summarization_analysis

April 26, 2021

0.1 1. Data retrieval

[2]:
```
import torch
import gzip
import json
import pandas as pd
import csv
import io

from operator import itemgetter
from summarizer import Summarizer, TransformerSummarizer

import nltk
from nltk.translate.bleu_score import corpus_bleu
```

```
C:\Users\teemu\anaconda3\envs\pytorchEnv\lib\site-
packages\torchaudio\extension\extension.py:13: UserWarning: torchaudio C++
extension is not available.
  warnings.warn('torchaudio C++ extension is not available.')
C:\Users\teemu\anaconda3\envs\pytorchEnv\lib\site-
packages\torchaudio\backend\utils.py:89: UserWarning: No audio backend is
available.
  warnings.warn('No audio backend is available.')
```

0.1.1 Data directory and batch size selection

[3]:
```
# Modify this to wherever you locally downloaded the data
data_base_path = './data/newsroom-release/release/'
wordpiece_cased_path = 'bert-base-cased-vocab.txt'

# train_path = data_base_path + 'train.jsonl.gz' DONT USE THIS
validation_path = data_base_path + 'dev.jsonl.gz'
test_path = data_base_path + 'dev.jsonl.gz'

batch_size = 1
```

[4]:
```
class NewsroomDataset(torch.utils.data.Dataset):
    '''
    Attributes:
        batch_size: Batch size to be taken on single getitem
        file: path to the dataset file
        category: category of the data summarization. i.e. 'extractive'
    '''
    def __init__(self, path, category: str):
        self.category = category
        data = []
        with gzip.open(path) as f:
            for ln in f:
                obj = json.loads(ln)
                data.append(obj)
        data = pd.DataFrame(data)
        # Take only samples with certain category
        self.data = data.loc[data['density_bin'] == self.category, :]

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

    def __getitem__(self, idx):
        return dict(self.data.iloc[idx, :])
```

[5]:

```
test_dset = NewsroomDataset(test_path, "extractive")
testloader = torch.utils.data.DataLoader(test_dset, batch_size=batch_size,␣
 ↪shuffle=True)
```

0.2 2. Initialize model and do predictions

[6]:

```
from nltk.tokenize import sent_tokenize # For Lede-3
```

Define functions for making predictions and writing to file

Generate data for predictions

[10]:

```
def generate_dset(n):
    dset = []
    for i, batch_df in enumerate(testloader):
        txt, summary = itemgetter('text', 'summary')(batch_df)
        txt = ''.join(txt)
        summary = ''.join(summary)

        if True:
            txt = txt.lower()
            summary = summary.lower()
        dset.append((txt, summary))

        if i == n:
            break
    return dset
```

[9]:

```
def make_predictions(transformer_type, transformer_model_key, lower_case=True):
    model = TransformerSummarizer(transformer_type=transformer_type,
                                 transformer_model_key=transformer_model_key)
    results = [] # Predictions for the BERT
    lede3_preds = [] # Lede 3 predictions

    for i, batch_df in enumerate(dset):
        txt, summary = batch_df[0], batch_df[1]

        try:
            pred = model(txt)
        except RuntimeError as exception:
            if "out of memory" in str(exception):
                print("WARNING: out of memory")
                if hasattr(torch.cuda, 'empty_cache'):
                    torch.cuda.empty_cache()

        results.append((pred, summary))

        # Lede-3
        lede3 = ' '.join(sent_tokenize(txt)[:3])
        lede3_preds.append((lede3, summary))

        if i % 10 == 0:
            print(f"prediction: {i}\n")



    return results, lede3_preds

def save_to_file(results, name, column_headers: list, dialect=None):
    # Save model to file
    with io.open(name, 'w', encoding="utf-8") as out:
        csv_out = csv.writer(out)
        csv_out.writerow(column_headers)
        for row in results:
            csv_out.writerow(row)
```

Do predictions and save to file

[11]:

```
n_predictions = 500
BERT = 'Bert'
GPT2_NAME = 'GPT2'

BERT_LARGE = 'bert-large-uncased'
BERT_BASE = 'bert-base-uncased'
```

```
GPT2 = 'gpt2-medium'
GPT2_L = 'gpt2-large'
LEDE = 'lede3'

CLASSIFIERS = [(BERT, BERT_LARGE), (BERT, BERT_BASE), (GPT2_NAME, GPT2),␣
 ↪(GPT2_NAME, GPT2_L)]

dset = generate_dset(n_predictions)
```

[ ]:

```
for i, clf in enumerate(CLASSIFIERS):
    model, lede = make_predictions(clf[0],
                                   clf[1],
                                   n_predictions)
    save_to_file(model, f'{clf[1]}.csv', ['prediction', 'actual'])

    if i == 0:
        # Get Lede-3 to format that csv.writerows wants
        save_to_file(lede, f'{LEDE}.csv', ['prediction', 'actual'])
```

[12]:

```
CLASSIFIERS = [(BERT, BERT_LARGE), (BERT, BERT_BASE), (GPT2_NAME, GPT2),␣
 ↪(GPT2_NAME, GPT2_L), (LEDE, LEDE)]
```

0.3 3. Performance evaluation and results

Get mean Rouge-1, Rouge-2 and Rouge-L scores

[13]:

```
import rouge
from rouge import Rouge

rouge = Rouge()
dfs = []
for df_name in CLASSIFIERS:
    filename = df_name[1]
    name = df_name[0]

    df = pd.read_csv(f"{filename}.csv")
    scores = rouge.get_scores(df.iloc[:, 0], df.iloc[:, 1], avg=True)

    dfs.append((name, pd.DataFrame(scores)))
```

[14]:

```
import plotly
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots

def plot_rouges(rdfs, titles=['Rouge-1', 'Rouge-2', 'Rouge-L']):
    figs = []
    for df, title in zip(rdfs, titles):
        df.insert(0, 'Model name', ['BERT-large', 'BERT-base', 'GPT2-medium',␣
↪'GPT2-large', 'Lede-3'])
        fig = go.Figure(data=[go.Table(
            header=dict(
                values=['<b>Model name</b>', '<b>f1-score</b>', '<b>precision</
↪b>', '<b>recall</b>']
            ),
            cells=dict(
                values=df.T,
                fill_color='white',
            )
        )])
        fig.update_layout(title_text=f"<b>{title}<b>")
        fig.update_layout({'margin':{'t':50}})
        fig.update_layout(height=300)
        figs.append(fig)
    return figs
```

[15]:

```
r1_df = pd.DataFrame([round(m['rouge-1']*100, 2) for n,m in dfs])
r2_df = pd.DataFrame([round(m['rouge-2']*100, 2) for n,m in dfs])
r_df = pd.DataFrame([round(m['rouge-l']*100, 2) for n,m in dfs])
figs = plot_rouges([r1_df, r2_df, r_df])

[f.show() for f in figs]
```

[15]:

```
[None, None, None]
```

0.4 BLEU

[245]:

```
bleu_scores = []
for df_name in CLASSIFIERS:
    filename = df_name[1]
    name = df_name[0]

    df = pd.read_csv(f"{filename}.csv")
    # Tokenize to sentences
    sent_df = df.applymap(lambda x: sent_tokenize(x))
    # Tokenize to words
    word_df = sent_df.applymap(lambda x: [nltk.word_tokenize(s) for s in x])
    # Targets should be joined lists
    word_df.iloc[:, 1] = word_df.iloc[:, 1].apply(lambda x: sum(x, []))
    # Calculate score
    score = corpus_bleu(word_df.iloc[:, 0], word_df.iloc[:, 1])

    bleu_scores.append((name, score))
```

[247]:

```
bleu_df = pd.DataFrame([(b[0], round(b[1]*100, 2)) for b in bleu_scores])
fig = go.Figure(data=[go.Table(
            header=dict(
                values=['<b>Model name</b>', '<b>BLEU score</b>']
            ),
            cells=dict(
                values=bleu_df.T,
                fill_color='white',
            )
        )])
fig.update_layout(title_text=f"<b>BLEU scores<b>")
fig.update_layout({'margin':{'t':50}})
fig.update_layout(height=300)
```

Amount of sentences in summary

[16]:

```
summary_len = []
for d in dset:
    txt, summary = d[0], d[1]
    summary_len.append(len(sent_tokenize(summary)))

fig = px.histogram(pd.DataFrame({'Sentence amount': summary_len}))
fig.update_xaxes(title="Amount of sentences in summary")
```

Amount of words

[27]:

```
summary_words = []
for d in dset:
    txt, summary = d[0], d[1]
    lens = len(nltk.word_tokenize(summary))
    if lens < 200:
        summary_words.append(lens)

fig = px.histogram(pd.DataFrame({'Sentence amount': summary_words}))
fig.update_xaxes(title="Summary word count")
fig.update_layout(showlegend=False)
```