**Commonlit Readability Prize**

Technical Report

Name: Raymond Luo

Date: 17/09/2021

**Preliminary requirements**

**Install Huggingface datasets library**

* Datasets library is required to package training data, both inputs and targets into a dataset compatible with the Huggingface Trainer API

!pip install datasets

**Import libraries**

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import random

import torch

from datasets.arrow\_dataset import Dataset

from sklearn.model\_selection import train\_test\_split

from transformers import AutoTokenizer

from transformers import Trainer

from transformers import TrainingArguments

from transformers import AutoModelForSequenceClassification

import os

for dirname, \_, filenames in os.walk('/kaggle/input'):

    for filename in filenames:

        print(os.path.join(dirname, filename))

**Seed everything for reproducibility**

* Considerable amount of randomness during model training would lead to different results each timethe notebook is executed
* Sources of randomness include dropout and weight initialization

seed = 3

# python RNG

random.seed(seed)

# pytorch RNGs

torch.manual\_seed(seed)

torch.backends.cudnn.deterministic = True

if torch.cuda.is\_available(): torch.cuda.manual\_seed\_all(seed)

# numpy RNG

np.random.seed(seed)

**Data Preparation**

**Load training and prediction data**

* Utilise pandas library to read in csv files for the training and test sets
* Columns are dropped in the train DataFrame to contain only useful data

source\_dir = '/kaggle/input/commonlitreadabilityprize/'

train\_data = pd.read\_csv(source\_dir + 'train.csv')

# Drop unnecessary columns and rename target column to labels (required for Trainer)

train\_data = train\_data.drop(columns = ["id", "url\_legal", "license",

  "standard\_error"]).rename(columns = {"target": "labels"})

preds\_data = pd.read\_csv(source\_dir + 'test.csv')

train\_data.head()

A screenshot of a computer

Description automatically generated

**Output:**

**Replace new line indicators with spaces to remove unnecessary tokens**

train\_data['excerpt'] = train\_data['excerpt'].apply(lambda x : x.replace('\n', ' '))

preds\_data['excerpt'] = preds\_data['excerpt'].apply(lambda x : x.replace('\n', ' '))

train\_data['excerpt'][0]

**Output:**

“When the young people returned to the ballroom, it presented a decidedly changed appearance. Instead of an interior scene, it was a winter landscape. The floor was covered with snow-white canvas, not laid on smoothly, but rumpled over bumps and hillocks, like a real snow field….

**Split train.csv into training and validation sets to evaluate model training**

**Using scikit-learn's train\_test\_split:**

* Specify evaluation size to be 20% of the entire train set
* random\_state argument shuffles the data and makes the shuffling reproducible

train\_set, eval\_set = train\_test\_split(train\_data,

test\_size = 0.2,

random\_state = 42)

train\_set.head()

Graphical user interface

Description automatically generated with low confidence

**Output:**

**Convert pandas DataFrame to Dataset object**

* This is done in order to convert DataFrame into a type which is compatible with the in-built Huggingface Trainer

train\_dataset = Dataset.from\_pandas(df = train\_set)

eval\_dataset = Dataset.from\_pandas(df = eval\_set)

train\_dataset

A picture containing text

Description automatically generated

**Output:**

**Load pretrained RoBERTa tokenizer**

* Tokenizer has a pretrained set of vocabulary and assigns an integer to each word in a passage
* Convert text into numerical form to pass into the transformer

tokenizer = AutoTokenizer.from\_pretrained("roberta-base")

tokenizer.save\_pretrained('./Commonlit-RoBERTa-Base/tokenizer')

tokenizer

**Output:**

PreTrainedTokenizerFast(name\_or\_path='roberta-base', vocab\_size=50265, model\_max\_len=512, is\_fast=True, padding\_side='right', special\_tokens={'bos\_token': '<s>', 'eos\_token': '</s>', 'unk\_token': '<unk>', 'sep\_token': '</s>', 'pad\_token': '<pad>', 'cls\_token': '<>', 'mask\_token': AddedToken("<mask>", rstrip=False, Istrip=True, single\_word=False, normalized=False) })

**Map tokenized text excerpts to Dataset**

1. Define a function which returns sequences of tokens, taking the texts as input in order to prepare the data for the model
   * The sequences are padded to the maximum length defined by the max\_length argument
   * Truncation of the sequence occurs when the excerpt is too long
   * Sequences will have an attention mask with them, which 'masks' padded tokens and tells the model to disregard them
   * Token '1' correlates to the padding tokens
2. Utilise map method on the Datasets to execute the 'tokenize\_data' function on each excerpt stored  in the Dataset
3. Convert data to PyTorch tensors

**Reference:** <https://huggingface.co/transformers/training.html>

MAX\_LENGTH = 256

def tokenize\_data(dataset):

token\_sequence = tokenizer(dataset["excerpt"],

 padding = "max\_length",

truncation = True,

max\_length = MAX\_LENGTH)

    return token\_sequence

# Map tokenized text (input\_ids, attention\_mask) to new dataset

tokenized\_train\_dataset = train\_dataset.map(tokenize\_data,

batched = True).remove\_columns(["\_\_index\_level\_0\_\_", "excerpt"])

tokenized\_eval\_dataset = eval\_dataset.map(tokenize\_data,

batched = True).remove\_columns(["\_\_index\_level\_0\_\_", "excerpt"])

# Convert into PyTorch tensors

tokenized\_train\_dataset.set\_format("torch")

tokenized\_eval\_dataset.set\_format("torch")

tokenized\_train\_dataset['input\_ids'][0]

**Table

Description automatically generatedOutput:**

**Model Selection**

**Load RoBERTa-Base model**

* RoBERTa transformer was pretrained on larger dataset, and generally performs better than BERT
* Attention mechanism is one factor which makes transformers more effective compared to RNN or LSTM, as it enables the model to model long-term dependencies more effectively and weight each word in terms of its significance
* Set **num\_labels** to 1 to indicate a regression problem is involved

# Load model from Huggingface

model = AutoModelForSequenceClassification.from\_pretrained("roberta-base", num\_labels = 1)

**Send the model to GPU memory**

* Required to increase training rate drastically

%%capture

# Send model to GPU

device = torch.device('cuda') if torch.cuda.is\_available() else torch.device('cpu')

model.to(device)

**Set up Trainer instance to train model**

* Trained on 5 epochs and save checkpoints of the model at the end of each epoch
* Best model is loaded at the end and determined by the evaluation loss

batch\_size = 16

training\_args = TrainingArguments(

    output\_dir="./Commonlit-RoBERTa-Base-CP", # Select model path for checkpoint

    overwrite\_output\_dir=True,

    num\_train\_epochs=5,

    per\_device\_train\_batch\_size = batch\_size,

    per\_device\_eval\_batch\_size = batch\_size,

    evaluation\_strategy = 'epoch',

    save\_strategy = "epoch", # Save checkpoint at end of each epoch

    metric\_for\_best\_model = 'eval\_loss',

    greater\_is\_better = False,

    load\_best\_model\_at\_end = True,

    report\_to = "none",

    seed = 3)

# Create Trainer

trainer = Trainer(

    model = model,

    args = training\_args,

    train\_dataset = tokenized\_train\_dataset,

eval\_dataset = tokenized\_eval\_dataset)

# Train model with given parameters

trainer.train()

**Table

Description automatically generatedOutput:**

**Convert model to evaluation mode**

* Required to perform predictions, otherwise the model would still behave as if it was being trained and implement dropout, leading to inconsistent predictions

%%capture

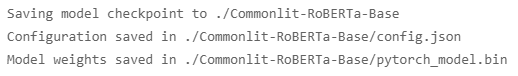
# Convert model to evaluation mode to indicate model is ready for prediction

model.eval()

**Save trained model**

trainer.save\_model('./Commonlit-RoBERTa-Base')

**Output:**

****

**Evaluation**

**Make predictions**

* Tokenize data in the test dataset and pass it through model to obtain the scores
* Add scores to list and export the scores as csv file

preds\_targets = []

for excerpt in preds\_data['excerpt']:

token\_seq = tokenizer(excerpt,

padding = "max\_length",

max\_length = MAX\_LENGTH,

truncation = True,

return\_tensors = "pt"

    token\_seq.to(device)

    pred = model(\*\*token\_seq) # Unpack token sequences tensor

    preds\_targets.append(pred.logits[0].item())

preds\_targets

**Output:**

[-0.007161587942391634, -0.3461619019508362, -0.18076074123382568, -2.6938998699188232, -1.7281757593154907, -0.8599075675010681, 0.5470359325408936]

submission\_df = pd.DataFrame({'id' : preds\_data['id'], 'target': preds\_targets})

submission\_df

Table

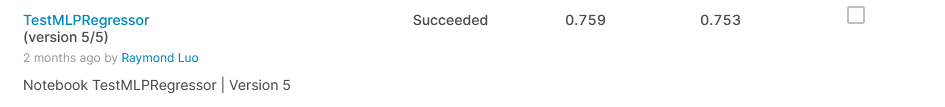
Description automatically generated

**Output:**

submission\_df.to\_csv("submission.csv", index = False)

***For the full notebook, refer to deployment-files/RoBERTa\_Base\_Train.ipynb***

**Evaluate model on Kaggle**

A picture containing graphical user interface

Description automatically generatedGraphical user interface, application

Description automatically generatedModel achieved an RMSE score of **0.538** on the private leaderboard and **0.553** on the public leaderboard which is a significant improvement compared to the feed forward dense neural network, which attained a score of **0.759** on the private leaderboard and **0.753** on the public leaderboard, as well as the bidirectional LSTM which I implemented using the TensorFlow framework, which achieved a score of **0.695** on the private leaderboard and **0.683** on the public leaderboard.

**Evaluate model on scraped texts**

The top 100 most downloaded texts on the Project Gutenberg website were scraped in order to extract a number of excerpts from each of the texts. The excerpts were then passed through the model to obtain a readability score.

To refer to the scraping process, view “gutenberg-scrape/gutenberg-scrape.ipynb”.

10 sets of excerpts from each of the texts have been extracted, with a score associated with each of them. The scores were averaged to give the overall score for the text. Scores can be seen in “gutenberg-scrape/Gutenberg-Predictions/gutenberg-predictions-all.xlsx”. Note that the 90th text: “Calculus Made Easy” was skipped since the only available format of that text is a pdf.

**Limitations**

* From evaluating this model on the scraped excerpts, it is evident that this model has limitations in providing a holistic score for entire texts. The model was trained on excerpts of around 100-250 words, meaning that it was appropriate to utilize the same excerpt length for the scraping. However, since texts typically have far more words, the perceived ‘difficulty’ of each excerpt within the same text will likely be subject to significant variation.
* In addition, often the difficulty of texts especially in a high school setting, lie in the symbolic meanings of the text and is not solely influenced by the complexity of its vocabulary or how difficult a sentence is to read. Such a model is incapable of picking up on such factors, due to the scope of which it has been trained on.
* The target scores of which the model has been trained is not clearly defined in terms of which scores constitute a particular difficulty. Through observation, it can be seen that scores which are lower, tend to be more difficult to read. In order to group these scores into the respective year levels, I have manually defined thresholds of which to group the different texts
* Text

  Description automatically generated with medium confidenceThe HTML of each book in the Gutenberg page have varying structures and identifiers for text. This includes certain texts having contents lists which are enclosed in <p> tags, making it more difficult to extract useful text. A number of texts are also plays, which contain character names in between the useful text. Such an example is given below, and such names should be removed in order to provide a more accurate evaluation.

**Text grouping**

The group of each text is shown below:

**Year 9-**

* The Wonderful Wizard of Oz

**Year 9**

* Alice's Adventures in Wonderland
* The Happy Prince, and Other Tales
* Metamorphosis
* Grimms' Fairy Tales
* Old Granny Fox
* The Picture of Dorian Gray
* Anne of Green Gables

**Year 10**

* The Adventures of Tom Sawyer, Complete
* Peter Pan
* Ethan Frome
* The Secret Garden
* Dubliners
* The Great Gatsby
* The Awakening, and Selected Short Stories
* The Adventures of Sherlock Holmes
* Little Women
* Anna Karenina
* Uncle Tom's Cabin
* A Doll's House : a play
* Complete Original Short Stories of Guy De Maupassant
* The Jungle Book
* The Jungle
* Adventures of Huckleberry Finn
* The Yellow Wallpaper
* Wuthering Heights
* The Importance of Being Earnest: A Trivial Comedy for Serious People
* Around the World in Eighty Days
* A Study in Scarlet

**Year 11**

* The Mysterious Affair at Styles
* Heart of Darkness
* The Hound of the Baskervilles
* The Time Machine
* The Romance of Lust: A classic Victorian erotic novel
* Notes from the Underground
* A Tale of Two Cities
* Oliver Twist
* The Brothers Karamazov
* The American Diary of a Japanese Girl
* Crime and Punishment
* The Strange Case of Dr. Jekyll and Mr. Hyde
* Jane Eyre: An Autobiography
* Anthem
* The Call of the Wild
* The Extraordinary Adventures of Arsene Lupin, Gentleman-Burglar
* Emma
* A Christmas Carol in Prose; Being a Ghost Story of Christmas
* Narrative of the Captivity and Restoration of Mrs. Mary Rowlandson
* Dracula
* Siddhartha
* Treasure Island
* Sense and Sensibility
* War and Peace
* The Count of Monte Cristo, Illustrated
* The War of the Worlds
* The Scarlet Letter
* David Copperfield
* Persuasion
* Narrative of the Life of Frederick Douglass, an American Slave
* Les Misérables

**Year 12**

* Frankenstein; Or, The Modern Prometheus
* Great Expectations
* Pride and Prejudice
* The Art of War
* The Odyssey
* Songs of Innocence, and Songs of Experience
* Don Quixote
* The Souls of Black Folk

**Year 13**

* The Kama Sutra of Vatsyayana
* The Prophet
* Moby Dick; Or, The Whale
* Candide
* The Legend of Sleepy Hollow
* The Republic
* Walden, and On The Duty Of Civil Disobedience
* The King James Version of the Bible
* The Problems of Philosophy
* A Pickle for the Knowing Ones
* The Elements of Style
* Ulysses

**Year 13+**

* Gulliver's Travels into Several Remote Nations of the World
* Beyond Good and Evil
* A Modest Proposal
* The Interesting Narrative of the Life of Olaudah Equiano, Or Gustavus Vassa, The
* The Prince
* ~~The Slang Dictionary: Etymological, Historical and Andecdotal~~
* Essays of Michel de Montaigne — Complete
* Autobiography of Benjamin Franklin
* Thus Spake Zarathustra: A Book for All and None
* Common Sense
* ~~The Devil's Dictionary~~
* The Confessions of St. Augustine
* The History of the Peloponnesian War
* Second Treatise of Government
* The Iliad
* Meditations
* Leviathan
* Beowulf: An Anglo-Saxon Epic Poem
* An Index of The Divine Comedy

**Note:** Texts have been crossed-out as they are unsuitable for reading.

**Boundary Scores**

|  |  |
| --- | --- |
| **Boundary Scores** *(square bracket denotes inclusive, round bracket denotes non-inclusive)* | |
| **Year Level** | **Boundaries** |
| Year 9- | 0.3+ |
| Year 9 | [-0.1, 0.3) |
| Year 10 | [-0.6, -0.1) |
| Year 11 | [-1.1, -0.6) |
| Year 12 | [-1.6, -1.1) |
| Year 13 | [-2.1, -1.6) |
| Year 13+ | -2.1- |

**Text Counts**

**Including crossed-out texts**

|  |  |
| --- | --- |
| **Year Level Counts** | |
| Year Level | Count |
| Year 9- | 1 |
| Year 9 | 7 |
| Year 10 | 21 |
| Year 11 | 31 |
| Year 12 | 8 |
| Year 13 | 12 |
| Year 13+ | 19 |
| Total | 99 |

**Excluding crossed-out texts**

|  |  |
| --- | --- |
| **Year Level Counts** | |
| Year Level | Count |
| Year 9- | 1 |
| Year 9 | 7 |
| Year 10 | 21 |
| Year 11 | 31 |
| Year 12 | 8 |
| Year 13 | 12 |
| Year 13+ | 17 |
| Total | 97 |

**Note:** For more detailed statistics refer to: “gutenberg-scrape/Gutenberg-Predictions/gutenberg-predictions-all.xlsx”