TransformersRule

April 21, 2026

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.

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:

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 -> Save and Export Notebook as -> PDF 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:

3.2 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 is explained on slides 5-7 of the lecture on manual annotation for NLP.

YOUR SOLUTION HERE:

4 Implementation using HuggingFace Transformers

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

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

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

[ ]:
```
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)
```

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 provides 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

[ ]:
```
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/itcs4101/hw10/data/
  ↪imdb/train.txt"
url_test = "https://webpages.charlotte.edu/rbunescu/courses/itcs4101/hw10/data/
  ↪imdb/test.txt"
url_dev = "https://webpages.charlotte.edu/rbunescu/courses/itcs4101/hw10/data/
  ↪imdb/dev.txt"

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

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

[ ]:
```
# 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
```

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.

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

  # YOUR CODE HERE




  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:])
```

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.

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

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.

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

  # YOUR CODE HERE



  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.

[ ]:
```
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 = True
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.')
```

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`.

```
[ ]: def compute_accuracy(labels, predictions):
       accuracy = 0 # YOUR CODE HERE. CAN YOU DO IT IN ONE LINE?

       return accuracy

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

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.
- [**(25p) Bonus**] Elaborate on how the model performance could be improved and write code that leads to improved performance.

5.4 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.

[1]:
```
# Only the development portion will be used.
url_train = "https://webpages.charlotte.edu/rbunescu/courses/itcs4101/hw10/data/
  ↪rt/train.txt"
url_test = "https://webpages.charlotte.edu/rbunescu/courses/itcs4101/hw10/data/
  ↪rt/test-blind.txt"
url_dev = "https://webpages.charlotte.edu/rbunescu/courses/itcs4101/hw10/data/
  ↪rt/dev.txt"

# YOUR CODE HERE
```

5.5 Sentiment Analysis using LLMs (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 an LLM (GPT, Llama, Gemini) to do sentiment analysis of the 50 reviews.
- (10p) Compute the accuracy of the LLM and compare it with the accuracy of DistilBERT on the same set of 50 examples.
- (25p) If the LLM does not achieve 100% accuracy, identify the examples on which it makes mistakes and ask it (the LLM) to explain its decision. Use that error analysis to design a better prompt and re-evaluate GPT with the new prompt.

[1]:
```
# YOUR CODE HERE
```

---

6 Bonus: Named Entity Recognition (30p)

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?.

[ ]:
```
from transformers import pipeline

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

# YOUR CODE HERE
```

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 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.

[3]:
```
# Only the development portion will be used.
url_train = "https://webpages.charlotte.edu/rbunescu/courses/itcs4101/hw10/ner/
 ↪eng.train"
url_test = "https://webpages.charlotte.edu/rbunescu/courses/itcs4101/hw10/ner/
 ↪eng.testb.blind"
url_dev = "https://webpages.charlotte.edu/rbunescu/courses/itcs4101/hw10/ner/
 ↪eng.testa"

# YOUR CODE HERE
```

7 Bonus: Zero-shot Classification (30p)

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.

[ ]:
```
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"],
)
```

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.

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

8 Bonus: Anything extra goes here

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