algorithm6-kaggle_AXC210110

April 16, 2024

[ ]:
```
import os
from   transformers import pipeline
import torch
from   transformers import MarianMTModel, MarianTokenizer
from   torch.utils.data import Dataset, DataLoader, TensorDataset
import csv
import ast
import json
import pandas as pd
from   transformers import  Trainer, TrainingArguments
from   transformers import DistilBertForSequenceClassification,␣
 ↪DistilBertTokenizer
from   transformers import BertTokenizer, BertForSequenceClassification, AdamW
from   sklearn.model_selection import train_test_split
from   sklearn.preprocessing import LabelEncoder
from   keras.utils import to_categorical
from   nltk.tokenize import word_tokenize
from   nltk.corpus import stopwords
from   nltk.stem import PorterStemmer
from   datasets import load_dataset, Dataset
from   sklearn.metrics import accuracy_score
from   tqdm import tqdm


if __name__ =="__main__":
  print("hi")
```

```
hi
```

[ ]:
```
#Text Classification #Sentiment Analysis #Text Translation
#Using Pytorch and Hugging Face


#Name   : Ashwin Sai C
#Course : NLP - CS6320-001
#Title  : Text Classification, Text Translation and Sentiment Analysis using␣
 ↪Hugging Face & Pytorch
#Term   : Spring 2024
```

[ ]:
```python
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
nltk.download("words")
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data…

[nltk_data]   Package stopwords is already up-to-date!

[nltk_data] Downloading package wordnet to /usr/share/nltk_data…

[nltk_data]   Package wordnet is already up-to-date!

[nltk_data] Downloading package punkt to /usr/share/nltk_data…

[nltk_data]   Package punkt is already up-to-date!

[nltk_data] Downloading package words to /usr/share/nltk_data…

[nltk_data]   Package words is already up-to-date!
```

[ ]: True

1 Sentimental Analysis : Transformer

[ ]:
```python
classifier = pipeline("sentiment-analysis")

statement  = ["Hey, the internship scenario looks really worried, hope it gets␣
↪better soon!", "Hey, the NLP is really nice."]
print(classifier(statement))
```

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-
finetuned-sst-2-english and revision af0f99b
(https://huggingface.co/distilbert/distilbert-base-uncased-finetuned-
sst-2-english).

Using a pipeline without specifying a model name and revision in production is
not recommended.

config.json:   0%|          | 0.00/629 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/268M [00:00<?, ?B/s]

tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

[{'label': 'NEGATIVE', 'score': 0.9908674955368042}, {'label': 'POSITIVE',
'score': 0.999823272228241}]
```

2 Zero-Shot Classification

[ ]:
```
classifier = pipeline("zero-shot-classification")
classifier("Black-holes have a really strong gravitational pull and cant be␣
↪seen with the naked eye.", candidate_labels =␣
↪["astronomy","astrology","sports"])
```

```
No model was supplied, defaulted to facebook/bart-large-mnli and revision
c626438 (https://huggingface.co/facebook/bart-large-mnli).

Using a pipeline without specifying a model name and revision in production is
not recommended.
```

```
config.json:   0%|          | 0.00/1.15k [00:00<?, ?B/s]
```

```
model.safetensors:   0%|          | 0.00/1.63G [00:00<?, ?B/s]
```

```
tokenizer_config.json:   0%|          | 0.00/26.0 [00:00<?, ?B/s]
```

```
vocab.json:   0%|          | 0.00/899k [00:00<?, ?B/s]
```

```
merges.txt:   0%|          | 0.00/456k [00:00<?, ?B/s]
```

```
tokenizer.json:   0%|          | 0.00/1.36M [00:00<?, ?B/s]
```

[ ]:
```
{'sequence': 'Black-holes have a really strong gravitational pull and cant be
seen with the naked eye.',
 'labels': ['astronomy', 'astrology', 'sports'],
 'scores': [0.4928959906101227, 0.30658620595932007, 0.20051778852939606]}
```

3 Text Generation

[ ]:
```
generator = pipeline("text-generation")
statement = "Natural language Processing teaches about how Machine learning␣
↪methods can be"
generator(statement)
```

```
No model was supplied, defaulted to openai-community/gpt2 and revision 6c0e608
(https://huggingface.co/openai-community/gpt2).

Using a pipeline without specifying a model name and revision in production is
not recommended.
```

```
config.json:   0%|          | 0.00/665 [00:00<?, ?B/s]
```

```
model.safetensors:   0%|          | 0.00/548M [00:00<?, ?B/s]
```

```
generation_config.json:   0%|          | 0.00/124 [00:00<?, ?B/s]
```

```
tokenizer_config.json:   0%|          | 0.00/26.0 [00:00<?, ?B/s]
```

```
vocab.json:   0%|          | 0.00/1.04M [00:00<?, ?B/s]
```

```
merges.txt:   0%|          | 0.00/456k [00:00<?, ?B/s]
```

```
tokenizer.json:   0%|          | 0.00/1.36M [00:00<?, ?B/s]

Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

[ ]:
```
[{'generated_text': 'Natural language Processing teaches about how Machine
learning methods can be applied to data to give accurate predictions of language
and cognition from other systems. This provides an overview of the field of
computer vision and computer science in general. It also reveals some
fundamental new tools for'}]
```

[ ]:
```python
generator = pipeline("text-generation", model="distilgpt2")
generator("Natural language Processing teaches about how Machine learning␣
↪methods can be", max_length=30, num_return_sequences=4)
```

```
config.json:   0%|          | 0.00/762 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/353M [00:00<?, ?B/s]

generation_config.json:   0%|          | 0.00/124 [00:00<?, ?B/s]

tokenizer_config.json:   0%|          | 0.00/26.0 [00:00<?, ?B/s]

vocab.json:   0%|          | 0.00/1.04M [00:00<?, ?B/s]

merges.txt:   0%|          | 0.00/456k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/1.36M [00:00<?, ?B/s]

Truncation was not explicitly activated but `max_length` is provided a specific
value, please use `truncation=True` to explicitly truncate examples to max
length. Defaulting to 'longest_first' truncation strategy. If you encode pairs
of sequences (GLUE-style) with the tokenizer you can select this strategy more
precisely by providing a specific strategy to `truncation`.

Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

[ ]:
```
[{'generated_text': 'Natural language Processing teaches about how Machine
learning methods can be helpful. It  is a very helpful language, but in that
it s not'},
 {'generated_text': 'Natural language Processing teaches about how Machine
learning methods can be used to improve performance and performance of
programming language, especially in languages like Python and C#.'},
 {'generated_text': 'Natural language Processing teaches about how Machine
learning methods can be integrated into your software.\n\n\n\n\nMachine learning
is an excellent tool for learning software'},
 {'generated_text': 'Natural language Processing teaches about how Machine
learning methods can be implemented in complex languages, and how to implement
Machine Learning concepts in many languages, but there is'}]
```

[ ]:
```python
ner = pipeline("ner", grouped_entities=True)
statement = "My name is Ashwin Sai and I play Volleyball."
ner(statement)
```

```
No model was supplied, defaulted to dbmdz/bert-large-cased-finetuned-conll03-english and revision f2482bf (https://huggingface.co/dbmdz/bert-large-cased-finetuned-conll03-english).

Using a pipeline without specifying a model name and revision in production is not recommended.

Some weights of the model checkpoint at dbmdz/bert-large-cased-finetuned-conll03-english were not used when initializing BertForTokenClassification: ['bert.pooler.dense.bias', 'bert.pooler.dense.weight']

- This IS expected if you are initializing BertForTokenClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).

- This IS NOT expected if you are initializing BertForTokenClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
```

[ ]:
```
[{'entity_group': 'PER',
  'score': 0.9993536,
  'word': 'Ashwin Sai',
  'start': 11,
  'end': 21}]
```

4 Question Answering

[ ]:
```python
question_ans = pipeline("question-answering")
question_ans(question="What do I play?",context="My name is Ashwin Sai and I␣
↪play Volleyball.")
```

```
No model was supplied, defaulted to distilbert/distilbert-base-cased-distilled-squad and revision 626af31 (https://huggingface.co/distilbert/distilbert-base-cased-distilled-squad).

Using a pipeline without specifying a model name and revision in production is not recommended.

config.json:   0%|          | 0.00/473 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/261M [00:00<?, ?B/s]

tokenizer_config.json:   0%|          | 0.00/29.0 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/213k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/436k [00:00<?, ?B/s]
```

[ ]: {'score': 0.9757400751113892, 'start': 33, 'end': 43, 'answer': 'Volleyball'}

5 Summarization

```
[ ]: summarizer = pipeline("summarization")
     summarizer("The paper explores various techniques commonly employed in␣
       ↪low-resource machine translation (MT) scenarios to translate spoken language␣
       ↪text into sign language glosses. These techniques include data augmentation,␣
       ↪semi-supervised NMT, transfer learning, and multilingual NMT. Extensive␣
       ↪experimentation was conducted on two natural datasets with gloss annotation:␣
       ↪the RWTH-PHOENIX-Weather 2014T dataset and the Public DGS Corpus. Results␣
       ↪indicate significant improvement in evaluation scores for both datasets,␣
       ↪with the Multilingual NMT model yielding the best performance (6.18 and 2.65␣
       ↪BLEU scores against the baseline, respectively). The best-performing system␣
       ↪outperforms previous state-of-the-art systems on the same test sets.␣
       ↪Furthermore, the effectiveness of the best setting is confirmed through␣
       ↪experiments on a corpus of American Sign Language (ASL). Human evaluation␣
       ↪supports the conclusions drawn from the automatic metrics.")
```

```
No model was supplied, defaulted to sshleifer/distilbart-cnn-12-6 and revision
a4f8f3e (https://huggingface.co/sshleifer/distilbart-cnn-12-6).

Using a pipeline without specifying a model name and revision in production is
not recommended.

config.json:   0%|          | 0.00/1.80k [00:00<?, ?B/s]

pytorch_model.bin:   0%|          | 0.00/1.22G [00:00<?, ?B/s]

/opt/conda/lib/python3.10/site-packages/torch/_utils.py:831: UserWarning:
TypedStorage is deprecated. It will be removed in the future and UntypedStorage
will be the only storage class. This should only matter to you if you are using
storages directly.  To access UntypedStorage directly, use
tensor.untyped_storage() instead of tensor.storage()

  return self.fget.__get__(instance, owner)()

tokenizer_config.json:   0%|          | 0.00/26.0 [00:00<?, ?B/s]

vocab.json:   0%|          | 0.00/899k [00:00<?, ?B/s]

merges.txt:   0%|          | 0.00/456k [00:00<?, ?B/s]
```

```
[ ]: [{'summary_text': ' The paper explores various techniques commonly employed in
     low-resource machine translation (MT) scenarios . These techniques include data
     augmentation, semi-supervised NMT, transfer learning, and multilingual NMT .
     Extensive experimentation was conducted on two natural datasets with gloss
     annotation . Results indicate significant improvement in evaluation scores for
     both datasets .'}]
```

6 Translation

[ ]:
```
translator = pipeline("translation",model="Helsinki-NLP/opus-mt-fr-en")
statement  = "je m appelle ashwin sai"
translator(statement)
```

```
config.json:   0%|          | 0.00/1.42k [00:00<?, ?B/s]

pytorch_model.bin:   0%|          | 0.00/301M [00:00<?, ?B/s]

generation_config.json:   0%|          | 0.00/293 [00:00<?, ?B/s]

tokenizer_config.json:   0%|          | 0.00/42.0 [00:00<?, ?B/s]

source.spm:   0%|          | 0.00/802k [00:00<?, ?B/s]

target.spm:   0%|          | 0.00/778k [00:00<?, ?B/s]

vocab.json:   0%|          | 0.00/1.34M [00:00<?, ?B/s]

/opt/conda/lib/python3.10/site-
packages/transformers/models/marian/tokenization_marian.py:197: UserWarning:
Recommended: pip install sacremoses.

  warnings.warn("Recommended: pip install sacremoses.")
```

[ ]:
```
[{'translation_text': 'my name is Ashwin sai'}]
```

[ ]:
```
# Define your translation dataset class
class TranslationDataset(Dataset):
    def __init__(self, source_texts, target_texts, tokenizer,␣
 ↪max_source_length, max_target_length):
        self.source_texts      = source_texts
        self.target_texts      = target_texts
        self.tokenizer         = tokenizer
        self.max_source_length = max_source_length
        self.max_target_length = max_target_length

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

    def __getitem__(self, idx):
        source_text   = self.source_texts[idx]
        target_text   = self.target_texts[idx]
        source_tokens = self.tokenizer.encode(source_text, max_length=self.
 ↪max_source_length, padding='max_length', truncation=True)
        target_tokens = self.tokenizer.encode(target_text, max_length=self.
 ↪max_target_length, padding='max_length', truncation=True)

        return {'input_ids': torch.tensor(source_tokens, dtype=torch.long),
```

```
                'attention_mask': torch.tensor([1] * len(source_tokens),␣
↪dtype=torch.long),
                'labels': torch.tensor(target_tokens, dtype=torch.long)}
```

[ ]:
```
# Load pretrained model and tokenizer
model_name = 'Helsinki-NLP/opus-mt-en-fr'
tokenizer  = MarianTokenizer.from_pretrained(model_name)
model      = MarianMTModel.from_pretrained(model_name)
```

```
tokenizer_config.json:   0%|          | 0.00/42.0 [00:00<?, ?B/s]

source.spm:   0%|          | 0.00/778k [00:00<?, ?B/s]

target.spm:   0%|          | 0.00/802k [00:00<?, ?B/s]

vocab.json:   0%|          | 0.00/1.34M [00:00<?, ?B/s]

config.json:   0%|          | 0.00/1.42k [00:00<?, ?B/s]

pytorch_model.bin:   0%|          | 0.00/301M [00:00<?, ?B/s]

generation_config.json:   0%|          | 0.00/293 [00:00<?, ?B/s]
```

[1]:
```
#Dataset used : de-en_train.csv

#The dataset contains german language and english language translation.
#It is used to translate german -> english language.
#Further details of the dataset regarding the data is given below.
```

[ ]:
```
file_name = "/kaggle/input/de-en-file-csv/de-en_train.csv"
file_handle = open(file_name,"r")
csv_reader = csv.reader(file_handle)

de_language = []
en_language = []
count = 0

for row in list(csv_reader)[:5000]:
  json_acceptable_string = row[1].replace("'", "\"")
  try:
    d = json.loads(json_acceptable_string)
    # print("-----")
    # print(d['de'])
    # print(d['en'])
    # print("-----")
    de_language.append(d['de'])
    en_language.append(d['en'])
    count+=1
  except Exception as e:
    pass
```

```
print("Valid Translation count : ",count)
print("Valid de language count : ",len(de_language))
print("Valid en language count : ",len(en_language))

file_handle.close()
```

```
Valid Translation count :  2263

Valid de language count :  2263

Valid en language count :  2263
```

[ ]:
```
source_texts = en_language.copy()
target_texts = de_language.copy()
max_source_length = 20
max_target_length = 20
translation_dataset = TranslationDataset(source_texts, target_texts, tokenizer,␣
↪max_source_length, max_target_length)
```

[ ]:
```
# Step 2: Run the data through your model and evaluate the results
def evaluate_translation_model(model, dataset):
    model.eval()
    total_loss = 0
    with torch.no_grad():
        for batch in DataLoader(dataset, batch_size=100):
            input_ids      = batch['input_ids'].to(model.device)
            attention_mask = batch['attention_mask'].to(model.device)
            labels         = batch['labels'].to(model.device)
            outputs        = model(input_ids=input_ids,␣
↪attention_mask=attention_mask, labels=labels)
            total_loss    += outputs.loss.item()

    return total_loss / len(dataset)

initial_loss = evaluate_translation_model(model, translation_dataset)
print("Initial loss:", initial_loss)
```

```
Initial loss: 0.08289358397790199
```

[ ]:
```
# Step 3: Fine-tune the pretrained model
def fine_tune_translation_model(model, dataset, num_epochs=3,␣
↪learning_rate=1e-5):
    model.train()
    optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate)
    for epoch in range(num_epochs):
        total_loss = 0
```

```
        for batch in DataLoader(dataset, batch_size=100, shuffle=True):
            input_ids      = batch['input_ids'].to(model.device)
            attention_mask = batch['attention_mask'].to(model.device)
            labels         = batch['labels'].to(model.device)
            optimizer.zero_grad()
            outputs        = model(input_ids=input_ids,␣
 ↪attention_mask=attention_mask, labels=labels)
            loss           = outputs.loss
            loss.backward()
            optimizer.step()
            total_loss    += loss.item()
        avg_loss           = total_loss / len(dataset)

        print(f"Epoch {epoch+1}/{num_epochs}, Average Loss: {avg_loss:.4f}")
```

[ ]:
```
# Fine-tune the translation model
fine_tune_translation_model(model, translation_dataset)
```

```
Epoch 1/3, Average Loss: 0.0711

Epoch 2/3, Average Loss: 0.0607

Epoch 3/3, Average Loss: 0.0558
```

[ ]:
```
# Step 4: Run the data through the fine-tuned model and compare the results to␣
 ↪the previous model
fine_tuned_loss = evaluate_translation_model(model, translation_dataset)
print("Fine-tuned loss:", fine_tuned_loss)
```

```
Fine-tuned loss: 0.05220528656120486
```

[ ]:
```
# Compare the results
print("Difference in loss:", initial_loss - fine_tuned_loss)
```

```
Difference in loss: 0.03068829741669713
```

7 Text Classification

[ ]:
```
#This dataset is about the IT Service Ticket Categeory Classification.
#By accurately classifying tickets, IT teams can prioritize tasks, allocate␣
 ↪resources effectively,
#and streamline the resolution process.
#The dataset contains ticket descriptions and category type as target field
#We train the dataset on the descriptions and map it to the target field using␣
 ↪Supervised learning.
#The objective of the model is to classify which category a ticket should be␣
 ↪assigned to.
#The following cells gives a glance about the dataset used.
```

[ ]:
```python
#Read the csv file
file_name = r"/kaggle/input/it-ticket-dataset/all_tickets_processed_improved_v3.
↪csv"
df        = pd.read_csv(file_name)
```

[ ]:
```python
#Print the head data section first 5 rows
df.head()
```

[ ]:

	Document	Topic_group
0	connection with icon icon dear please setup ic…	Hardware
1	work experience user work experience user hi w…	Access
2	requesting for meeting requesting meeting hi p…	Hardware
3	reset passwords for external accounts re expir…	Access
4	mail verification warning hi has got attached …	Miscellaneous

[ ]:
```python
#Describe the data set
df.describe()
```

[ ]:

	Document	Topic_group
count	47837	47837
unique	47837	8
top	running out on extensions hello please be advi…	Hardware
freq	1	13617

[ ]:
```python
print("The list of IT Service Requests:\n")
df['Document']
```

```
The list of IT Service Requests:
```

[ ]:
```
0        connection with icon icon dear please setup ic…
1        work experience user work experience user hi w…
2        requesting for meeting requesting meeting hi p…
3        reset passwords for external accounts re expir…
4        mail verification warning hi has got attached …
                               …
47832    git space for a project issues with adding use…
47833    error sent july error hi guys can you help out…
47834    connection issues sent tuesday july connection…
47835    error cube reports sent tuesday july error hel…
47836    running out on extensions hello please be advi…
Name: Document, Length: 47837, dtype: object
```

[ ]:
```python
print("The list of Topic_group:\n")
df['Topic_group']
```

```
The list of Topic_group:
```

[ ]:
```
0              Hardware
1                Access
2              Hardware
3                Access
4         Miscellaneous
                  …
47832            Access
47833     Miscellaneous
47834          Hardware
47835        HR Support
47836          Hardware
Name: Topic_group, Length: 47837, dtype: object
```

[ ]:
```python
#Unique topic group
topic_group_list = list(df['Topic_group'])
unique_topic_set = set(topic_group_list)
print("The different types of Topic groups:\n")
print(unique_topic_set)
```

```
The different types of Topic groups:



{'Purchase', 'Access', 'Hardware', 'Internal Project', 'Administrative rights',
'Storage', 'HR Support', 'Miscellaneous'}
```

[ ]:
```python
X = list(df['Document'])
Y = list(df['Topic_group'])

print("Data length (X, Y) is (",len(X),",",len(Y),")")
```

```
Data length (X, Y) is ( 47837 , 47837 )
```

[ ]:
```python
#Using LabelEncoder converting Y values into unique integers
# Encoding labels
def Y_Encoder_function(Y):
    label_encoder   = LabelEncoder()
    y_encoded       = label_encoder.fit_transform(Y)

    return y_encoded

Y_encoded = Y_Encoder_function(Y)
print("Length of Y dataset : ",len(Y_encoded))
```

```
Length of Y dataset :  47837
```

[ ]:
```python
#Converting to Categorical Features
num_classes = len(set(Y))
print("Number of Classes : ",num_classes)

X_new = []

#Data preprocessing

#lower case the documents
X_lower = [i.lower() for i in X]
print("----Lower casing of terms----")
#Tokenization
X_Tokens = [word_tokenize(i) for i in X_lower]
print("----Tokenization of terms----")
#Removable of Punctuation and Non-Alpha
X_alpha  = [[word for word in doc if word.isalnum()]for doc in X_Tokens]
print("----Filtering out alphanumeric terms----")
#Removal of stop words
stop_words = set(stopwords.words('english'))
X_without_stopwords = [[word for word in doc if word not in stop_words]for doc␣
↪in X_alpha]
print("----Removing Stop words----")
#Lemmatize the words
ps = PorterStemmer()
X_stemmed   = [[ps.stem(word) for word in doc]for doc in X_without_stopwords]
print("----Stemming of terms----")

X = [" ".join(set(row)) for row in X_stemmed]
# X = [" ".join(row) for row in X_without_stopwords]
print("\nNo. of Documents:",len(X))
```

```
Number of Classes :  8

----Lower casing of terms----

----Tokenization of terms----

----Filtering out alphanumeric terms----

----Removing Stop words----

----Stemming of terms----



No. of Documents: 47837
```

```
[ ]: # Assuming you have data in the form of lists: texts and labels
texts  = X
labels = Y_encoded  # 0 for one class, 1 for another, etc.

# Tokenize texts
tokenizer       = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
tokenized_texts = tokenizer(texts, padding=True, truncation=True,␣
 ↪return_tensors='pt')

# Split data into training and testing sets
train_inputs, test_inputs, train_labels, test_labels =␣
 ↪train_test_split(tokenized_texts['input_ids'],torch.
 ↪tensor(labels),test_size=0.2, random_state=42)
```

```
tokenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

config.json:   0%|          | 0.00/483 [00:00<?, ?B/s]
```

```
[ ]: batch_size = 32

train_data   = TensorDataset(train_inputs, train_labels)
train_loader = DataLoader(train_data, batch_size=batch_size)

test_data    = TensorDataset(test_inputs, test_labels)
test_loader  = DataLoader(test_data, batch_size=batch_size)
```

```
[ ]: num_labels = num_classes
model = DistilBertForSequenceClassification.
 ↪from_pretrained('distilbert-base-uncased', num_labels=num_labels)
```

```
model.safetensors:   0%|          | 0.00/268M [00:00<?, ?B/s]

Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'pre_classifier.bias',
'pre_classifier.weight']

You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

```
[ ]: # Evaluate the model
device = torch.device('cuda')
model.to(device)

model.eval()
total_correct = 0
```

```
total_samples = 0

with torch.no_grad():
    for batch in test_loader:
        batch = tuple(t.to(device) for t in batch)
        inputs, labels = batch

        outputs = model(inputs)
        _, predicted   = torch.max(outputs.logits, 1)
        total_samples += labels.size(0)
        total_correct += (predicted == labels).sum().item()

accuracy = total_correct / total_samples
print(f'Pre-Tune-Test Accuracy: {accuracy*100:.2f} %')
```

```
We strongly recommend passing in an `attention_mask` since your input_ids may be
padded. See https://huggingface.co/docs/transformers/troubleshooting#incorrect-
output-when-padding-tokens-arent-masked.
```

```
Pre-Tune-Test Accuracy: 28.85 %
```

[ ]:
```
optimizer = torch.optim.Adam(model.parameters(), lr=2e-5)
criterion = torch.nn.CrossEntropyLoss()
```

[ ]:
```
epochs = 3
# device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
# device = torch.device('cuda')
# model.to(device)

for epoch in range(epochs):
    model.train()
    total_loss = 0

    for batch in train_loader:
        batch = tuple(t.to(device) for t in batch)
        inputs, labels = batch

        optimizer.zero_grad()
        outputs = model(inputs, labels=labels)
        loss = criterion(outputs.logits, labels)
        total_loss += loss.item()

        loss.backward()
        optimizer.step()

    avg_train_loss = total_loss / len(train_loader)

    print(f'Epoch {epoch+1}/{epochs}')
```

```
    print(f'Training Loss: {avg_train_loss:.4f}')
```

```
Epoch 1/3

Training Loss: 0.9104

Epoch 2/3

Training Loss: 0.5326

Epoch 3/3

Training Loss: 0.4348
```

[ ]:
```
model.eval()
total_correct = 0
total_samples = 0

with torch.no_grad():
    for batch in test_loader:
        batch = tuple(t.to(device) for t in batch)
        inputs, labels = batch

        outputs = model(inputs)
        _, predicted = torch.max(outputs.logits, 1)
        total_samples += labels.size(0)
        total_correct += (predicted == labels).sum().item()

accuracy = total_correct / total_samples
print(f'Test Accuracy: {accuracy*100:.2f} %')
```

```
Test Accuracy: 82.10 %
```

8 Sentiment Analysis

[ ]:
```
#The Dataset used is : Twitter Data Tweets for Sentiment Analysis
#Used to predict the sentiment of the tweets.
#-1 - Negative
#0  - Neutral
#+1 - Positive
```

[ ]:
```
# Load custom CSV file
df = pd.read_csv("/kaggle/input/twitter-data/Twitter_Data.csv")  # Assuming␣
 ↪'custom_data.csv' contains 'text' and 'label' columns

#Replace -1 with 2 #Cant process with negative values
df['category'] = df['category'].replace({-1:2})
```

```python
# Drop rows with missing values
df = df.dropna()

# Check if there are any NaN values remaining
if df.isnull().values.any():
    raise ValueError("There are still NaN values in the DataFrame after␣
↪dropping.")
```

[ ]:
```python
# Load pre-trained BERT model and tokenizer
model_name = 'bert-base-uncased'
tokenizer  = BertTokenizer.from_pretrained(model_name)
model      = BertForSequenceClassification.from_pretrained(model_name,␣
↪num_labels=3)
```

```
tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/440M [00:00<?, ?B/s]

Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight']

You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[ ]:
```python
# Tokenize input text
def tokenize_text(text):
    return tokenizer.encode_plus(
        text,
        max_length=128,
        padding='max_length',
        truncation=True,
        return_attention_mask=True,
        return_tensors='pt'
    )
```

[ ]:
```python
# Split data into train and test sets
train_data, test_data = train_test_split(df[:50000], test_size=0.2,␣
↪random_state=42)

print("Length of Train data : ",len(train_data))
print("Length of Test  data : ",len(test_data))
```

```
Length of Train data :  40000

Length of Test  data :  10000
```

[ ]:
```
# Create DataLoader for training and testing
def create_data_loader(data, tokenizer, max_length, batch_size):
    texts = data['clean_text'].tolist()
    labels = data['category'].tolist()
    inputs = [tokenize_text(text) for text in texts]
    input_ids = torch.cat([inputs[i]['input_ids'] for i in range(len(inputs))],␣
 ↪dim=0)
    attention_masks = torch.cat([inputs[i]['attention_mask'] for i in␣
 ↪range(len(inputs))], dim=0)
    labels = torch.tensor(labels, dtype=torch.long)  # Ensure labels are of␣
 ↪type torch.long

    dataset = TensorDataset(input_ids, attention_masks, labels)
    return DataLoader(dataset, batch_size=batch_size)

batch_size   = 100
train_loader = create_data_loader(train_data, tokenizer, 128, batch_size)
test_loader  = create_data_loader(test_data, tokenizer, 128, batch_size)
num_epochs   = 3

# Evaluate the model
device = torch.device('cuda:1')
model.to(device)
```

[ ]:
```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
```

```
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=3, bias=True)
)
```

[ ]:
```
# Evaluation
model.eval()
num_correct = 0
num_samples = 0
with torch.no_grad():
    for batch in tqdm(test_loader, desc='Evaluating', unit='batches'):
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device),␣
↪attention_mask.to(device), labels.to(device)

        outputs = model(input_ids, attention_mask=attention_mask)
        _, predicted = torch.max(outputs.logits, 1)
        num_correct += (predicted == labels).sum().item()
        num_samples += labels.size(0)

accuracy = num_correct / num_samples
print(f'Initial Accuracy: {accuracy*100}%')
```

```
Evaluating: 100%|      | 100/100 [01:03<00:00,  1.58batches/s]
```

```
Initial Accuracy: 43.14%
```

[ ]:
```
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)
for epoch in range(num_epochs):
    model.train()
    total_loss = 0.0
    for batch in tqdm(train_loader, desc=f'Epoch {epoch + 1}/{num_epochs}',␣
↪unit='batches'):
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device),␣
↪attention_mask.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        loss_function = torch.nn.CrossEntropyLoss()
        loss = loss_function(logits, labels)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    average_loss = total_loss / len(train_loader)
    print(f'Epoch {epoch + 1}/{num_epochs}, Average Loss: {average_loss}')
```

```
Epoch 1/3: 100%|      | 400/400 [12:02<00:00,  1.81s/batches]
Epoch 1/3, Average Loss: 0.5126290038041771
Epoch 2/3: 100%|      | 400/400 [12:02<00:00,  1.81s/batches]
Epoch 2/3, Average Loss: 0.17570224887225777
Epoch 3/3: 100%|      | 400/400 [12:02<00:00,  1.81s/batches]
Epoch 3/3, Average Loss: 0.1072592743858695
```

[ ]:
```
# Evaluation
model.eval()
num_correct = 0
num_samples = 0
with torch.no_grad():
    for batch in tqdm(test_loader, desc='Evaluating', unit='batches'):
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device),␣
↪attention_mask.to(device), labels.to(device)

        outputs = model(input_ids, attention_mask=attention_mask)
        _, predicted = torch.max(outputs.logits, 1)
```

```
        num_correct += (predicted == labels).sum().item()
        num_samples += labels.size(0)

accuracy = num_correct / num_samples
print(f'Accuracy: {accuracy*100}%')
```

```
Evaluating: 100%|      | 100/100 [01:06<00:00,  1.50batches/s]
```

```
Accuracy: 95.89%
```

[3]:
```
#Thus after training the model on the training data with 3 epochs, the model␣
↪performs better almost gives a 2x performance.
#Accuracy before train : 43.14 %
#Accuracy after train  : 95.89 %
```

[ ]:
```
# Initial Accuracy:
#     Before tuning we get 43.14%

# Fine Tuning:
#     Epochs: We run limited epochs.
#     Optimizer : AdamW optimizer with a learning rate of 0.00002
#     Loss Function : Cross Function Entropy (Best of classification tasks)

# Final Accuracy:
#     After tuning we get 95.89%

# Comparison:
#     There has been a substantial improvement in the accuracy after␣
↪fine-tuning the model.
#     This suggest that the model has learned to better generalize the task at␣
↪hand after being
#     trained on your dataset.
#     The average loss per epoch during training would give further information␣
↪regarding the
#     convergence of the model. Lower loss indicates better convergence.
#     Also, comparing the prediction and actual value gives us the accuracy␣
↪which is used as
#     mesaurement metric here.

# Inteerpretation:
#     The increase in accuracy from 43% to 95% demonstrates the effectiveness␣
↪of fine-tuning the pre-trained BERT
#     model on the specific task or dataset.
#     The significant improvement suggests that the model has learned useful␣
↪patterns from your
```

```
#     dataseet during the fine-tuning process, enabling it to make more␣
↪accurate predictions.


# Overall, the results indicate successful fine-tuning of the BERT model,␣
↪resulting in a highly accurate
# model for the task. Further given more epochs can be run on the model to␣
↪improve performance even better.

# Also, few possible reasons for using BERT,
# a.) BERT excels due to it transfomer architecture and state of the art NLP␣
↪performance.
# b.) Pre-trained on vast text data, BERT offers rich language representations.
# c.) Fine-tuning BERT on specific tasks unleashes its adaptibility and power.
```

[ ]:

```
```