')
plt.title('GPT-2 Histogram (Medium)')

plt.show()
```

```python
#Create a GPT2-Large Model Histogram
gpt2_large_values = list(perplexity_dicitonary_gpt2_large.values())
plt.hist(gpt2_large_values, edgecolor="red", bins=30)

# Adding labels and title
plt.xlabel('Perplexity GPT-2 Large Values')
plt.ylabel('Frequency')
plt.title('GPT-2 Histogram (Large)')

plt.show()
```

```python
#GPT-2 Medium Model Results
gpt2_medium_values = list(perplexity_dicitonary_gpt2_medium.values())

#25TH AND 75TH PERCENTILES

#Percentile Calculation for gpt2
percentile_25 = np.percentile(gpt2_medium_values, 25)
print("25th percentiles of the perplexity: ", percentile_25)
percentile_75 = np.percentile(gpt2_large_values, 75)
print("75th percentiles of the perplexity: ", percentile_75)

gpt2_medium_values.sort()
mid = len(gpt2_medium_values) // 2
res = (gpt2_medium_values[mid] + gpt2_medium_values[~mid]) / 2
print("median perplexity: " + str(res))

midhinge = (percentile_25 + percentile_75) / 2
print("midhinge perplexity:",midhinge)
```

```
#GPT-2 Large Model Results

#25TH AND 75TH PERCENTILES
gpt2_large_values = list(perplexity_dicitonary_gpt2_large.values())
#Percentile Calculation for gpt2
percentile_25 = np.percentile(gpt2_large_values, 25)
print("25th percentiles of the perplexity: ", percentile_25)
percentile_75 = np.percentile(gpt2_large_values, 75)
print("75th percentiles of the perplexity: ", percentile_75)

gpt2_large_values.sort()
mid = len(gpt2_large_values) // 2
res = (gpt2_large_values[mid] + gpt2_large_values[~mid]) / 2
print("median perplexity: " + str(res))

midhinge = (percentile_25 + percentile_75) / 2
print("midhinge perplexity:",midhinge)
```

Problem 2

```
!pip install mauve-text
```

```
Requirement already satisfied: mauve-text in /usr/local/lib/python3.10/dist-packages (0.4.0)
Requirement already satisfied: numpy>=1.18.1 in /usr/local/lib/python3.10/dist-packages (from mauve-text) (1.26.4)
Requirement already satisfied: scikit-learn>=0.22.1 in /usr/local/lib/python3.10/dist-packages (from mauve-text) (1.5.2)
Requirement already satisfied: faiss-cpu>=1.7.0 in /usr/local/lib/python3.10/dist-packages (from mauve-text) (1.8.0.post
Requirement already satisfied: tqdm>=4.40.0 in /usr/local/lib/python3.10/dist-packages (from mauve-text) (4.66.5)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from mauve-text) (2.32.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from faiss-cpu>=1.7.0->mauve-text)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22.1->mauve
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22.1->mauv
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->mauve
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->mauve-text) (3.10
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->mauve-text)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->mauve-text)
```

```
#Imports
from transformers import GPT2Tokenizer, GPT2LMHeadModel

from tqdm import tqdm
import torch
import mauve
import json
import os
import sys
import requests
import pickle

seed = 123
torch.manual_seed(seed)
np.random.seed(seed)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remoun
```

```
# Load test set
path = '/content/drive/MyDrive/CS6983_GenAI_Folder/Homework_1/Problem2_dataset.pkl'

with open(path, 'rb') as file:
    data = pickle.load(file)
print(data)

samples = ""
for i in range(len(data)):
  text_data = data[i]['text']
  samples += text_data
#print(samples)
#print(type(samples))
#print(len(samples))

print(len(samples.split("\n\n")))
text_input = samples.split("\n\n")
print(len(text_input))
```

```
signing him for Aston Villa.\n\nHe stated on Twitter that he should move to Monaco.\n\nGarde admitted he is keen on signi
```

```
device = torch.device("cuda")
large_tokenizer = GPT2TokenizerFast.from_pretrained('gpt2-large')
large_model = GPT2LMHeadModel.from_pretrained('gpt2-large')
```

```
large_tokenizer.eos_token_id
```

```
50256
```

```
#The Greedy Algorithm for Sentence Generation
torch_device = "cuda"
MAX_LEN = 50
TOKEN_ID = large_tokenizer.eos_token_id
greedy_output_sentences = []
for i in range(len(text_input)):
  prefix=text_input[i].split()[:10]
  generation_input = ' '.join(prefix)
  #print(generation_input)
  input_ids = large_tokenizer.encode(generation_input,return_tensors="pt")
  attention_mask = torch.ones(input_ids.shape)
  greedy_output=large_model.generate(input_ids, max_length=MAX_LEN,attention_mask=attention_mask,pad_token_id=TOKEN_ID).t
  greedy_output_sentences.append(large_tokenizer.decode(greedy_output[0], remove_special_tokens = True))


#Top-k:
for i in range(len(text_input)):
  prefix=text_input[i].split()[:10]
  generation_input = ' '.join(prefix)
  input_ids = large_tokenizer.encode(generation_input, return_tensors='pt')
  output_generated = large_model.generate(input_ids, do_sample = True, max_length = MAX_LEN, top_k = 10,attention_mask=Nc
  print(large_tokenizer.decode(output_generated[0],remove_special_tokens=True))


#Beam Search:
beam_generated = []
for i in range(len(text_input)):
  prefix=text_input[i].split()[:10]
  generation_input = ' '.join(prefix)
  input_ids = large_tokenizer.encode(generation_input, return_tensors='pt')
  beam_outputs = large_model.generate(input_ids, max_length=MAX_LEN, num_beams = 5,attention_mask=None,pad_token_id=TOKEN
    beam_generated.append(beam_output)
    print(large_tokenizer.decode(beam_output, remove_special_tokens=True))




```

```
# print(greedy_output_sentences)
```

```
['Is this restaurant family-friendly? Yes No Unsure\n\nCan a gluten free person get a good meal at this restaurant? Yes
```

```
!pip install evaluate
```

```
# Compute the MAUVE score

#list of original text sentences
original_data = []
for i in range(len(text_input)):
  original_data.append(text_input[i])
print(len(original_data))

# print(len(greedy_output_sentences))
# print()
```

```
2414
```

```
#Print results:
print(len(original_data))
print(len(text_input))
# print(type(text_input))
# print(text_input[0])
# print(text_input[60])
# print(text_input[2413])

#Greedy Algorithm Mauve score
def tensor_version_of_sentences(list_str):
    inputs = tokenizer(list_str, return_tensors='pt', padding=True, truncation=True, max_length=50)
    return inputs['input_ids']

generated_tensors = tensor_version_sentences(text_generation_output)
original_tensors = tensor_version_sentences(original)
mauve_results = mauve.compute_mauve(generated_tensors, original_tensors)
print("the mauve score", mauve_results.mauve)

#Top K algorithm Mauve score
def tensor_version_of_sentences(list_str):
    inputs = tokenizer(list_str, return_tensors='pt', padding=True, truncation=True, max_length=50)
    return inputs['input_ids']

generated_tensors = tensor_version_sentences(text_generation_output)
original_tensors = tensor_version_sentences(original)
mauve_results = mauve.compute_mauve(generated_tensors, original_tensors)
print("the mauve score", mauve_results.mauve)


#Beam Algorithm Mauve score
def tensor_version_of_sentences(list_str):
    inputs = tokenizer(list_str, return_tensors='pt', padding=True, truncation=True, max_length=50)
    return inputs['input_ids']
generated_tensors = tensor_version_sentences(text_generation_output)
original_tensors = tensor_version_sentences(original)
mauve_results = mauve.compute_mauve(generated_tensors, original_tensors)
print("the mauve score", mauve_results.mauve)
```

```
2414
2414
the mauve score 0.0040720962619612555
```

```
# Select the sentence of the highest and lowest MAUVE score
#for each generation strategy and print it. Discuss your observations.

# The sentence with the highest mauve score was:
#The ships ' secondary armament consisted of twenty 50 @-@ calibre 14 @-@ centimetre "
 #'Type 3 . Eighteen of these were mounted in casemates in the forecastle and '
#  'superstructure and the remaining pair were mounted on the deck above them and '
#  'protected by gun shields . They had a maximum elevation of + 20 degrees which gave '
#  'them ranges of 16 @,@ 300 metres ( 17 @,@ 800 yd ) . Each gun had a rate of fire of up '
#  'to 10 rounds per minute . Anti @-@ aircraft defence was provided by four 40 @-@ '
#  'calibre 3rd Year Type 8 @-@ centimetre AA guns in single mounts . The 7 @.@ 62 @-@ '
#  'centimetre ( 3 in ) high @-@ angle guns had a maximum elevation of + 75 degrees , and '
#  'had a rate of fire of 13 to 20 rounds per minute . They fired a 6 kg ( 13 lb ) '
#  'projectile with a muzzle velocity of 680 m / s ( 2 @,@ 200 ft / s ) to a maximum '
#  'height of 7 @,@ 500 metres ( 24 @,@ 600 ft ) .

# The sentnce with the lowest mauve score was:
# CD single

#It would be expected that the sentences with the more complex grammartical stucture,
#or unreleated words next to one another would have a lowest mauve score compared to
#Much more normal and fluent language/sentences

```

Problem 3

```
!pip install datasets scikit-learn
```

```
!pip install pyarrow==15.0.2
```

```
Requirement already satisfied: pyarrow==15.0.2 in /usr/local/lib/python3.10/dist-packages (15.0.2)
Requirement already satisfied: numpy<2,>=1.16.6 in /usr/local/lib/python3.10/dist-packages (from pyarrow==15.0.2) (1.26.
```

```
import torch
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification, AdamW
from transformers import get_scheduler
from transformers import TrainingArguments, Trainer
from datasets import load_dataset
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from torch.utils.data import DataLoader
from tqdm.auto import tqdm
from sklearn.metrics import roc_curve, auc

```

```
# Load IMDb dataset and limit to 1000 samples
dataset = load_dataset('imdb')
```

```python
train_dataset = dataset['train'].shuffle(seed=42).select(range(1000))
test_dataset = dataset['test'].shuffle(seed=42).select(range(100)) #Select 100 Samples
```

```python
print('5 Training examples')
for i, example in enumerate(train_dataset['text'][:5]):
  print(f'{i+1}. {example}')

```

```
5 Training examples
1. There is no relation at all between Fortier and Profiler but the fact that both are police series about violent crime
2. This movie is a great. The plot is very true to the book which is a classic written by Mark Twain. The movie starts o
3. George P. Cosmatos' "Rambo: First Blood Part II" is pure wish-fulfillment. The United States clearly didn't win the w
4. In the process of trying to establish the audiences' empathy with Jake Roedel (Tobey Maguire) the filmmakers slander
5. Yeh, I know -- you're quivering with excitement. Well, *The Secret Lives of Dentists* will not upset your expectation
```

```python
print('5 Testing examples')
for i, example in enumerate(test_dataset['text'][:5]):
  print(f'{i+1}. {example}')

```

```
5 Testing examples
1. <br /><br />When I unsuspectedly rented A Thousand Acres, I thought I was in for an entertaining King Lear story and
2. This is the latest entry in the long series of films with the French agent, O.S.S. 117 (the French answer to James Bo
3. This movie was so frustrating. Everything seemed energetic and I was totally prepared to have a good time. I at least
4. I was truly and wonderfully surprised at "O' Brother, Where Art Thou?" The video store was out of all the movies I wa
5. This movie spends most of its time preaching that it is the script that makes the movie, but apparently there was no
```

```python
# Complete code for Fine-tuning BERT for sentiment classification
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')

DistilBertmodel = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=2)


for layer in DistilBertmodel.parameters():
    layer.requires_grad = False

for layer in DistilBertmodel.classifier.parameters():
    layer.requires_grad = True

```

```
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenizat
  warnings.warn(
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-un
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```python

def tokenize_function(examples):
    return tokenizer(examples['text'], padding='max_length', truncation=True)


tokenized_train = train_dataset.map(tokenize_function, batched=True)
tokenized_eval = test_dataset.map(tokenize_function, batched=True)
```

```python
training_args = TrainingArguments(
    output_dir="my_model_directory",
    learning_rate=1e-5,
    per_device_train_batch_size=64,
    per_device_eval_batch_size=16,
    num_train_epochs=5,
)

trainer = Trainer(
    model=DistilBertmodel,
    args=training_args,
    train_dataset= tokenized_train,
    eval_dataset=tokenized_eval,
)


trainer.train()

```

[80/80 01:37, Epoch 5/5]

```
Step  Training Loss
TrainOutput(global_step=80, training_loss=0.6880020141601563, metrics={'train_runtime': 98.9664,
'train_samples_per_second': 50.522, 'train_steps_per_second': 0.808, 'total_flos': 662336993280000.0, 'train_loss':
0.6880020141601563, 'epoch': 5.0})
```

```python
# Print results:
eval_results = trainer.predict(tokenized_eval)
```

```

preds = np.argmax(eval_results.predictions, axis=1)
labels = eval_results.label_ids

# Accuracy
accuracy = accuracy_score(labels, preds)

# Precision, Recall, F1 Score
precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average='weighted')

# Error rate (1 - Accuracy)
error_rate = 1 - accuracy


```

```
#Printing all metrics
print("Accuracy metric", accuracy)
print("Precision, Recall, F1 Score", precision, recall, f1)
print("Error rate", 1 - accuracy)
```

```
Accuracy metric 0.59
Precision, Recall, F1 Score 0.7108152173913044 0.59 0.49977429467084633
Error rate 0.41000000000000003
```

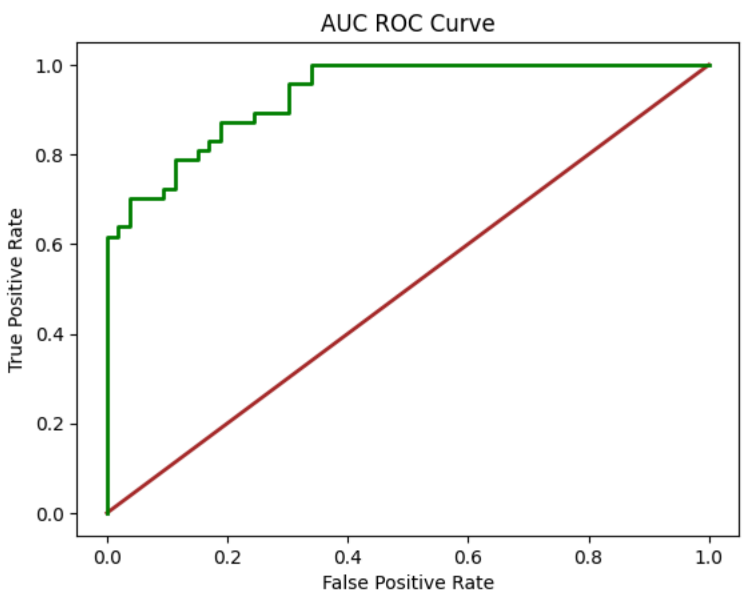
```
modl_eval_results = trainer.predict(tokenized_eval)
labels = eval_results.label_ids
model_y_scores = modl_eval_results.predictions[:, 1]

fpr, tpr, thresholds = roc_curve(labels, model_y_scores)
roc_auc = auc(fpr, tpr)


plt.figure()
plt.plot([0, 1], [0, 1], color='brown', lw=2)
plt.plot(fpr, tpr, color='green', lw=2)

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('AUC ROC Curve')
plt.show()

```



```
for layer in DistilBertmodel.parameters():
    layer.requires_grad = True
```

```
training_args = TrainingArguments(
    output_dir="my_model_directory",
    learning_rate=1e-5,
    per_device_train_batch_size=64,
    per_device_eval_batch_size=16,
    num_train_epochs=5,
)

```

```python

trainer = Trainer(
    model=DistilBertmodel,
    args=training_args,
    train_dataset= tokenized_train,
    eval_dataset=tokenized_eval,
)


trainer.train()
```

[315/315 04:05, Epoch 5/5]

Step	Training Loss
10	0.676600
20	0.609200
30	0.488300
40	0.455800
50	0.321100
60	0.368800
70	0.438300
80	0.214400
90	0.232200
100	0.255800
110	0.206000
120	0.232500
130	0.231100
140	0.150000
150	0.102500
160	0.126000
170	0.066800
180	0.146000
190	0.120100
200	0.070200
210	0.007700
220	0.007100
230	0.037300
240	0.007200
250	0.110700
260	0.027000
270	0.004600
280	0.003200
290	0.018800
300	0.039000
310	0.037200

```
TrainOutput(global_step=315, training_loss=0.18454010839618387, metrics={'train_runtime': 246.1843,
'train_samples_per_second': 20.31, 'train_steps_per_second': 1.28, 'total_flos': 679292820307968.0, 'train_loss':
0.18454010839618387, 'epoch': 5.0})
```

```python
eval_results = trainer.evaluate()
print("The cross-entropy loss",eval_results['eval_loss'])
```

[2/2 07:42]

```
The test cross-entropy loss 0.8423845767974854
```

```python
print("Accuracy metric", accuracy)
print("Precision, Recall, F1 Score", precision, recall, f1)
print("Error rate", 1 - accuracy)
```

```
Accuracy metric 0.82
Precision, Recall, F1 Score 0.8300202020202019 0.82 0.8197118847539017
Error rate 0.18000000000000005
```

```python
#Print results:
eval_results = trainer.predict(tokenized_eval)

preds = np.argmax(eval_results.predictions, axis=1)
labels = eval_results.label_ids

# Accuracy
accuracy = accuracy_score(labels, preds)

# Precision, Recall, F1 Score
precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average='weighted')

# Error rate (1 - Accuracy)
error_rate = 1 - accuracy
```

```python
# Comments:
# It is evident that the models that have been fine tuned on the last layer
# are much more accurate at classification compared to the models that have only
# been fine tuned on the final layer.
```