ArdernClaire_HW9

December 5, 2023

1 NLP with Transformers

In the implementation part of assignment, we will evaluate Transformer models (DistilBERT and GPT) on text classification, e.g. sentiment analysis of IMDB reviews.

A large number of bonus points can be obtained by further evaluating Transformer models on:

- Text classification: sentiment analysis of Rotten Tomatoes reviews.
- Token classification: named entity recognition of CoNLL text.
- Zero-shot classification: text classification into arbitrary categories with textual names.

1.1 Google Colab

While the code in this notebook can be run locally on a powerful machine, it is highly recommended that the notebook is run on the GPU infrastructure availabe for free through Google's Colab. To load the notebook in Colab:

1. Point your browser to https://colab.research.google.com/
2. If a pop-up window opens, click on the Upload button and select this notebook file `code/skeleton/TransformersRule.ipynb`) from the homework folder.
3. Alternatively, in the notebook window menu clik File -> Open notebook and load the same notebook file.

1.2 HuggingFace Transformers

To complete that tasks in this assignment, we will use lightweight (DistilBERT) pretrained Transformer models and the corresponding high-level API from the open source repository of the HuggingFace platform. If you are interested in learning how to use the API and the datasets, it is recommended that you go through the relevant sections in the HuggingFace course on Transformers, which are linked from each portion in the assignment below.

While going through the HuggingFace course is highly recommended, it is not strictly necessary: this assignment can be completed by writing only Python code, without knowledge of the Transformers API. For this, you may be able to also reuse code that you have written in previous assignments.

You do not need a HuggingFace account to work on this assignment.

1.3 Write Your Name Here: Claire Ardern

2 Submission Instructions

2.1 Google Colab (recommended):

1. Click the File -> Save in the menu at the top of the Jupyter Notebook.
2. Please make sure to have entered your name above.
3. Select Edit -> Clear all outputs. This will clear all the outputs from all cells (but will keep the content of all cells).
4. Select Runtime -> Run all. This will run all the cells in order, and will take less than half hour for the sentiment analysis part.
5. Once you've rerun everything, select File -> Download -> Download .ipynb and download the notebook file showing the code and the output of all cells, and save it in the subfolder `code/complete/`.
6. Also save a PDF version of the notebook by selecting File -> Print, select Print as PDF, and Save it as `code/complete/TransformersRule.pdf`
7. Look at the PDF file and make sure all your solutions and outputs are there, displayed correctly.
8. Submit **both** your PDF and the notebook .ipynb file on Canvas. Make sure the PDF and notebook show the outputs of the training and evaluation procedures. Also upload any extra datasets that you used for bonus points by placing them in the `data/` folder (alternatively the notebook can show the web addresses from where the datasets are uploaded in Colab in your code).
9. Verify your Canvas submission contains the correct files by downloading them after posting them on Canvas.

2.2 Local computer (only for powerful machines):

1. Click the Save button at the top of the Jupyter Notebook.
2. Please make sure to have entered your name above.
3. Select Cell -> All Output -> Clear. This will clear all the outputs from all cells (but will keep the content of ll cells).
4. Select Cell -> Run All. This will run all the cells in order, and will take several minutes.
5. Once you've rerun everything, select File -> Download as -> PDF via LaTeX and download a PDF version *TransformersRule.pdf* showing the code and the output of all cells, and save it in the same folder that contains the notebook file *TransformersRule.ipynb*.
6. Look at the PDF file and make sure all your solutions are there, displayed correctly.
7. Submit **both** your PDF and notebook on Canvas. Make sure the PDF and notebook show the outputs of the training and evaluation procedures. Also upload any extra datasets that you used for bonus points by placing them in the `data/` folder.
8. Verify your Canvas submission contains the correct files by downloading them after posting them on Canvas.

3 Theory

3.1 Language Models and Perplexity (15p + 5p)

Consider a bigram language model LM according to which p(Time) = 0.03, p(flies | time) = 0.01, p(like | flies) = 0.04 p(an | like) = 0.05 p(arrow | an) = 0.1.

1. What is the *probability* of the sentence "Time flies like an arrow" according to this LM?
2. What is the *perplexity* that this LM obtains when evaluated on the sentence "Time flies like an arrow"?

Show your work.

YOUR SOLUTION HERE:

1. Probability("Time flies like an arrow") $= p(Time) \times p(flies|time) \times p(like|flies) \times p(an|like) \times p(arrow|an)$

$$= 0.03 \times 0.01 \times 0.04 \times 0.05 \times 0.01$$

$$= 6 \times 10^{-9}$$

2. Perplexity("Time flies like an arrow") $= \left(\frac{1}{p(Time)} \times \frac{1}{p(flies|time)} \times \frac{1}{p(like|flies)} \times \frac{1}{p(an|like)} \times \frac{1}{p(arrow|an)}\right)^{\frac{1}{5}}$

$$= \left(\frac{1}{0.03} \times \frac{1}{0.01} \times \frac{1}{0.04} \times \frac{1}{0.05} \times \frac{1}{0.01}\right)^{\frac{1}{5}}$$

$$= 44.0930$$

3.2 Bonus: Named Entity Recognition (15p + 10p)

Provide **BIO-style annotation** of the named entities (Person, Place, Organization, Date, or Product) in the following sentences:

1. The third mate was Flask, a native of Tisbury, in Martha's Vineyard.
2. Its official Nintendo announced today that they Will release the Nintendo 3DS in north America march 27.
3. Jessica Reif, a media analyst at Merrill Lynch & Co., said, "If they can get up and running with exclusive programming within six months, it doesn't set the venture back that far."

Also tag the examples above using the HuggingFace named entity recognition tagger (starter code provided in the implementation section on NER) and compare against your manual annotation: * Show your manually annotated named entities (labeled chunks) and the predicted named entities (labeled chunks). Do the predicted entities match your annotations?

The **BIO-style annotation** of named entities was explained on slides 5-7 of the lecture on manual annotation for NLP.

3.2.1 Manual Annotation:

P = Person Pl = Place Org = Organization D = Date Pr = Product

NOTE: Any tokens (words) not labeled can be assumed to have "O" tag. These are ommited to have more clear labeling.

1. The third mate was Flask_B-P, a native of Tisbury_B-Pl, in Martha's_B-Pl Vineyard_I-Pl.
2. Its official Nintendo_B-Org announced today that they Will release the Nintendo_B-Pr 3DS_I-Pr in north_B-Pl America_I-Pl march_B-D 27_I-D.
3. Jessica_B-P Reif_I-P, a media analyst at Merrill_B-Org Lynch_I-Org &_I-Org Co._I-Org, said, "If they can get up and running with exclusive programming within six months, it doesn't set the venture back that far."

3.2.2 HuggingFace Predicted Named Entities:

(Code for this can be seen in the NER bonus section below.)

1. [{'entity_group': 'PER', 'score': 0.99465436, 'word': 'Flask', 'start': 19, 'end': 24}, {'entity_group': 'LOC', 'score': 0.9023142, 'word': 'Tisbury', 'start': 38, 'end': 45}, {'entity_group': 'LOC', 'score': 0.96189433, 'word': 'Martha' s Vineyard', 'start': 50, 'end': 67}]
2. [{'entity_group': 'ORG', 'score': 0.9988274, 'word': 'Nintendo', 'start': 13, 'end': 21}, {'entity_group': 'MISC', 'score': 0.9949834, 'word': 'Nintendo 3DS', 'start': 65, 'end': 77}, {'entity_group': 'LOC', 'score': 0.99817383, 'word': 'America', 'start': 87, 'end': 94}]
3. [{'entity_group': 'PER', 'score': 0.9987101, 'word': 'Jessica Reif', 'start': 0, 'end': 12}, {'entity_group': 'ORG', 'score': 0.99618655, 'word': 'Merrill Lynch & Co.', 'start': 33, 'end': 52}]

It seems that the HuggingFace predicted NE mostly matches the manual annotations. However, there is one difference in the second sentence. The product "Nintendo 3DS" is marked by HuggingFace as "MISC." This does not label the Nintendo 3DS as a product, although it does recognize it as some named entity. This may be due to settings within the HuggingFace NER API. Also, the HuggingFace prediction for the second sentence does not recognize the date at the end of the sentence. This could be due to the format of the date (if it was given as 3/27/xxxx, this may have been recognized).

4 Implementation using HuggingFace Transformers

[1]:
```
# Install HuggingFace Transformers API modules.
!pip install transformers[sentencepiece]

# Install HuggingFace Datasets API modules.
!pip install datasets
```

```
Requirement already satisfied: transformers[sentencepiece] in
/usr/local/lib/python3.10/dist-packages (4.35.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from transformers[sentencepiece]) (3.13.1)
```

```
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in
/usr/local/lib/python3.10/dist-packages (from transformers[sentencepiece])
(0.19.4)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
packages (from transformers[sentencepiece]) (1.23.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers[sentencepiece])
(23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-
packages (from transformers[sentencepiece]) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers[sentencepiece])
(2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from transformers[sentencepiece]) (2.31.0)
Requirement already satisfied: tokenizers<0.19,>=0.14 in
/usr/local/lib/python3.10/dist-packages (from transformers[sentencepiece])
(0.15.0)
Requirement already satisfied: safetensors>=0.3.1 in
/usr/local/lib/python3.10/dist-packages (from transformers[sentencepiece])
(0.4.1)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-
packages (from transformers[sentencepiece]) (4.66.1)
Collecting sentencepiece!=0.1.92,>=0.1.91 (from transformers[sentencepiece])
  Downloading
sentencepiece-0.1.99-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(1.3 MB)
                           1.3/1.3 MB
6.2 MB/s eta 0:00:00
Requirement already satisfied: protobuf in /usr/local/lib/python3.10/dist-
packages (from transformers[sentencepiece]) (3.20.3)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.16.4->transformers[sentencepiece]) (2023.6.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.16.4->transformers[sentencepiece]) (4.5.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from
requests->transformers[sentencepiece]) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->transformers[sentencepiece]) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from
requests->transformers[sentencepiece]) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from
requests->transformers[sentencepiece]) (2023.11.17)
```

```
Installing collected packages: sentencepiece
Successfully installed sentencepiece-0.1.99
Collecting datasets
  Downloading datasets-2.15.0-py3-none-any.whl (521 kB)
                             521.2/521.2
kB 7.4 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from datasets) (1.23.5)
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.10/dist-
packages (from datasets) (9.0.0)
Collecting pyarrow-hotfix (from datasets)
  Downloading pyarrow_hotfix-0.6-py3-none-any.whl (7.9 kB)
Collecting dill<0.3.8,>=0.3.0 (from datasets)
  Downloading dill-0.3.7-py3-none-any.whl (115 kB)
                             115.3/115.3
kB 12.2 MB/s eta 0:00:00
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-
packages (from datasets) (1.5.3)
Requirement already satisfied: requests>=2.19.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist-
packages (from datasets) (4.66.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages
(from datasets) (3.4.1)
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.15-py310-none-any.whl (134 kB)
                             134.8/134.8
kB 13.7 MB/s eta 0:00:00
Requirement already satisfied: fsspec[http]<=2023.10.0,>=2023.1.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (2023.6.0)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-
packages (from datasets) (3.9.1)
Requirement already satisfied: huggingface-hub>=0.18.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (0.19.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from datasets) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-
packages (from datasets) (6.0.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-
packages (from aiohttp->datasets) (23.1.0)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (6.0.4)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-
packages (from aiohttp->datasets) (1.9.3)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.4.0)
```

```
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: async-timeout<5.0,>=4.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from huggingface-hub>=0.18.0->datasets) (3.13.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.18.0->datasets)
(4.5.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests>=2.19.0->datasets) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets)
(2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets)
(2023.11.17)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->datasets) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.1->pandas->datasets) (1.16.0)
Installing collected packages: pyarrow-hotfix, dill, multiprocess, datasets
Successfully installed datasets-2.15.0 dill-0.3.7 multiprocess-0.70.15 pyarrow-
hotfix-0.6
```

The most basic object in the Transformers library is the pipeline() function. It connects a model with its necessary preprocessing and postprocessing steps, allowing us to directly input any text and get an intelligible answer. More details are here:
https://huggingface.co/course/chapter1/3?fw=pt

[2]:
```
from transformers import pipeline

classifier = pipeline("sentiment-analysis")

# Process just one document. Note that, although the document string is split␣
 ↪on multiple lines, this is still just one string object.
output = classifier("The most merciful thing in the world, I think, is the␣
 ↪inability of the human mind to correlate all its contents. "
                    "We live on a placid island of ignorance in the midst of␣
 ↪black seas of infinity, and it was not meant that we should voyage far. "
                    "The sciences, each straining in its own direction, have␣
 ↪hitherto harmed us little; but some day the piecing together of "
```

```
                    "dissociated knowledge will open up such terrifying vistas␣
 ↪of reality, and of our frightful position therein, that we shall "
                    "either go mad from the revelation or flee from the light␣
 ↪into the peace and safety of a new dark age.")
print(output)

# Process a batch of documents.
output = classifier(["I've been waiting for a HuggingFace course my whole life.
 ↪",
                     "The spirit is willing, but the flesh is weak.",
                     "It might and does take until the last scene to do so, but␣
 ↪every plot thread is sewn-up, and there isn't a clue doled out along the way␣
 ↪that doesn't make sense or fit into the final puzzle. Its as close to␣
 ↪perfect as a screenplay can get.",
                     "The cast is great, the plot is quite ingenious and the␣
 ↪runtime is nothing too overbearing.",
                     "This film was filmed in France.",
                     "This film was filmed in Afghanistan."])
print(output)
```

```
No model was supplied, defaulted to distilbert-base-uncased-finetuned-
sst-2-english and revision af0f99b (https://huggingface.co/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': 0.6746031641960144}]
[{'label': 'POSITIVE', 'score': 0.9598048329353333}, {'label': 'NEGATIVE',
'score': 0.9832836389541626}, {'label': 'NEGATIVE', 'score':
0.9997618794441223}, {'label': 'POSITIVE', 'score': 0.9980607628822327},
{'label': 'POSITIVE', 'score': 0.9652073979377747}, {'label': 'NEGATIVE',
'score': 0.8350332379341125}]
```

5 Sentiment Analysis of IMDB Reviews

In this part of the assignment, we evaluate the performance of DistilBERT (the default Transformer) on the development portion of the IMDB reviews dataset, and compare it against the performance of Logistic Regression and RNNs from previous assignments.

5.1 Working with Datasets in Colab and HuggingFace

First, we need to load the IMDB Reviews dataset. HuggingFace already provide a large number of datasets in their Hub. Below we will be loading the IMDB dataset from an external source (http addresses on the course web page). More details on loading local and remote datasets are provided here: https://huggingface.co/course/chapter5/2?fw=pt

[3]:
```
from datasets import load_dataset

# Load the IMDB dataset. Only the development portion will be used later.
url_train = "https://webpages.charlotte.edu/rbunescu/courses/itcs4111/hw09/data/
 ↪imdb/train.txt"
url_test = "https://webpages.charlotte.edu/rbunescu/courses/itcs4111/hw09/data/
 ↪imdb/test.txt"
url_dev = "https://webpages.charlotte.edu/rbunescu/courses/itcs4111/hw09/data/
 ↪imdb/dev.txt"

data_files = {"train": url_train, "test": url_test, "dev": url_dev}
dataset = load_dataset('text', data_files = data_files)
dataset
```

```
Downloading data files:   0%|          | 0/3 [00:00<?, ?it/s]

Downloading data:   0%|          | 0.00/2.00M [00:00<?, ?B/s]

Downloading data:   0%|          | 0.00/1.88M [00:00<?, ?B/s]

Downloading data:   0%|          | 0.00/1.94M [00:00<?, ?B/s]

Extracting data files:   0%|          | 0/3 [00:00<?, ?it/s]

Generating train split: 0 examples [00:00, ? examples/s]

Generating test split: 0 examples [00:00, ? examples/s]

Generating dev split: 0 examples [00:00, ? examples/s]
```

[3]:
```
DatasetDict({
    train: Dataset({
        features: ['text'],
        num_rows: 1500
    })
    test: Dataset({
        features: ['text'],
        num_rows: 1500
    })
    dev: Dataset({
        features: ['text'],
        num_rows: 1500
    })
})
```

The `dataset` object is a `DatasetDict` dictionary mapping to the train, test, and dev objects of type `Dataset`. Since these were created from text files, each line in the file leads to a string example in the `Dataset` object. There are `num_rows` examples in total that can be retrieved through iterators or the usual indexing mechanism, as shown below. More powerfull mechanisms for working with datasets are shown here: https://huggingface.co/course/chapter5/3?fw=pt

[4]:
```
# Print an example from the training dataset. Note how it is represented as a␣
 ↪dictionary, mapping one feature named 'text' to its string value.
print(dataset['train'][5])

# To get just the example string itself, the way it was stored in the file.
print(dataset['train'][5]['text'])

# Since we evaluate only on the development data, delete the train and test␣
 ↪portions. This will reduce the memory footprint in Colab.
del dataset['train']
del dataset['test']
dataset
```

```
{'text': 'pos Grey Gardens is shocking, amusing, sad and mesmerizing. I watched
in amazement as Ediths Jr. and Sr. bickered and performed while reminiscing of
their past. Their existence in a dilapidated mansion, (which they had not left
for more than fifteen years) is both a comedy and a tragedy. This is a film you
will not soon forget.'}
pos Grey Gardens is shocking, amusing, sad and mesmerizing. I watched in
amazement as Ediths Jr. and Sr. bickered and performed while reminiscing of
their past. Their existence in a dilapidated mansion, (which they had not left
for more than fifteen years) is both a comedy and a tragedy. This is a film you
will not soon forget.
```

[4]:
```
DatasetDict({
    dev: Dataset({
        features: ['text'],
        num_rows: 1500
    })
})
```

5.1.1 Process the development `Dataset` (20p)

Write a function `read_examples(ds)` that takes a `Dataset` object as input and returns a tuple containg the list of `labels` and the corresponding list of `reviews`. A label should have value 1 if the review was labeled as positive and 0 otherwise.

[5]:
```
def read_examples(ds):
  labels = []
  reviews = []

  # YOUR CODE HERE
```

```
  for i in range(len(ds)):
      line = ds[i]['text']
      [label, text] = line.strip().split(' ', maxsplit = 1)
      reviews.append(text)
      if label == 'pos':
        labels.append(1)
      else:
        labels.append(0)


  return reviews, labels

reviews, labels = read_examples(dataset['dev'])

# Since the top half of the reviews in the file are labeled as 'pos' and the␣
 ↪bottom half are labeled as 'neg', the lines below should display
# [1, 1, 1, 1, 1]
# [0, 0, 0, 0, 0]
print(labels[:5])
print(labels[-5:])
```

```
[1, 1, 1, 1, 1]
[0, 0, 0, 0, 0]
```

5.2 Sentiment Analysis using DistilBERT Transformer

In this section, we apply the default Transformer model for sentiment classification.

Simply calling the `classifier` on the list of `reviews` will not work because some reviews are longer than the maximum length of 512 that can be accomodated by DistilBERT. Even if the reviews were to be truncated, Colab may run out of memory, as the Transformer classifier tries to process all examples in parallel.

[7]:
```
# Attempt to classify all reviews, store labels in 'predictions'.
predictions = classifier(reviews)
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-7-060645753147> in <cell line: 2>()
      1 # Attempt to classify all reviews, store labels in 'predictions'.
----> 2 predictions = classifier(reviews)

/usr/local/lib/python3.10/dist-packages/transformers/pipelines/
 ↪text_classification.py in __call__(self, *args, **kwargs)
    154             If `top_k` is used, one such dictionary is returned per␣
 ↪label.
    155         """
--> 156         result = super().__call__(*args, **kwargs)
```

```
    157         # TODO try and retrieve it in a nicer way from␣
 ↪_sanitize_parameters.
    158         _legacy = "top_k" not in kwargs

/usr/local/lib/python3.10/dist-packages/transformers/pipelines/base.py in␣
 ↪__call__(self, inputs, num_workers, batch_size, *args, **kwargs)
   1119                     inputs, num_workers, batch_size, preprocess_params,␣
 ↪forward_params, postprocess_params
   1120                 )
-> 1121                 outputs = list(final_iterator)
   1122                 return outputs
   1123             else:

/usr/local/lib/python3.10/dist-packages/transformers/pipelines/pt_utils.py in␣
 ↪__next__(self)
    122
    123         # We're out of items within a batch
--> 124         item = next(self.iterator)
    125         processed = self.infer(item, **self.params)
    126         # We now have a batch of "inferred things".

/usr/local/lib/python3.10/dist-packages/transformers/pipelines/pt_utils.py in␣
 ↪__next__(self)
    123         # We're out of items within a batch
    124         item = next(self.iterator)
--> 125         processed = self.infer(item, **self.params)
    126         # We now have a batch of "inferred things".
    127         if self.loader_batch_size is not None:

/usr/local/lib/python3.10/dist-packages/transformers/pipelines/base.py in␣
 ↪forward(self, model_inputs, **forward_params)
   1044                 with inference_context():
   1045                     model_inputs = self.
 ↪_ensure_tensor_on_device(model_inputs, device=self.device)
-> 1046                     model_outputs = self._forward(model_inputs,␣
 ↪**forward_params)
   1047                     model_outputs = self.
 ↪_ensure_tensor_on_device(model_outputs, device=torch.device("cpu"))
   1048             else:

/usr/local/lib/python3.10/dist-packages/transformers/pipelines/
 ↪text_classification.py in _forward(self, model_inputs)
    185         if "use_cache" in inspect.signature(model_forward).parameters.
 ↪keys():
    186             model_inputs["use_cache"] = False
--> 187         return self.model(**model_inputs)
    188
```

```
    189     def postprocess(self, model_outputs, function_to_apply=None,␣
 ↪top_k=1, _legacy=True):

/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py in␣
 ↪_wrapped_call_impl(self, *args, **kwargs)
   1516             return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
   1517         else:
-> 1518             return self._call_impl(*args, **kwargs)
   1519
   1520     def _call_impl(self, *args, **kwargs):

/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py in␣
 ↪_call_impl(self, *args, **kwargs)
   1525                 or _global_backward_pre_hooks or _global_backward_hooks
   1526                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1527             return forward_call(*args, **kwargs)
   1528
   1529         try:

/usr/local/lib/python3.10/dist-packages/transformers/models/distilbert/
 ↪modeling_distilbert.py in forward(self, input_ids, attention_mask, head_mask,␣
 ↪inputs_embeds, labels, output_attentions, output_hidden_states, return_dict)
    777         return_dict = return_dict if return_dict is not None else self.
 ↪config.use_return_dict
    778
--> 779         distilbert_output = self.distilbert(
    780             input_ids=input_ids,
    781             attention_mask=attention_mask,

/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py in␣
 ↪_wrapped_call_impl(self, *args, **kwargs)
   1516             return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
   1517         else:
-> 1518             return self._call_impl(*args, **kwargs)
   1519
   1520     def _call_impl(self, *args, **kwargs):

/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py in␣
 ↪_call_impl(self, *args, **kwargs)
   1525                 or _global_backward_pre_hooks or _global_backward_hooks
   1526                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1527             return forward_call(*args, **kwargs)
   1528
   1529         try:
```

```
/usr/local/lib/python3.10/dist-packages/transformers/models/distilbert/
 ↪modeling_distilbert.py in forward(self, input_ids, attention_mask, head_mask,␣
 ↪inputs_embeds, output_attentions, output_hidden_states, return_dict)
    595         head_mask = self.get_head_mask(head_mask, self.config.
 ↪num_hidden_layers)
    596
--> 597         embeddings = self.embeddings(input_ids, inputs_embeds)  # (bs,␣
 ↪seq_length, dim)
    598
    599         return self.transformer(

/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py in␣
 ↪_wrapped_call_impl(self, *args, **kwargs)
   1516             return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
   1517         else:
-> 1518             return self._call_impl(*args, **kwargs)
   1519
   1520     def _call_impl(self, *args, **kwargs):

/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py in␣
 ↪_call_impl(self, *args, **kwargs)
   1525                 or _global_backward_pre_hooks or _global_backward_hooks
   1526                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1527             return forward_call(*args, **kwargs)
   1528
   1529         try:

/usr/local/lib/python3.10/dist-packages/transformers/models/distilbert/
 ↪modeling_distilbert.py in forward(self, input_ids, input_embeds)
    133         position_embeddings = self.position_embeddings(position_ids)  #␣
 ↪(bs, max_seq_length, dim)
    134
--> 135         embeddings = input_embeds + position_embeddings  # (bs,␣
 ↪max_seq_length, dim)
    136         embeddings = self.LayerNorm(embeddings)  # (bs, max_seq_length,␣
 ↪dim)
    137         embeddings = self.dropout(embeddings)  # (bs, max_seq_length,␣
 ↪dim)

RuntimeError: The size of tensor a (517) must match the size of tensor b (512)␣
 ↪at non-singleton dimension 1
```

5.2.1 Delving deeper into the NLP pipeline

While HuggingFace may provide API for truncating and batching examples, below we show how to work with the NLP pipeline, which will give us more control over the inputs and outputs. More

details are provided in the Behind the pipeline course section.

5.2.2 Subsample examples for quick evaluations (20p)

Classifying the entire dataset of 1,500 examples can be time consuming. For debugging purposes, it is useful to evaluate on a subset of examples. Write a function `sample_dataset(reviews, labels, k)` that extracts a subset of `sreviews` and their associated `slabels` containing just the top k followed by the bottom k examples in the dataset that is provided as input, for a total of 2k examples. We do this so that we have an equal number of positive (top k) and negative (bottom k) examples.

[29]:
```
def sample_dataset(reviews, labels, k):
  sreviews, slabels = [], []

  # YOUR CODE HERE
  sreviews.extend(reviews[:k])
  sreviews.extend(reviews[-k:])

  slabels.extend(labels[:k])
  slabels.extend(labels[-k:])

  return sreviews, slabels
```

5.2.3 Pipeline: From Tokenizer to Model to Post Processing

This is the main code, where first the reviews are Tokenized, then classified with the Model, followed by a slight Post Processing of the predicted labels. For more on the pipeline, see the Putting it all together section in the course and the previous sections in the Using Transformers chapter.

If we ran the sentiment analysis model on all 1,500 reviews directly, Colab would run out of memory (try it). Therefore, the code below runs the model on batches of 10 reviews at a time and accumulates the predicted labels in the `predictions` list.

[8]:
```
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification

# A machine learning model, like DistilBERT, can be trained using many␣
 ↪hyper-parameters configurations,
# such as learning rate, number of training epochs, preprocessing of its data,␣
 ↪finetuning, etc. A checkpoint
# corresponds to a particular training setting. All checkpoints are listed and␣
 ↪explained on the Hub page.
checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"

# Extract the Tokenizer that was used for the training data. It is important␣
 ↪that the same tokenizer is used on the test data.
# https://huggingface.co/course/chapter2/4?fw=pt
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
```

```
# Extract the sequence classification model associated with the checkpoint.
# https://huggingface.co/course/chapter2/3?fw=pt
model = AutoModelForSequenceClassification.from_pretrained(checkpoint)

# Print the numerical IDs for the two labels, to verify 0 means negative, 1␣
 ↪means positive.
print(model.config.id2label)

# By default run evaluations on all examples.
sreviews, slabels = reviews, labels

# When set to True, experiments are run only on 2 * 50 examples.
# Change to False for running on all 1,500 examples for the final submission.
debug = False
if debug:
  sreviews, slabels = sample_dataset(reviews, labels, 50)

# Position for the current batch.
position = 0
# Accumulate batch predicted labels here.
predictions = []

while position < len(sreviews):
  # Evaluate on a batch of 10 reviews at a time.
  batch = sreviews[position: position + 10]

  # Will pad the sequences up to the max length in the dataset.
  # Will also truncate the sequences that are longer than the model max length␣
 ↪(512 for BERT or DistilBERT).
  tokens = tokenizer(batch, padding = "longest", truncation = True,␣
 ↪return_tensors = "pt")

  output = model(**tokens)
  predictions += list(map(lambda logit: int(logit[0] < logit[1]), output.
 ↪logits))

  position += 10
  print('Processed', position, 'reviews.')
```

```
{0: 'NEGATIVE', 1: 'POSITIVE'}
Processed 10 reviews.
Processed 20 reviews.
Processed 30 reviews.
Processed 40 reviews.
Processed 50 reviews.
Processed 60 reviews.
```

```
Processed 70 reviews.
Processed 80 reviews.
Processed 90 reviews.
Processed 100 reviews.
Processed 110 reviews.
Processed 120 reviews.
Processed 130 reviews.
Processed 140 reviews.
Processed 150 reviews.
Processed 160 reviews.
Processed 170 reviews.
Processed 180 reviews.
Processed 190 reviews.
Processed 200 reviews.
Processed 210 reviews.
Processed 220 reviews.
Processed 230 reviews.
Processed 240 reviews.
Processed 250 reviews.
Processed 260 reviews.
Processed 270 reviews.
Processed 280 reviews.
Processed 290 reviews.
Processed 300 reviews.
Processed 310 reviews.
Processed 320 reviews.
Processed 330 reviews.
Processed 340 reviews.
Processed 350 reviews.
Processed 360 reviews.
Processed 370 reviews.
Processed 380 reviews.
Processed 390 reviews.
Processed 400 reviews.
Processed 410 reviews.
Processed 420 reviews.
Processed 430 reviews.
Processed 440 reviews.
Processed 450 reviews.
Processed 460 reviews.
Processed 470 reviews.
Processed 480 reviews.
Processed 490 reviews.
Processed 500 reviews.
Processed 510 reviews.
Processed 520 reviews.
Processed 530 reviews.
Processed 540 reviews.
```

```
Processed 550 reviews.
Processed 560 reviews.
Processed 570 reviews.
Processed 580 reviews.
Processed 590 reviews.
Processed 600 reviews.
Processed 610 reviews.
Processed 620 reviews.
Processed 630 reviews.
Processed 640 reviews.
Processed 650 reviews.
Processed 660 reviews.
Processed 670 reviews.
Processed 680 reviews.
Processed 690 reviews.
Processed 700 reviews.
Processed 710 reviews.
Processed 720 reviews.
Processed 730 reviews.
Processed 740 reviews.
Processed 750 reviews.
Processed 760 reviews.
Processed 770 reviews.
Processed 780 reviews.
Processed 790 reviews.
Processed 800 reviews.
Processed 810 reviews.
Processed 820 reviews.
Processed 830 reviews.
Processed 840 reviews.
Processed 850 reviews.
Processed 860 reviews.
Processed 870 reviews.
Processed 880 reviews.
Processed 890 reviews.
Processed 900 reviews.
Processed 910 reviews.
Processed 920 reviews.
Processed 930 reviews.
Processed 940 reviews.
Processed 950 reviews.
Processed 960 reviews.
Processed 970 reviews.
Processed 980 reviews.
Processed 990 reviews.
Processed 1000 reviews.
Processed 1010 reviews.
Processed 1020 reviews.
```

```
Processed 1030 reviews.
Processed 1040 reviews.
Processed 1050 reviews.
Processed 1060 reviews.
Processed 1070 reviews.
Processed 1080 reviews.
Processed 1090 reviews.
Processed 1100 reviews.
Processed 1110 reviews.
Processed 1120 reviews.
Processed 1130 reviews.
Processed 1140 reviews.
Processed 1150 reviews.
Processed 1160 reviews.
Processed 1170 reviews.
Processed 1180 reviews.
Processed 1190 reviews.
Processed 1200 reviews.
Processed 1210 reviews.
Processed 1220 reviews.
Processed 1230 reviews.
Processed 1240 reviews.
Processed 1250 reviews.
Processed 1260 reviews.
Processed 1270 reviews.
Processed 1280 reviews.
Processed 1290 reviews.
Processed 1300 reviews.
Processed 1310 reviews.
Processed 1320 reviews.
Processed 1330 reviews.
Processed 1340 reviews.
Processed 1350 reviews.
Processed 1360 reviews.
Processed 1370 reviews.
Processed 1380 reviews.
Processed 1390 reviews.
Processed 1400 reviews.
Processed 1410 reviews.
Processed 1420 reviews.
Processed 1430 reviews.
Processed 1440 reviews.
Processed 1450 reviews.
Processed 1460 reviews.
Processed 1470 reviews.
Processed 1480 reviews.
Processed 1490 reviews.
Processed 1500 reviews.
```

5.2.4 Evaluate the DistilBERT Model Accuracy on the IMDB Development Dataset (10p)

Write a function `compute_accuracy(labels, predictions)` that calculates the accuracy of the model `predicted` labels with respect to the ground truth `labels`.

[9]:
```
def compute_accuracy(labels, predictions):
  accuracy = 0 # YOUR CODE HERE. CAN YOU DO IT IN ONE LINE?
  for i in range(len(labels)):
    if labels[i] == predictions[i]:
      accuracy += 1
  accuracy = accuracy / len(labels)
  return accuracy


print('Accuracy = ', compute_accuracy(slabels, predictions))
```

```
Accuracy =  0.8933333333333333
```

5.3 Analysis of Results (35p)

1. (5p) Compare the performance of DistilBERT in IMDB with Logistic Regression and RNN performance from previous assignments.
2. Error analysis:

- (15p) Look at a sample of reviews that were misclassified by DistilBERT, try to determine if there is a common type of errors that it makes.
- (15p) Look at a sample of reviews that were misclassified by Logistic Regression but corectly classified by DistilBERT, elaborate on why you think DistilBERT was able to do better.
- [**(15p) Bonus**] Elaborate on how the model performance could be improved.
- [**(50p) Bigger Bonus**] Write code that leads to improved performance.

The accuracy achieved by the DistilBERT model on the development data is around 89%. This is better than the 82% accuracy that was achieved by the LR model in Homework 5. In addition, this is a drastic improvement on the accuracy score of around 78% that was achieved by the RNN model in Homework 8. The fact that the LR model performs better than the RNN model is interesting, as the RNN model is much more sophisticated. However, this fact has already been discussed in Homework 8. The more important fact here is that the DistilBERT model has outperformed all of the models created in previous homework assignments. This is likely due to the additional context and more intricate embeddings that are offered with the use of transformers. A tabular representation of the model performances is shown below.

Assignment	Homework 9	Homework 5	Homework 8
Model	DistilBERT	Logistic Regression	RNN
Test Accuracy	89.33%	82.2%	77.86%

[10]:
```
count = 0
for i in range(len(slabels)):
```

```
    if slabels[i] != predictions[i] and count < 20:
        print("Test case ", i, " with class ", slabels[i], " was incorrectly␣
↪classified as ", predictions[i])
        count += 1
```

```
Test case  0  with class  1  was incorrectly classified as  0
Test case  5  with class  1  was incorrectly classified as  0
Test case  12  with class  1  was incorrectly classified as  0
Test case  18  with class  1  was incorrectly classified as  0
Test case  27  with class  1  was incorrectly classified as  0
Test case  56  with class  1  was incorrectly classified as  0
Test case  57  with class  1  was incorrectly classified as  0
Test case  61  with class  1  was incorrectly classified as  0
Test case  63  with class  1  was incorrectly classified as  0
Test case  68  with class  1  was incorrectly classified as  0
Test case  71  with class  1  was incorrectly classified as  0
Test case  73  with class  1  was incorrectly classified as  0
Test case  82  with class  1  was incorrectly classified as  0
Test case  108  with class  1  was incorrectly classified as  0
Test case  112  with class  1  was incorrectly classified as  0
Test case  114  with class  1  was incorrectly classified as  0
Test case  121  with class  1  was incorrectly classified as  0
Test case  128  with class  1  was incorrectly classified as  0
Test case  130  with class  1  was incorrectly classified as  0
Test case  147  with class  1  was incorrectly classified as  0
```

[11]:
```
print("Data:", sreviews[0], '\n')
print("Correct Class: ", slabels[0], '\n')
print("Predicted Class: ", predictions[0], '\n')
```

```
Data: So Dark The Night poses a tough challenge: It's very hard to write about
it in any detail without ruining it for those who haven't yet seen it. Since it
remains quite obscure, that includes just about everybody. The movie will strike
those familiar with its director Joseph H. Lewis' better known titles in the
noir cycle  Gun Crazy, The Big Combo, even My Name Is Julia Ross, which in its
brevity it resembles  as an odd choice.<br /><br />For starters, the bucolic
French countryside serves as its setting. Steven Geray, a middle-aged detective
with the Surété in Paris, sets out for a vacation in the village of Ste. Margot
(or maybe Margaux). Quite unexpectedly, he finds himself falling in love with
the inkeepers' daughter (Micheline Cheirel), even though she's betrothed to a
rough-hewn local farmer. But the siren song of life in Paris is hard to resist,
so she agrees to marry him, despite the disparity in their ages, which
inevitably becomes the talk of the town. <br /><br />But on the night of their
engagement party, she fails to return to the inn. Soon, a hunchback finds her
body by the river. Her jealous, jilted lover is the logical suspect, but he,
too, is found dead. Then anonymous notes threaten more deaths, which come to
pass. For the first time in his career, the bereaved Geray finds himself
```

stumped…

A particularly weak script all but does the movie in; it plays like bad Cornell Woolrich crossed with The Murder of Roger Ackroyd. But Lewis does this creaky vehicle proud. He takes his time near the beginning, but then the story and the storytelling gain momentum (alas, just about the time the script breaks an axle). Burnett Guffey lighted and photographed the film, with an intriguing leitmotif of peering out of and peeping into windows; there's also an effective score by Hugo Friedhofer, who supplied aural menace to many noirs. A good deal of talent has been lavished on So Dark The Night, but at the end it boils down to not much more than a gimmick and not a very good gimmick at that. It's a one-trick pony of a movie.

Correct Class: 1

Predicted Class: 0

[12]:
```
print("Data:", sreviews[5], '\n')
print("Correct Class: ", slabels[5], '\n')
print("Predicted Class: ", predictions[5], '\n')
```

Data: I love the other reviews of this movie. They mirror my attitude. I am a 70's sort of guy, minus disco and "Star Wars" childishness. There was nothing great about this movie, except for a chase scene. That is why it was good, because it was tough, basic and economical. Roy Scheider carried the movie, which was based on the crew, the 7 Ups, that backed up Gene Hackman in the "French Connection". The people in it were believable and average, who burned themselves pouring coffee, showed fear in chase scene and almost lost it after a close call crash.

Maybe it would be easier to tell you what it lacked. There was no fancy weapons, just basic revolvers and crude sawed off shotguns. There was no tough guy philosophizing, ala Tarantino. There was no kung fu or samurai nonsense and no fancy trick shooting either. There was no clever guy who carries out some complicated scheme based on hundreds of things going just the way he planned including everyone else's reactions. The criminals were bad guys but they didn't shoot people for the hell of it. As a matter of fact, there was a body count of just three. something that the average movie these days would pass in the opening credits. It could be a G movie today! No bus load of orphan school children were kidnapped nor were terrorists threatening to kill half of the city. There were no high tech hijinks, nor were the crimes themselves very moving or ingenious, the highest tech thing I saw was a touch tone ATT wall phone. It had no subplots or amusing character developments. Also, no sex or women, except for one mobster's wife who did some screaming as the Buddy our hero had her menaced.

It was some little undertaker who exploited his connections with the local mob and the police to kidnap local mobsters for some easy payoffs. The undertakers. Vito, was played by Tony Lo Bianco who did a great job, as good as Roy Schneider, Buddy the head of 7 Ups cop, whom he informed and exploited. What ever happened to Tony Lo Bianco, he seemed like a Pacino shoe in, good looking and talented? What it did have was a great NYC backdrop to a simple crime story. Locations that were bleak

and dehumanizing without being a sociological study. It had a simple plot that involved this kidnapping scheme where one of Buddy's cop got accidentally involved, literally accidentally dragged in then accidentally shot dead. Since Buddy and his 7 ups are a hot dogs unit, both the NYPD Brass and mobsters thought he was involved, since the kidnappers masqueraded as plain clothes cops to lure the mobsters into compliance. Obviously the mobsters figured they had lawyers and rights to protect them from normal police. Even the mobsters were plain, old and ugly, no Godfather royalty or Soprano hipness here.

It is a good basic movie with a standout chase scene between two 70's d Pontiacs. Even the cars were plain and economical, not even a GTO or a Trans Am, like the acting and the story. In the days of Batman uber-hype or "24" levels of intensity doomsday scenarios, this movie reminds us that less is better. It should be shown to movie screen writers and directors as a caveat not to dazzle, amuse then ultimately insult us with stunts, gadgets and clown psychotic behavior galore.

Correct Class: 1

Predicted Class: 0

[13]:
```
print("Data:", sreviews[12], '\n')
print("Correct Class: ", slabels[12], '\n')
print("Predicted Class: ", predictions[12], '\n')
```

Data: Another Asian horror movie packed with intense, and creepy moments. Another Asian horror trademark is the complexity of the plot, which is here as well. MAJOR SPOILER WARNING!

The movie starts pretty simple - two sisters go to live with their dad and stepmother after being put in a mental institution after their mother hanged herself. The sisters seem very hostile towards their mother - especially the elder one - and they seem to ignore their father. All goes smoothly until the mother locks the young sister in the wardrobe and the elder sister tells her father. Then it hits you, "your sister has been dead for years now" It turns out the older sister is still not recovered from the death of her mother and what we didn't know is that the wardrobe the mother was hanged in fell on the younger sister and killed her as well.As for the stepmother she is the alter ego of the older sister - revealed when the stepmother (actually the sister's alter ego) is sitting on a couch when the real stepmother walks in! I hope it has been made clearer for confused Asian horror fans out there.

Finally - my favourite scene is the scene where the father invites friends over for dinner and one of the friends starts to choke which erupts into a panic attack. Very creepy! 7 out of 10

Correct Class: 1

Predicted Class: 0

It seems that the DistilBERT model often misclassifies in cases where there is more of a desription of the movie rather than an opinion. In the first two examples of misclassification shown above, the DistilBERT model classified these reviews as having a negative sentiment. In all honesty, I would have made the same decision. Although these reviews both briefly state that the movie was "good," they spend more time pointing out flaws or missing pieces in the movie. So, although they ultimately leave a positive review on the movie, they pair this with several negative comments. For this reason, it makes sense that the model would classify these reviews as negative.

In the last example case shown above, the same general idea holds true though it is less obvious. This review shows a positive sentiment tied to the complexity of the plot. However, most of the content of the reivew is simply given to explain this complex plot. As it is a horror movie, there is no shortage of negative terminology. In addition, the "positive" sentiment that is given in this movie review is given in the form "very creepy!" As this is a review for a horror movie, that would be a positive remark. However, in general, "creepy" would not be seen as a positive word. So, it makes sense that the model would classify this review as negative.

To take a closer look at the improved performance of the DistilBERT model (89%) over the Logistic Regression model (82%) in Homework 5, we can look at specific cases of misclassification by the LR model that DistilBERT has correctly classified. However, since we used the test set to evaluate the LR model in Homework 5, we must now run the DistilBERT model on the same portion of data. We will use the example cases of misclassification that were shown in Homework 5: indexes 48 and 99.

[14]:
```
# Load the IMDB dataset. Only the development portion will be used later.
url_train = "https://webpages.charlotte.edu/rbunescu/courses/itcs4111/hw09/data/
 ↪imdb/train.txt"
url_test = "https://webpages.charlotte.edu/rbunescu/courses/itcs4111/hw09/data/
 ↪imdb/test.txt"
url_dev = "https://webpages.charlotte.edu/rbunescu/courses/itcs4111/hw09/data/
 ↪imdb/dev.txt"

data_files = {"train": url_train, "test": url_test, "dev": url_dev}
dataset = load_dataset('text', data_files = data_files)
del dataset['train']
del dataset['dev']
dataset

reviews, labels = read_examples(dataset['test'])
```

[15]:
```
checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"

tokenizer = AutoTokenizer.from_pretrained(checkpoint)

model = AutoModelForSequenceClassification.from_pretrained(checkpoint)

# Print the numerical IDs for the two labels, to verify 0 means negative, 1␣
 ↪means positive.
print(model.config.id2label)
```

```
# Run evaluations on all examples.
sreviews, slabels = reviews, labels

# Position for the current batch.
position = 0
# Accumulate batch predicted labels here.
predictions = []

while position < len(sreviews):
  # Evaluate on a batch of 10 reviews at a time.
  batch = sreviews[position: position + 10]

  # Will pad the sequences up to the max length in the dataset.
  # Will also truncate the sequences that are longer than the model max length␣
 ↪(512 for BERT or DistilBERT).
  tokens = tokenizer(batch, padding = "longest", truncation = True,␣
 ↪return_tensors = "pt")

  output = model(**tokens)
  predictions += list(map(lambda logit: int(logit[0] < logit[1]), output.
 ↪logits))

  position += 10
  print('Processed', position, 'reviews.')
```

```
{0: 'NEGATIVE', 1: 'POSITIVE'}
Processed 10 reviews.
Processed 20 reviews.
Processed 30 reviews.
Processed 40 reviews.
Processed 50 reviews.
Processed 60 reviews.
Processed 70 reviews.
Processed 80 reviews.
Processed 90 reviews.
Processed 100 reviews.
Processed 110 reviews.
Processed 120 reviews.
Processed 130 reviews.
Processed 140 reviews.
Processed 150 reviews.
Processed 160 reviews.
Processed 170 reviews.
Processed 180 reviews.
Processed 190 reviews.
Processed 200 reviews.
```

```
Processed 210 reviews.
Processed 220 reviews.
Processed 230 reviews.
Processed 240 reviews.
Processed 250 reviews.
Processed 260 reviews.
Processed 270 reviews.
Processed 280 reviews.
Processed 290 reviews.
Processed 300 reviews.
Processed 310 reviews.
Processed 320 reviews.
Processed 330 reviews.
Processed 340 reviews.
Processed 350 reviews.
Processed 360 reviews.
Processed 370 reviews.
Processed 380 reviews.
Processed 390 reviews.
Processed 400 reviews.
Processed 410 reviews.
Processed 420 reviews.
Processed 430 reviews.
Processed 440 reviews.
Processed 450 reviews.
Processed 460 reviews.
Processed 470 reviews.
Processed 480 reviews.
Processed 490 reviews.
Processed 500 reviews.
Processed 510 reviews.
Processed 520 reviews.
Processed 530 reviews.
Processed 540 reviews.
Processed 550 reviews.
Processed 560 reviews.
Processed 570 reviews.
Processed 580 reviews.
Processed 590 reviews.
Processed 600 reviews.
Processed 610 reviews.
Processed 620 reviews.
Processed 630 reviews.
Processed 640 reviews.
Processed 650 reviews.
Processed 660 reviews.
Processed 670 reviews.
Processed 680 reviews.
```

```
Processed 690 reviews.
Processed 700 reviews.
Processed 710 reviews.
Processed 720 reviews.
Processed 730 reviews.
Processed 740 reviews.
Processed 750 reviews.
Processed 760 reviews.
Processed 770 reviews.
Processed 780 reviews.
Processed 790 reviews.
Processed 800 reviews.
Processed 810 reviews.
Processed 820 reviews.
Processed 830 reviews.
Processed 840 reviews.
Processed 850 reviews.
Processed 860 reviews.
Processed 870 reviews.
Processed 880 reviews.
Processed 890 reviews.
Processed 900 reviews.
Processed 910 reviews.
Processed 920 reviews.
Processed 930 reviews.
Processed 940 reviews.
Processed 950 reviews.
Processed 960 reviews.
Processed 970 reviews.
Processed 980 reviews.
Processed 990 reviews.
Processed 1000 reviews.
Processed 1010 reviews.
Processed 1020 reviews.
Processed 1030 reviews.
Processed 1040 reviews.
Processed 1050 reviews.
Processed 1060 reviews.
Processed 1070 reviews.
Processed 1080 reviews.
Processed 1090 reviews.
Processed 1100 reviews.
Processed 1110 reviews.
Processed 1120 reviews.
Processed 1130 reviews.
Processed 1140 reviews.
Processed 1150 reviews.
Processed 1160 reviews.
```

```
Processed 1170 reviews.
Processed 1180 reviews.
Processed 1190 reviews.
Processed 1200 reviews.
Processed 1210 reviews.
Processed 1220 reviews.
Processed 1230 reviews.
Processed 1240 reviews.
Processed 1250 reviews.
Processed 1260 reviews.
Processed 1270 reviews.
Processed 1280 reviews.
Processed 1290 reviews.
Processed 1300 reviews.
Processed 1310 reviews.
Processed 1320 reviews.
Processed 1330 reviews.
Processed 1340 reviews.
Processed 1350 reviews.
Processed 1360 reviews.
Processed 1370 reviews.
Processed 1380 reviews.
Processed 1390 reviews.
Processed 1400 reviews.
Processed 1410 reviews.
Processed 1420 reviews.
Processed 1430 reviews.
Processed 1440 reviews.
Processed 1450 reviews.
Processed 1460 reviews.
Processed 1470 reviews.
Processed 1480 reviews.
Processed 1490 reviews.
Processed 1500 reviews.
```

[16]:
```
print("Data:", sreviews[48], '\n')
print("Correct Class: ", slabels[48], '\n')
print("Predicted Class: ", predictions[48], '\n')
```

```
Data: "Hit and Run" is a shattering story starring the always wonderful Margaret
Colin as a society lady who "has it all" until she hits a child with her car and
leaves the scene. Hence the title. The tragedy is that she goes to call for help
and returns, but is frightened away by angry passers-by who think the hitter
abandoned the scene. This was made in the days when not everyone had a cell
phone or there wouldn't be a story.<br /><br />Colin's guilt and anguish are
palpable and cause her to act so strangely that a detective gets onto her right
away. Her lies sink her deeper and deeper into a self-loathing hole, causing her
to make a bad situation worse.<br /><br />This is a very thought-provoking
```

```
story, and one can't help but to feel this lady's pain, wishing throughout that
she would simply come clean.<br /><br />As a TV movie, thanks to Colin and a
strong script, this is a well above average TV movie.

Correct Class:  1

Predicted Class:  1
```

[17]:
```
print("Data:", sreviews[99], '\n')
print("Correct Class: ", slabels[99], '\n')
print("Predicted Class: ", predictions[99], '\n')
```

```
Data: Good, boring or bad? It's good. Worth your money? If you can spare it for
a ticket, sure. Better than the trailer makes it seem? Yes, oddly.<br /><br
/>There isn't much to the script - Guards working at armored truck company move
vast amounts of cash. Guards see opportunity to retire as millionaires, one of
them is too honest to go along with it all, and a well-laid plan goes to hell.
<br /><br />This could have been a poorly-executed Reservoir Dogs ripoff, but
the skill of the cast and the director's ability to make just about anything
tense pull it out of that realm and put it onto a solid footing.

Correct Class:  1

Predicted Class:  1
```

We can see that the DistilBERT model has correctly classified these reviews while the LR model did not. In Homework 5, the reasoning behind the misclassification in these cases was hypothesized to be due to the presence of negative words in these reviews. Although these reviews carry a positive sentiment, there are several negative words that are used in each such as "bad," "boring," and "poorly-executed." However, these words are not directly used to describe an opinion on the movie. Unforunately, the LR model simply notes the presence of these words, which likely led to the misclassification.

On the other hand, the DistilBERT model, which is more sophisticated, is able to track more of the context that surrounds these negative words. So, it is easier for this model to determine whether these words are used to describe an opinion about the movie in question. This is likely what gave the DistilBERT model the upperhand and allowed these previously misclassified reviews to be classified correctly.

5.4 Bonus: Sentiment Analysis of Rotten Tomatoes Reviews (40p)

Evaluate the model on the development portion of the Rotten Tomatoes dataset. Do a similar analysis as done for the IMDB dataset.

[18]:
```
def read_examples(ds):
  labels = []
  reviews = []

  # YOUR CODE HERE
  for i in range(len(ds)):
      line = ds[i]['text']
      [label, text] = line.strip().split('\t', maxsplit = 1)
      reviews.append(text)
      labels.append(int(label))

  return reviews, labels
```

[19]:
```
dev = 'dev.txt'

data_files = {"dev": dev}
dataset = load_dataset('text', data_files = data_files)

reviews, labels = read_examples(dataset['dev'])

print(labels[:5])
print(labels[-5:])
```

```
Downloading data files:   0%|          | 0/1 [00:00<?, ?it/s]
Extracting data files:   0%|          | 0/1 [00:00<?, ?it/s]
Generating dev split: 0 examples [00:00, ? examples/s]
[0, 1, 0, 1, 0]
[0, 0, 1, 0, 0]
```

[20]:
```
checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"

tokenizer = AutoTokenizer.from_pretrained(checkpoint)

model = AutoModelForSequenceClassification.from_pretrained(checkpoint)

# Print the numerical IDs for the two labels, to verify 0 means negative, 1␣
 ↪means positive.
print(model.config.id2label)

# Run evaluations on all examples.
sreviews, slabels = reviews, labels

# Position for the current batch.
position = 0
# Accumulate batch predicted labels here.
predictions = []
```

```
while position < len(sreviews):
  # Evaluate on a batch of 10 reviews at a time.
  batch = sreviews[position: position + 10]

  # Will pad the sequences up to the max length in the dataset.
  # Will also truncate the sequences that are longer than the model max length␣
 ↪(512 for BERT or DistilBERT).
  tokens = tokenizer(batch, padding = "longest", truncation = True,␣
 ↪return_tensors = "pt")

  output = model(**tokens)
  predictions += list(map(lambda logit: int(logit[0] < logit[1]), output.
 ↪logits))

  position += 10
  print('Processed', position, 'reviews.')
```

```
{0: 'NEGATIVE', 1: 'POSITIVE'}
Processed 10 reviews.
Processed 20 reviews.
Processed 30 reviews.
Processed 40 reviews.
Processed 50 reviews.
Processed 60 reviews.
Processed 70 reviews.
Processed 80 reviews.
Processed 90 reviews.
Processed 100 reviews.
Processed 110 reviews.
Processed 120 reviews.
Processed 130 reviews.
Processed 140 reviews.
Processed 150 reviews.
Processed 160 reviews.
Processed 170 reviews.
Processed 180 reviews.
Processed 190 reviews.
Processed 200 reviews.
Processed 210 reviews.
Processed 220 reviews.
Processed 230 reviews.
Processed 240 reviews.
Processed 250 reviews.
Processed 260 reviews.
Processed 270 reviews.
Processed 280 reviews.
```

```
Processed 290 reviews.
Processed 300 reviews.
Processed 310 reviews.
Processed 320 reviews.
Processed 330 reviews.
Processed 340 reviews.
Processed 350 reviews.
Processed 360 reviews.
Processed 370 reviews.
Processed 380 reviews.
Processed 390 reviews.
Processed 400 reviews.
Processed 410 reviews.
Processed 420 reviews.
Processed 430 reviews.
Processed 440 reviews.
Processed 450 reviews.
Processed 460 reviews.
Processed 470 reviews.
Processed 480 reviews.
Processed 490 reviews.
Processed 500 reviews.
Processed 510 reviews.
Processed 520 reviews.
Processed 530 reviews.
Processed 540 reviews.
Processed 550 reviews.
Processed 560 reviews.
Processed 570 reviews.
Processed 580 reviews.
Processed 590 reviews.
Processed 600 reviews.
Processed 610 reviews.
Processed 620 reviews.
Processed 630 reviews.
Processed 640 reviews.
Processed 650 reviews.
Processed 660 reviews.
Processed 670 reviews.
Processed 680 reviews.
Processed 690 reviews.
Processed 700 reviews.
Processed 710 reviews.
Processed 720 reviews.
Processed 730 reviews.
Processed 740 reviews.
Processed 750 reviews.
Processed 760 reviews.
```

```
Processed 770 reviews.
Processed 780 reviews.
Processed 790 reviews.
Processed 800 reviews.
Processed 810 reviews.
Processed 820 reviews.
Processed 830 reviews.
Processed 840 reviews.
Processed 850 reviews.
Processed 860 reviews.
Processed 870 reviews.
Processed 880 reviews.
Processed 890 reviews.
Processed 900 reviews.
Processed 910 reviews.
Processed 920 reviews.
Processed 930 reviews.
Processed 940 reviews.
Processed 950 reviews.
Processed 960 reviews.
Processed 970 reviews.
Processed 980 reviews.
Processed 990 reviews.
Processed 1000 reviews.
Processed 1010 reviews.
Processed 1020 reviews.
Processed 1030 reviews.
Processed 1040 reviews.
Processed 1050 reviews.
Processed 1060 reviews.
Processed 1070 reviews.
```

[21]:
```
print('Accuracy = ', compute_accuracy(slabels, predictions))
```

```
Accuracy =  0.8864915572232646
```

The final accuracy achieved by the DistilBERT model on the Rotten Tomatoes dataset was about 88.65%. This accuracy is not as high as what was achieved on the IMDB dataset, but it is very close and is still a high score, which suggests that overall this model performs well on sentiment classification tasks.

In the previous homework assignment, Homework 8, the same Rotten Tomatoes dataset was used with different variations of RNN models to find the best performing model. Through experimentation, the best model was found to be two LSTMs (bidirectional) + a fully connected network, concatenating the averages of their states. This model achieved an accuracy of 78.04%. Our DistilBERT model has shown an improvement of around 10% on the same data! This difference in accuracy is not insignificant and shows that the DistilBERT transformer model is much more effective with sentiment classification. A tabular representation of the model results can be seen below.

Assignment	Homework 9	Homework 8
Model	DistilBERT	RNN
Test Accuracy	88.65%	78.04%

[22]:
```
count = 0
for i in range(len(slabels)):
    if slabels[i] != predictions[i] and count < 10:
        print("Test case ", i, " with class ", slabels[i], " was incorrectly␣
 ↪classified as ", predictions[i])
        count += 1
```

```
Test case  0  with class  0  was incorrectly classified as  1
Test case  6  with class  1  was incorrectly classified as  0
Test case  24  with class  0  was incorrectly classified as  1
Test case  25  with class  1  was incorrectly classified as  0
Test case  30  with class  0  was incorrectly classified as  1
Test case  31  with class  1  was incorrectly classified as  0
Test case  36  with class  0  was incorrectly classified as  1
Test case  37  with class  0  was incorrectly classified as  1
Test case  39  with class  1  was incorrectly classified as  0
Test case  40  with class  0  was incorrectly classified as  1
```

[23]:
```
print("Data:", sreviews[0], '\n')
print("Correct Class: ", slabels[0], '\n')
print("Predicted Class: ", predictions[0], '\n')
```

```
Data: exactly what you'd expect from a guy named kaos .

Correct Class:  0

Predicted Class:  1
```

[24]:
```
print("Data:", sreviews[31], '\n')
print("Correct Class: ", slabels[31], '\n')
print("Predicted Class: ", predictions[31], '\n')
```

```
Data: if this movie were a book , it would be a page-turner , you can't wait to
see what happens next .

Correct Class:  1

Predicted Class:  0
```

It seems that many of the cases where the DistilBERT model misclassified were cases of very short reviews. There is not much textual content in these reviews for the model to work with which

already makes the classification task difficult. However, in addition to this, we can also see in the above examples that several of the misclassified reviews include expressions that may not be recognized by the model such as "page-turner" and sarcastic comments such as "exactly what you'd expect from a guy named kaos." There are simply not many indicative words in these reviews that the model could use to form a solid conclusion about the sentiment held in these reviews. For these reasons, it makes sense that the model misclassified in these cases.

5.5 Sentiment Analysis using GPT (90p)

- (5p) Create a subset of 50 reviews from the top 25 reviews (positive) an the bottom 25 reviews (negative) in the develoment portion of the IMDB dataset.
- (50p) Use the chat completion API with GPT-3.5 or GPT-4 to do sentiment analysis of the 50 reviews.
- (10p) Compute the accuracy of GPT and compare it with the accuracy of DistilBERT on the same set of 50 examples.
- (25p) If GPT does nto achieve 100% accuracy, identify the examples on which it makes mistakes and ask it to explain its decision. Use that error analysis to design a better prompt and re-evaluate GPT with the new prompt.

[27]:
```python
def read_examples(ds):
  labels = []
  reviews = []

  # YOUR CODE HERE
  for i in range(len(ds)):
      line = ds[i]['text']
      [label, text] = line.strip().split(' ', maxsplit = 1)
      reviews.append(text)
      if label == 'pos':
        labels.append(1)
      else:
        labels.append(0)


  return reviews, labels
```

[30]:
```python
# YOUR CODE HERE

# Load the IMDB dataset. Only the development portion will be used later.
url_train = "https://webpages.charlotte.edu/rbunescu/courses/itcs4111/hw09/data/
 ↪imdb/train.txt"
url_test = "https://webpages.charlotte.edu/rbunescu/courses/itcs4111/hw09/data/
 ↪imdb/test.txt"
url_dev = "https://webpages.charlotte.edu/rbunescu/courses/itcs4111/hw09/data/
 ↪imdb/dev.txt"
```

```
data_files = {"train": url_train, "test": url_test, "dev": url_dev}
dataset = load_dataset('text', data_files = data_files)
del dataset['train']
del dataset['test']
dataset

reviews, labels = read_examples(dataset['dev'])

sreviews, slabels = sample_dataset(reviews, labels, 25)
```

[31]:
```
!pip install --upgrade openai
!pip install tiktoken
!pip install python-dotenv
```

```
Collecting openai
  Downloading openai-1.3.7-py3-none-any.whl (221 kB)
                             221.4/221.4
kB 4.6 MB/s eta 0:00:00
Requirement already satisfied: anyio<4,>=3.5.0 in
/usr/local/lib/python3.10/dist-packages (from openai) (3.7.1)
Requirement already satisfied: distro<2,>=1.7.0 in /usr/lib/python3/dist-
packages (from openai) (1.7.0)
Collecting httpx<1,>=0.23.0 (from openai)
  Downloading httpx-0.25.2-py3-none-any.whl (74 kB)
                             75.0/75.0 kB
8.7 MB/s eta 0:00:00
Requirement already satisfied: pydantic<3,>=1.9.0 in
/usr/local/lib/python3.10/dist-packages (from openai) (1.10.13)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-
packages (from openai) (1.3.0)
Requirement already satisfied: tqdm>4 in /usr/local/lib/python3.10/dist-packages
(from openai) (4.66.1)
Requirement already satisfied: typing-extensions<5,>=4.5 in
/usr/local/lib/python3.10/dist-packages (from openai) (4.5.0)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-
packages (from anyio<4,>=3.5.0->openai) (3.6)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-
packages (from anyio<4,>=3.5.0->openai) (1.2.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-
packages (from httpx<1,>=0.23.0->openai) (2023.11.17)
Collecting httpcore==1.* (from httpx<1,>=0.23.0->openai)
  Downloading httpcore-1.0.2-py3-none-any.whl (76 kB)
                             76.9/76.9 kB
8.3 MB/s eta 0:00:00
Collecting h11<0.15,>=0.13 (from httpcore==1.*->httpx<1,>=0.23.0->openai)
  Downloading h11-0.14.0-py3-none-any.whl (58 kB)
                             58.3/58.3 kB
```

```
6.2 MB/s eta 0:00:00
Installing collected packages: h11, httpcore, httpx, openai
ERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the following
dependency conflicts.
llmx 0.0.15a0 requires cohere, which is not installed.
llmx 0.0.15a0 requires tiktoken, which is not installed.
Successfully installed h11-0.14.0 httpcore-1.0.2 httpx-0.25.2 openai-1.3.7
Collecting tiktoken
  Downloading
tiktoken-0.5.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.0
MB)
                           2.0/2.0 MB
21.0 MB/s eta 0:00:00
Requirement already satisfied: regex>=2022.1.18 in
/usr/local/lib/python3.10/dist-packages (from tiktoken) (2023.6.3)
Requirement already satisfied: requests>=2.26.0 in
/usr/local/lib/python3.10/dist-packages (from tiktoken) (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.26.0->tiktoken)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests>=2.26.0->tiktoken) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.26.0->tiktoken)
(2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.26.0->tiktoken)
(2023.11.17)
Installing collected packages: tiktoken
ERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the following
dependency conflicts.
llmx 0.0.15a0 requires cohere, which is not installed.
Successfully installed tiktoken-0.5.2
Collecting python-dotenv
  Downloading python_dotenv-1.0.0-py3-none-any.whl (19 kB)
Installing collected packages: python-dotenv
Successfully installed python-dotenv-1.0.0
```

[32]:
```
%env OPENAI_API_KEY = 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
```

```
env: OPENAI_API_KEY=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
```

[33]:
```
import os
from openai import OpenAI

client = OpenAI(api_key = os.environ['OPENAI_API_KEY'],)

def get_completion_from_messages(messages, model = "gpt-3.5-turbo", temperature␣
 ↪= 0, max_tokens = 500):
    response = client.chat.completions.create(model = model,
                                              messages = messages,
                                              temperature = temperature, # the␣
 ↪degree of randomness of the model's output.
                                              max_tokens = max_tokens) # the␣
 ↪maximum number of tokens the model can output.
    return response.choices[0].message.content.strip()
```

[34]:
```
import time
gpt_pred = []
gpt_explanation = []

for r in sreviews:
    review = r

    user_message = f"For the following movie review, determine if there is \
    an overall positive sentiment about the movie. Respond with an integer: use␣
 ↪a 1 to indicate positive sentiment, \
    otherwise 0. Then, provide an explanation for this answer. Review: {review}"

    messages =  [  {'role':'system', 'content': "You are a helpful assistant"},
                   {'role':'user', 'content': user_message}]

    response = get_completion_from_messages(messages)
    #print(response)
    # time.sleep(10)
    gpt_explanation.append(response)
    if '1' in response:
      gpt_pred.append(1)
    elif '0' in response:
      gpt_pred.append(0)

print("System labels complete. \n")
```

```
System labels complete.
```

[35]:
```
print('Accuracy = ', compute_accuracy(slabels, gpt_pred))
```

```
Accuracy =  0.86
```

The GPT model got an accuracy of 86% while the DistilBERT model achieved an accuracy of 88.65%. It is possible that the accuracy of the GPT model could be improved with a more effective prompt.

[36]:
```
count = 0
for i in range(len(slabels)):
    if slabels[i] != gpt_pred[i] and count < 10:
        print("Test case ", i, " with class ", slabels[i], " was incorrectly␣
 ↪classified as ", gpt_pred[i])
        count += 1
```

```
Test case  0  with class  1  was incorrectly classified as  0
Test case  15  with class  1  was incorrectly classified as  0
Test case  18  with class  1  was incorrectly classified as  0
Test case  30  with class  0  was incorrectly classified as  1
Test case  32  with class  0  was incorrectly classified as  1
Test case  33  with class  0  was incorrectly classified as  1
Test case  39  with class  0  was incorrectly classified as  1
```

[37]:
```
print("Data:", sreviews[0], '\n')
print("Correct Class: ", slabels[0], '\n')
print("Predicted Class: ", gpt_explanation[0], '\n')
```

```
Data: So Dark The Night poses a tough challenge: It's very hard to write about
it in any detail without ruining it for those who haven't yet seen it. Since it
remains quite obscure, that includes just about everybody. The movie will strike
those familiar with its director Joseph H. Lewis' better known titles in the
noir cycle  Gun Crazy, The Big Combo, even My Name Is Julia Ross, which in its
brevity it resembles  as an odd choice.<br /><br />For starters, the bucolic
French countryside serves as its setting. Steven Geray, a middle-aged detective
with the Surété in Paris, sets out for a vacation in the village of Ste. Margot
(or maybe Margaux). Quite unexpectedly, he finds himself falling in love with
the inkeepers' daughter (Micheline Cheirel), even though she's betrothed to a
rough-hewn local farmer. But the siren song of life in Paris is hard to resist,
so she agrees to marry him, despite the disparity in their ages, which
inevitably becomes the talk of the town. <br /><br />But on the night of their
engagement party, she fails to return to the inn. Soon, a hunchback finds her
body by the river. Her jealous, jilted lover is the logical suspect, but he,
too, is found dead. Then anonymous notes threaten more deaths, which come to
pass. For the first time in his career, the bereaved Geray finds himself
stumped…<br /><br />A particularly weak script all but does the movie in; it
plays like bad Cornell Woolrich crossed with The Murder of Roger Ackroyd. But
Lewis does this creaky vehicle proud. He takes his time near the beginning, but
then the story  and the storytelling  gain momentum (alas, just about the time
the script breaks an axle). Burnett Guffey lighted and photographed the film,
with an intriguing leitmotif of peering out of and peeping into windows; there's
also an effective score by Hugo Friedhofer, who supplied aural menace to many
noirs. A good deal of talent has been lavished on So Dark The Night, but at the
```

end it boils down to not much more than a gimmick and not a very good gimmick at that. It's a one-trick pony of a movie.

Correct Class: 1

Predicted Class: 0

Explanation: The review mentions several negative aspects of the movie, such as a weak script, a gimmicky plot, and a one-trick pony nature. These criticisms indicate a negative sentiment towards the movie.

[38]:
```
print("Data:", sreviews[18], '\n')
print("Correct Class: ", slabels[18], '\n')
print("Predicted Class: ", gpt_explanation[18], '\n')
```

Data: The four signs on the road say "If You're Looking For Fun…You Don't Need A Reason…All You Need Is A Gun…It's Rabbit Season!"

In the woods, we see hundreds of "Rabbit Season" signs posted on every tree. We see more and more signs pointing exactly to Bugs Bunny's hole. Who's putting up all these signs? Daffy Duck!

Daffy puts the last sign up, tiptoes away and says to us, the audience, "Awfully unsporting of me, I know. But, what the hey - I gotta have some fun! Besides, it's really duck season."

From that point, we now see Elmer Fudd, shotgun in hand…and a war of semantics between Bugs and Daffy with Bugs winning every time. Only in cartoons, thankfully, can we see someone getting shotgun-blasted in the head five times and keep going!

Correct Class: 1

Predicted Class: 0

The review does not provide any explicit positive sentiment about the movie. It simply describes a scene from a cartoon where there is a war of semantics between Bugs and Daffy, with Bugs winning every time. The review does not mention any specific aspects of the movie that would indicate a positive sentiment.

[39]:
```
print("Data:", sreviews[30], '\n')
print("Correct Class: ", slabels[30], '\n')
print("Predicted Class: ", gpt_explanation[30], '\n')
```

Data: This film has the language, the style and the attitude down … plus greats rides from Occy (a world champ) and the great Jerry Lopez. John Philbin as Turtle has the surf pidgin down, and the surfing scenes are still the best ever. A true classic that can be seen many times. Nia Peeples is a babe, and Laird Hamilton shows the early stuff that has made him the world's number one

```
extreme surfer.

Correct Class:  0

Predicted Class:  1

Explanation: The review mentions positive aspects of the movie such as the
language, style, attitude, great rides from professional surfers, and the
surfing scenes being the best ever. It also describes the movie as a true
classic that can be seen many times. Additionally, it praises the performances
of the actors and mentions Nia Peeples as a babe and Laird Hamilton as the
world's number one extreme surfer. Overall, the review expresses a positive
sentiment towards the movie.
```

In the first misclassification example shown above (review index 0), the GPT model provides a solid explanation for its prediction of negative sentiment. In fact, the explanation is so reasonable that I believe this review may have been misclassified in the ground-truth labels. The same can be said about the last misclassification example shown above (review index 30). I would argue that GPT has correctly classified these reviews to a point that it has performed better than whichever person manually labeled these reviews.

In the second misclassification example shown above (review index 18), GPT explains that there is no real sentiment given in the review. I would agree with this; there is simply a summary of a scene in the movie. With this in mind, it makes sense that the model assigned the negative class to this review as the prompt states that 1 indicates positive sentiment, otherwise 0.

---

6 Bonus: Named Entity Recognition (60p)

Run the HuggingFace NER Transformer on the examples from the Theory section. Below is sample code adapted from the HuggingFace course section on Transformers what they can do?.

[40]:
```python
from transformers import pipeline

ner = pipeline("ner", grouped_entities = True)
ner("UNC Charlotte is a public research university in North Carolina.")

# YOUR CODE HERE
# 1.   The third mate was Flask, a native of Tisbury, in Martha's Vineyard.
# 2.   Its official Nintendo announced today that they Will release the␣
 ↪Nintendo 3DS in north America march 27.
# 3.   Jessica Reif, a media analyst at Merrill Lynch & Co., said, "If they can␣
 ↪get up and running with exclusive programming within six months, it doesn't␣
 ↪set the venture back that far."
```

```
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.

config.json:   0%|          | 0.00/998 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/1.33G [00:00<?, ?B/s]

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).

tokenizer_config.json:   0%|          | 0.00/60.0 [00:00<?, ?B/s]

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

/usr/local/lib/python3.10/dist-
packages/transformers/pipelines/token_classification.py:169: UserWarning:
`grouped_entities` is deprecated and will be removed in version v5.0.0,
defaulted to `aggregation_strategy="simple"` instead.
  warnings.warn(
```

[40]:
```
[{'entity_group': 'ORG',
  'score': 0.982141,
  'word': 'UNC Charlotte',
  'start': 0,
  'end': 13},
 {'entity_group': 'LOC',
  'score': 0.99887604,
  'word': 'North Carolina',
  'start': 49,
  'end': 63}]
```

[41]:
```
ner("The third mate was Flask, a native of Tisbury, in Martha's Vineyard.")
```

[41]:
```
[{'entity_group': 'PER',
  'score': 0.99465436,
  'word': 'Flask',
  'start': 19,
  'end': 24},
 {'entity_group': 'LOC',
  'score': 0.9023142,
```

```
  'word': 'Tisbury',
  'start': 38,
  'end': 45},
 {'entity_group': 'LOC',
  'score': 0.96189433,
  'word': 'Martha ’ s Vineyard',
  'start': 50,
  'end': 67}]
```

[42]:
```
ner("Its official Nintendo announced today that they Will release the Nintendo␣
 ↪3DS in north America march 27.")
```

[42]:
```
[{'entity_group': 'ORG',
  'score': 0.9988274,
  'word': 'Nintendo',
  'start': 13,
  'end': 21},
 {'entity_group': 'MISC',
  'score': 0.9949834,
  'word': 'Nintendo 3DS',
  'start': 65,
  'end': 77},
 {'entity_group': 'LOC',
  'score': 0.99817383,
  'word': 'America',
  'start': 87,
  'end': 94}]
```

[43]:
```
ner("Jessica Reif, a media analyst at Merrill Lynch & Co., said, 'If they can␣
 ↪get up and running with exclusive programming within six months, it doesn't␣
 ↪set the venture back that far.'")
```

[43]:
```
[{'entity_group': 'PER',
  'score': 0.9987101,
  'word': 'Jessica Reif',
  'start': 0,
  'end': 12},
 {'entity_group': 'ORG',
  'score': 0.99618655,
  'word': 'Merrill Lynch & Co.',
  'start': 33,
  'end': 52}]
```

Evaluate the performance of the default Transformer model on the development portion of the CoNLL named entity dataset, and compare it against the performance of CRFs from a previous assignment.

To complete this exercise, you may consider reusing code from the CRF assignment and/or NER

code from the Token Classification section of the HuggingFace course. Similar to the sentiment analysis portion, you may need to run the NER Transformer on small batches of sentences at a time so that Colab does not run out of memory.

[ ]:
```
from datasets import load_dataset

# Only the development portion will be used.
url_train = "eng.train"
url_test = "eng.testb.blind"
url_dev = "eng.testa"

# YOUR CODE HERE
data_files = {"train": url_train, "test": url_test, "dev": url_dev}
dataset = load_dataset('text', data_files = data_files)
del dataset['train']
del dataset['test']
dataset
```

7 Bonus: Zero-shot Classification (50p)

Zero-shot classification refers to using a model to classify text into labels for which the model was not explicitly trained. The example below, adapted from the HuggingFace course, shows how to use the default Transformer classifier to compute probabilities for new textual labels.

[44]:
```
from transformers import pipeline

classifier = pipeline("zero-shot-classification")
classifier(
    "This is a computer science course about Natural Language Processing and␣
 ↪Machine Learning.",
    candidate_labels=["education", "politics", "business"],
)
```

```
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]
```

[44]:
```
{'sequence': 'This is a computer science course about Natural Language
Processing and Machine Learning.',
 'labels': ['education', 'business', 'politics'],
 'scores': [0.47796154022216797, 0.3644331991672516, 0.15760529041290283]}
```

Create or use an existing corpus that has at least 2 textual labels and 200 examples and evaluate the accuracy of the default Transformer model on text classification in the zero-shot setting. If in assignment 5 you created a text classification corpus, you can evaluate the Transformer model on that corpus.

[45]:
```
def read_examples(ds):
  labels = []
  reviews = []

  # YOUR CODE HERE
  for i in range(len(ds)):
      line = ds[i]['text']
      [label, text] = line.strip().split(' ', maxsplit = 1)
      reviews.append(text)
      labels.append(label)

  return reviews, labels
```

[46]:
```
# YOUR CODE HERE
from datasets import load_dataset

# USING TRAINING DATASET (480 samples) BECAUSE DEV AND TEST SETS ARE NOT >= 200␣
 ↪SAMPLES
url_train = "imdb_plot_train.txt"

data_files = {"data": url_train}
dataset = load_dataset('text', data_files = data_files)
dataset

reviews, labels = read_examples(dataset['data'])
```

```
Downloading data files:   0%|          | 0/1 [00:00<?, ?it/s]

Extracting data files:   0%|          | 0/1 [00:00<?, ?it/s]

Generating data split: 0 examples [00:00, ? examples/s]
```

[47]:
```
from transformers import AutoTokenizer
import numpy as np
checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"

# Position for the current batch.
position = 0
```

```
# Accumulate batch predicted labels here.
predictions = []

tokenizer = AutoTokenizer.from_pretrained(checkpoint)
classifier = pipeline("zero-shot-classification")

while position < len(reviews):
  # Evaluate on a batch of 10 reviews at a time.
  batch = reviews[position]
  # print(batch)

  # Will pad the sequences up to the max length in the dataset.
  # Will also truncate the sequences that are longer than the model max length␣
 ↪(512 for BERT or DistilBERT).
  tokens = tokenizer(batch, padding = "longest", truncation = True,␣
 ↪return_tensors = "pt")

  output = classifier(tokens, candidate_labels=["action", "comedy", "romance"],)
  predictions.append(np.argmax(output['scores']))

  position += 1
  if position % 20 == 0:
    print('Processed', position, 'reviews.')
```

```
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.
```

```
Processed 20 reviews.
Processed 40 reviews.
Processed 60 reviews.
Processed 80 reviews.
Processed 100 reviews.
Processed 120 reviews.
Processed 140 reviews.
Processed 160 reviews.
Processed 180 reviews.
Processed 200 reviews.
Processed 220 reviews.
Processed 240 reviews.
Processed 260 reviews.
Processed 280 reviews.
Processed 300 reviews.
Processed 320 reviews.
Processed 340 reviews.
Processed 360 reviews.
Processed 380 reviews.
```

```
Processed 400 reviews.
Processed 420 reviews.
Processed 440 reviews.
Processed 460 reviews.
Processed 480 reviews.
```

[48]:
```
for i, label in enumerate(labels):
  if labels[i] == 'action':
    labels[i] = 0
  elif labels[i] == 'comedy':
    labels[i] = 1
  elif labels[i] == 'romance':
    labels[i] = 2
```

[50]:
```
print('Accuracy = ', compute_accuracy(labels, predictions))
```

```
Accuracy =  0.3333333333333333
```

8 Bonus: Anything extra goes here

[ ]:
```
# YOUR CODE HERE
```