Lab 3 Evaluation

Dan HOU

1. Implement the Bleu metric.

In order to implement the complete process of `BLEU score` without relying on libraries like `NLTK`, we need to calculate the Brevity Penalty (BP) and Modified n-gram Precision in order to arrive at the final BLEU score. The following are the core mathematical expressions:

$$BP = \begin{cases} 1 & \text{if hypothesis longer than reference} \\ e^{1-\frac{\#r}{\#h}} & \text{otherwise} \end{cases}$$

where:

- r is a reference translation (gold)
- h is a translation hypothesis (prediction)

$$P_n = \sum_{n-gram \in hyp} \frac{\#_{clip} n-gram}{\#n-gram}$$

where:

- $\#_{clip}(a) = \min\{\#a \in h, \#a \in r\}$

$$BLEU = BP \times \exp\left(\sum_{n=1}^{N} w_n \log P_n\right)$$

where:

- w_n is the weight assigned to the precision of n-grams of size n, by default, the weights are equal, $w_n = \frac{1}{N}$

[ ]:
```
import math

def brevity_penalty(ref, hyp):
    return 1 if len(ref) < len(hyp) else math.exp(1 - len(ref)/len(hyp))
```

[ ]:
```
# test
brevity_penalty(['There', 'is', 'a', 'cat', 'on', 'the', 'mat'], ['The', 'cat',␣
↪'is', 'on', 'the', 'mat'])
```

[ ]:
```
0.846481724890614
```

[ ]:
```
# generate n-grams sequences
def create_ngrams(sentence, n):
    return [tuple(sentence[i:i+n]) for i in range(len(sentence)-n+1)]
```

[ ]:
```
# test
create_ngrams(['There', 'is', 'a', 'cat', 'on', 'the', 'mat'], 2)
```

[ ]:
```
[('There', 'is'),
 ('is', 'a'),
 ('a', 'cat'),
 ('cat', 'on'),
 ('on', 'the'),
 ('the', 'mat')]
```

[ ]:
```
from collections import Counter

# calculate n-grams accuracy
def clipped_ngram_precision(ref, hyp, n):
    bag_ref = Counter(create_ngrams(ref, n))
    bag_hyp = Counter(create_ngrams(hyp, n))
    # get matching n-grams
    common = sum((min(bag_hyp[ngram], bag_ref.get(ngram, 0)) for ngram in␣
 ↪bag_hyp))
    # compute accuracy
    return 0 if common == 0 else common / sum(bag_hyp.values())
```

[ ]:
```
# test
clipped_ngram_precision(['There', 'is', 'a', 'cat', 'on', 'the', 'mat'],␣
 ↪['The', 'cat', 'is', 'on', 'the', 'mat'], 2)
```

[ ]:
```
0.4
```

[ ]:
```
MIN_PROB = 1e-10

# final bleu score
def bleu_score(ref, hyp, N, tokenizer=lambda s : s.split()):
    # tokenize the sentence
    tokenized_ref = tokenizer(ref)
    tokenized_hyp = tokenizer(hyp)
    # compute_bp
    bp = brevity_penalty(tokenized_ref, tokenized_hyp)
    # compute ngram precision
    in_exp = sum([1/N * math.log(max(clipped_ngram_precision(tokenized_ref,␣
 ↪tokenized_hyp, n), MIN_PROB)) for n in range(1, N+1)])
    return bp * math.exp(in_exp)
```

[ ]:
```
# test
bleu_score('There is a cat on the mat', 'The cat is on the mat', 3)
```

[ ]:
```
0.36973494931036327
```

2. Using the WMT'15 test sets, evaluate the performance of mBart and MarianMT. These two models can be easily used with the HuggingFace API. What can you conclude.

[ ]:
```
import torch

device = torch.device("cuda") if torch.cuda.is_available() else torch.
 ↪device('cpu')
device
```

[ ]:
```
device(type='cpu')
```

[ ]:
```
! wget https://statmt.org/wmt15/test.tgz > log.txt
! tar zxvf test.tgz > log.txt
```

- Data preprocessing

[ ]:
```
from pathlib import Path
import xml.etree.ElementTree as ET
import html

def read_file(file_path):
    # well-formed, parse .sgm file and get root
    root = ET.fromstring(Path(file_path).read_text(encoding='utf-8').
 ↪replace('&', '&amp;').replace('<...>',"&lt;...&gt;"))
    # extracting sentences from <seg> tags
    return [html.unescape(seg).text for seg in root.findall(".//seg")]
```

- mBart

[ ]:
```
from transformers import MBartForConditionalGeneration, MBart50TokenizerFast

# load model and tokenizer
model_mBart = MBartForConditionalGeneration.from_pretrained("facebook/
 ↪mbart-large-50-many-to-many-mmt")
tokenizer_mBart = MBart50TokenizerFast.from_pretrained("facebook/
 ↪mbart-large-50-many-to-many-mmt")
```

[ ]:
```
tokenizer_mBart.lang_code_to_id.keys()
```

[ ]:
```
dict_keys(['ar_AR', 'cs_CZ', 'de_DE', 'en_XX', 'es_XX', 'et_EE', 'fi_FI',
'fr_XX', 'gu_IN', 'hi_IN', 'it_IT', 'ja_XX', 'kk_KZ', 'ko_KR', 'lt_LT', 'lv_LV',
'my_MM', 'ne_NP', 'nl_XX', 'ro_RO', 'ru_RU', 'si_LK', 'tr_TR', 'vi_VN', 'zh_CN',
'af_ZA', 'az_AZ', 'bn_IN', 'fa_IR', 'he_IL', 'hr_HR', 'id_ID', 'ka_GE', 'km_KH',
```

```
'mk_MK', 'ml_IN', 'mn_MN', 'mr_IN', 'pl_PL', 'ps_AF', 'pt_XX', 'sv_SE', 'sw_KE',
'ta_IN', 'te_IN', 'th_TH', 'tl_XX', 'uk_UA', 'ur_PK', 'xh_ZA', 'gl_ES',
'sl_SI'])
```

[ ]:
```python
from tqdm import tqdm

def translate_by_mBart(model, tokenizer, src_path, src_lang, ref_lang,␣
 ↪batch_size, device='cpu', sub_set=None):
    # read file and extract all the sentences
    src_sents = read_file(src_path)
    if sub_set:
        # for debug
        src_sents = src_sents[:sub_set]
    tokenizer.src_lang = src_lang
    translated_sents = []
    model.to(device)
    for i in tqdm(range(0, len(src_sents), batch_size)):
        batch = src_sents[i:i+batch_size]
        encoded_batch_sentences = tokenizer(batch, return_tensors="pt",␣
 ↪padding=True).to(device)
        generated_batch_tokens = model.generate(**encoded_batch_sentences,␣
 ↪forced_bos_token_id=tokenizer.lang_code_to_id[ref_lang]).to('cpu')
        translated_sents += tokenizer.batch_decode(generated_batch_tokens,␣
 ↪skip_special_tokens=True)
    model.to('cpu')
    return translated_sents
```

- MarianMT

[ ]:
```python
from transformers import AutoTokenizer, MarianMTModel

def translate_by_MarianMT(src_lang, ref_lang, src_path, batch_size,␣
 ↪device='cpu', sub_set=None):
    # load model and tokenizer
    model_name = f"Helsinki-NLP/opus-mt-{src_lang}-{ref_lang}"
    model = MarianMTModel.from_pretrained(model_name)
    tokenizer = AutoTokenizer.from_pretrained(model_name)

    # read file and extract all the sentences
    src_sents = read_file(src_path)
    if sub_set:
        # for debug
        src_sents = src_sents[:sub_set]

    translated_sents = []
    model.to(device)
    for i in tqdm(range(0, len(src_sents), batch_size)):
```

```
        batch = src_sents[i:i+batch_size]
        encoded_batch_sentences = tokenizer(batch, return_tensors="pt",␣
 ↪padding=True).to(device)
        generated_ids = model.generate(**encoded_batch_sentences).to('cpu')
        encoded_batch_sentences.to('cpu')
        translated_sents += tokenizer.batch_decode(generated_ids,␣
 ↪skip_special_tokens=True)
    model.to('cpu')
    return translated_sents
```

- Compute the performance

[ ]:
```
import statistics

# evaluate the performance
def performance(refs, hyps, tokenizer=lambda s : s.split()):
    bleu_scores = []
    for ref, hyp in zip(refs, hyps):
        bleu_scores.append(bleu_score(ref, hyp, 2, tokenizer))
    return statistics.mean(bleu_scores)
```

- All test

[ ]:
```
hyps_mBart = {}
hyps_MarianMT = {}
refs = {}
per_mBart = {}
per_MarianMT = {}

lang_pair = [('cs','en'), ('de', 'en'), ('en','cs'), ('en','de'), ('en','fi'),␣
 ↪('en','ru'),('fi','en'), ('ru','en')]
lang_code = {'cs':'cs_CZ', 'en':'en_XX', 'de':'de_DE', 'ru':'ru_RU', 'fi':
 ↪'fi_FI'}
```

[ ]:
```
# start to translate (too long)

for h, r in lang_pair:
    key = f'{h}_{r}'
    print(f"translation from {h} to {r}:")
    # translate (2 ways)
    hyps_mBart[key] = translate_by_mBart(model_mBart, tokenizer_mBart, f"/
 ↪content/test/newstest2015-{h}{r}-src.{h}.sgm", lang_code[h], lang_code[r],␣
 ↪50, device=device)
    hyps_MarianMT[key] = translate_by_MarianMT(h, r, f"/content/test/
 ↪newstest2015-{h}{r}-src.{h}.sgm", 50, device=device)
    # read reference file
    refs[key] = read_file(f"/content/test/newstest2015-{h}{r}-ref.{r}.sgm")
    # compute performance
```

```
    per_mBart[key] = performance(refs[key], hyps_mBart[key])
    per_MarianMT[key] = performance(refs[key], hyps_MarianMT[key])
```

[ ]:
```
import pandas as pd

data = {
    'hyp_ref': lang_pair,
    'performance_mBart': per_mBart.values(),
    'performance_MarianMT': per_MarianMT.values()
}
df = pd.DataFrame(data)
df
```

[ ]:

	hyp_ref	performance_mBart	performance_MarianMT
0	(cs, en)	0.380930	0.371851
1	(de, en)	0.420352	0.425130
2	(en, cs)	0.267206	0.300128
3	(en, de)	0.362379	0.385861
4	(en, fi)	0.224777	0.348175
5	(en, ru)	0.331373	0.335207
6	(fi, en)	0.320570	0.352416
7	(ru, en)	0.406453	0.385352

Based on the data in the above, we can conclude the performance of the two models in different translation tasks. On the whole, the average Bleu score of MarianMT (0.367) is slightly higher than that of mBart (0.346), which shows that MarianMT is more advantageous in these language pairs on the whole. In details, some language pairs such as (cs, en) and (ru, en), mBart has higher Bleu scores than MarianMT, while in other language pairs such as (de, en) and (en, de), MarianMT has slightly higher Bleu scores, and in particular in the pair (en, fi), MarianMT's scores are significantly higher than those of mBart.

This is due to the differences in model architectures between mBart and MarianMT. mBart adopts a many-to-many architecture, which is capable of handling translation tasks in multiple languages and learning translation relations between multiple languages in a single model, which gives it high performance in multilingual environments. MarianMT, on the other hand, is designed to be fine-tuned for specific source and target languages, ensuring that each model focuses on the translation of specific language pairs and is suitable for specific tasks that require high-quality translation. Overall, mBart is more flexible and suitable for multilingual scenarios, whereas MarianMT excels in language pair-specific specialization.

3. Explain on an example why such permutations will never decrease the Bleu score.

Because Bleu score is mainly based on n-gram matches to calculate the score, different number of n-gram matches will affect the final Bleu score, but the order of the n-grams will not affect the number of matches, that's why it won't decrease due to different permutations. Here is an example:

[ ]:
```
# given my example reference and hypothesis sentence
ref = 'I believe I can learn NLP well'
hyp = 'I am confident that I can learn NLP well'
```

[ ]:
```
original_bleuscore = bleu_score(ref, hyp, 2)
original_bleuscore
```

[ ]: 0.5773502691896257

[ ]:
```
# create bigrams
ref_bigrams = create_ngrams(ref.split(), 2)
print(ref_bigrams)
hyp_bigrams = create_ngrams(hyp.split(), 2)
print(hyp_bigrams)
```

```
[('I', 'believe'), ('believe', 'I'), ('I', 'can'), ('can', 'learn'), ('learn',
'NLP'), ('NLP', 'well')]
[('I', 'am'), ('am', 'confident'), ('confident', 'that'), ('that', 'I'), ('I',
'can'), ('can', 'learn'), ('learn', 'NLP'), ('NLP', 'well')]
```

[ ]:
```
# examine bigram mismatches, if they are add a vertical bar
for i, bigram in enumerate(hyp_bigrams):
    if bigram not in ref_bigrams:
        hyp_bigrams[i] = (bigram[0], '|', bigram[1])

print(hyp_bigrams)
```

```
[('I', '|', 'am'), ('am', '|', 'confident'), ('confident', '|', 'that'),
('that', '|', 'I'), ('I', 'can'), ('can', 'learn'), ('learn', 'NLP'), ('NLP',
'well')]
```

[ ]:
```
# get new hypothesis sentence
new_hyp = []
for item in hyp_bigrams:
    new_hyp += (item[:-1])
new_hyp.append(hyp.split()[-1])
new_hyp = ' '.join(new_hyp)
print(new_hyp)
```

```
I | am | confident | that | I can learn NLP well
```

[ ]:
```
from itertools import permutations

# get permutated hypothesis sentences
per_hyps = [' '.join(item) for item in permutations(new_hyp.split(' | '))]
```

[ ]:
```
import random

random_sample = random.sample(per_hyps, 10)
random_sample
```

[ ]:
```
['am I that confident I can learn NLP well',
 'I am confident that I can learn NLP well',
```

```
 'am I confident that I can learn NLP well',
 'confident I am I can learn NLP well that',
 'am I confident I can learn NLP well that',
 'confident I that am I can learn NLP well',
 'I am that confident I can learn NLP well',
 'confident I am that I can learn NLP well',
 'I am confident I can learn NLP well that',
 'confident I that I can learn NLP well am']
```

[ ]:
```
# compute new bleu scores
new_bleuscores = []
for item in per_hyps:
    new_bleuscores.append(bleu_score(ref, item, 2))
```

[ ]:
```
random_sample = random.sample(new_bleuscores, 10)
print(random_sample)
```

```
0.5773502691896257
0.5773502691896257
0.5773502691896257
0.5773502691896257
0.5773502691896257
0.5773502691896257
0.5773502691896257
0.5773502691896257
0.5773502691896257
0.5773502691896257
```

4. Given a sentence with n words and b bigram mismatches, how many sentences can you generate with this principle. Compute the number of sentences you will obtain on the WMT'15 test set.

For a given sentence, if n is its length, which means the total number of bigrams is $(n-1)$, and b is the number of bigram mismatches, we can get the number of bigram matches is $(n-1-b)$, then calculate the number of sentences it can generate with this formula: $(n-(n-1-b))! = (b+1)!$

[ ]:
```
import math

# compute the number of sentences will obtain on the WMT'15
def nb_sents(references, hypothesis):
    nb_sents = 0
    for ref , hyp in zip(references, hypothesis):
        # create bigrams
        bigram_ref = create_ngrams(ref.split(), 2)
        bigram_hyp = create_ngrams(hyp.split(), 2)
        # b bigram mismatches for one sentence
        b = sum([item not in bigram_ref for item in bigram_hyp])
        # accumulate the number of sentences we will obtain on the whole test␣
 ↪set
```

```
        nb_sents += math.factorial(b + 1)
    return nb_sents
```

[ ]:
```
count_sents_mBart = {}
count_sents_MarianMT = {}
for h, r in lang_pair:
    key = f"{h}_{r}"
    count_sents_mBart[key] = nb_sents(refs[key], hyps_mBart[key])
    count_sents_MarianMT[key] = nb_sents(refs[key], hyps_MarianMT[key])
```

[ ]:
```
import pandas as pd

data = {
    'hyp_ref': lang_pair,
    'mBart': count_sents_mBart.values(),
    'MarianMT': count_sents_MarianMT.values()
}
df = pd.DataFrame(data)
# using scientific notation to record numbers
df['mBart'] = df['mBart'].apply(lambda x: "{:.2e}".format(x))
df['MarianMT'] = df['MarianMT'].apply(lambda x: "{:.2e}".format(x))
df
```

[ ]:

	hyp_ref	mBart	MarianMT
0	(cs, en)	3.10e+64	8.25e+90
1	(de, en)	7.11e+74	4.05e+76
2	(en, cs)	1.61e+68	1.61e+68
3	(en, de)	3.15e+85	3.15e+85
4	(en, fi)	1.58e+66	6.04e+52
5	(en, ru)	8.25e+90	2.93e+188
6	(fi, en)	3.65e+94	1.27e+89
7	(ru, en)	4.05e+76	8.32e+81

5. Why this result question the use of Bleu as an evaluation metric.

Because word order is crucial for semantic expression in many languages, but the Bleu score has the property of being "independent of the order of the n-gram matches", which allows many ungrammatical sentences to get high scores and leads to a misleading result. Therefore, Bleu is not suitable as the only indicator for evaluating translation quality.

6. sacreBleu is an implementation of Bleu that aims to provide "hassle-free computation of shareable, comparable, and reproducible Bleu scores". Evaluate the two previous systems using sacreBleu. What can you conclude ?

[ ]:
```
!pip install sacrebleu > log.txt
!pip install evaluate > log.txt
```

[ ]:
```
import evaluate
```

```
word_sacrebleuscores_mBart = {}
word_sacrebleuscores_MarianMT = {}
sacrebleu = evaluate.load("sacrebleu")
for h, r in lang_pair:
    key = f"{h}_{r}"
    word_sacrebleuscores_mBart[key] = sacrebleu.
 ↪compute(predictions=hyps_mBart[key], references=refs[key])['score']
    word_sacrebleuscores_MarianMT[key] = sacrebleu.
 ↪compute(predictions=hyps_MarianMT[key], references=refs[key])['score']
```

[ ]:
```
data = {
    'hyp_ref': lang_pair,
    'performance_mBart': word_sacrebleuscores_mBart.values(),
    'performance_MarianMT': word_sacrebleuscores_MarianMT.values()
}
df = pd.DataFrame(data)
df
```

[ ]:

	hyp_ref	performance_mBart	performance_MarianMT
0	(cs, en)	30.396764	29.232791
1	(de, en)	34.004313	34.044129
2	(en, cs)	21.215161	23.700135
3	(en, de)	28.666893	31.070945
4	(en, fi)	17.067169	27.603179
5	(en, ru)	27.142213	27.032437
6	(fi, en)	24.073603	25.816301
7	(ru, en)	31.972505	29.879974

From the sacreBleu scores above we can conclude the performance of these two models mBART and MarianMT. Overall, mBART has a slight advantage over MarianMT in certain language pairs (e.g., cs_en, ru_en), which shows that it has a stronger ability to generate English on these groups of language pairs, But MarianMT is better than mBART on some other language pairs (e.g., en_fi, en_cs). The difference between the two models in the (de_en) and (en_ru) directions is very small, showing that they perform similarly on these two language pairs. In contrast, MarianMT significantly outperforms mBART in the (en_fi) direction (27.6 vs. 17.1), which shows that MarianMT is significantly stronger in terms of translation quality from English to Finnish, while mBART underperforms in this direction.

7. Using sacreBleu and your own implementation of Bleu compute the score achieved :

- subword tokenization

[ ]:
```
from transformers import BertTokenizer

# load mBert tokenizer
tokenizer = BertTokenizer.from_pretrained("bert-base-multilingual-cased")
```

[ ]:
```
subword_bleuscores_mBart = {}
subword_bleuscores_MarianMT = {}
for h, r in lang_pair:
    key = f"{h}_{r}"
    subword_bleuscores_mBart[key] = performance(refs[key], hyps_mBart[key],␣
 ↪tokenizer=tokenizer.tokenize)
    subword_bleuscores_MarianMT[key] = performance(refs[key],␣
 ↪hyps_MarianMT[key], tokenizer=tokenizer.tokenize)
```

[ ]:
```
subword_sacrebleuscores_mBart = {}
subword_sacrebleuscores_MarianMT = {}
for h, r in lang_pair:
    key = f"{h}_{r}"
    subword_sacrebleuscores_mBart[key] = sacrebleu.compute(predictions=[' '.
 ↪join(tokenizer.tokenize(sent)) for sent in hyps_mBart[key]], references=[' '.
 ↪join(tokenizer.tokenize(sent)) for sent in refs[key]], force=True,␣
 ↪tokenize=None)['score']
    subword_sacrebleuscores_MarianMT[key] = sacrebleu.compute(predictions=[' '.
 ↪join(tokenizer.tokenize(sent)) for sent in hyps_MarianMT[key]],␣
 ↪references=[' '.join(tokenizer.tokenize(sent)) for sent in refs[key]],␣
 ↪force=True, tokenize=None)['score']
```

- caracter tokenization

[ ]:
```
character_bleuscores_mBart = {}
character_bleuscores_MarianMT = {}
for h, r in lang_pair:
    key = f"{h}_{r}"
    character_bleuscores_mBart[key] = performance(refs[key], hyps_mBart[key],␣
 ↪tokenizer=lambda s : s)
    character_bleuscores_MarianMT[key] = performance(refs[key],␣
 ↪hyps_MarianMT[key], tokenizer=lambda s : s)
```

[ ]:
```
character_sacrebleuscores_mBart = {}
character_sacrebleuscores_MarianMT = {}
for h, r in lang_pair:
    key = f"{h}_{r}"
    character_sacrebleuscores_mBart[key] = sacrebleu.
 ↪compute(predictions=hyps_mBart[key], references=refs[key],␣
 ↪tokenize='char')['score']
    character_sacrebleuscores_MarianMT[key] = sacrebleu.
 ↪compute(predictions=hyps_MarianMT[key], references=refs[key],␣
 ↪tokenize='char')['score']
```

- For mBART

[ ]:
```
data = {
    'hyp_ref': lang_pair,
```

```
    'word_Bleu': per_mBart.values(),
    'subword_Bleu': subword_bleuscores_mBart.values(),
    'character_Bleu': character_bleuscores_mBart.values(),
    'word_Sacrebleu': word_sacrebleuscores_mBart.values(),
    'subword_Sacrebleu': subword_sacrebleuscores_mBart.values(),
    'character_Sacrebleu': character_sacrebleuscores_mBart.values()
}
df_mBart = pd.DataFrame(data)
df_mBart
```

[ ]:

	hyp_ref	word_Bleu	subword_Bleu	character_Bleu	word_Sacrebleu	subword_Sacrebleu	character_Sacrebleu
0	(cs, en)	0.380930	0.460977	0.720629	30.396764	38.491523	61.949720
1	(de, en)	0.420352	0.501891	0.750272	34.004313	43.035842	65.374474
2	(en, cs)	0.267206	0.426904	0.683626	21.215161	47.924923	56.217529
3	(en, de)	0.362379	0.458753	0.739520	28.666893	44.365647	63.616831
4	(en, fi)	0.224777	0.378312	0.690271	17.067169	44.418577	57.291907
5	(en, ru)	0.331373	0.460530	0.710566	27.142213	49.786564	60.642342
6	(fi, en)	0.320570	0.399354	0.694316	24.073603	32.574267	57.915550
7	(ru, en)	0.406453	0.476364	0.742836	31.972505	38.296699	64.116429

- For MarianMT

[ ]:
```
data = {
    'hyp_ref': lang_pair,
    'word_Bleu': per_MarianMT.values(),
    'subword_Bleu': subword_bleuscores_MarianMT.values(),
    'character_Bleu': character_bleuscores_MarianMT.values(),
    'word_Sacrebleu': word_sacrebleuscores_MarianMT.values(),
    'subword_Sacrebleu': subword_sacrebleuscores_MarianMT.values(),
    'character_Sacrebleu': character_sacrebleuscores_MarianMT.values()
}
df_MarianMT = pd.DataFrame(data)
df_MarianMT
```

[ ]:

	hyp_ref	word_Bleu	subword_Bleu	character_Bleu	word_Sacrebleu	subword_Sacrebleu	character_Sacrebleu
0	(cs, en)	0.371851	0.459340	0.723244	29.232791	38.491523	61.949720
1	(de, en)	0.425130	0.508805	0.754834	34.044129	43.035842	65.374474
2	(en, cs)	0.300128	0.462772	0.699709	23.700135	47.924923	56.217529

3	(en, de)	0.385861	0.486350	0.750117	31.070945
4	(en, fi)	0.348175	0.510284	0.751089	27.603179
5	(en, ru)	0.335207	0.471489	0.714650	27.032437
6	(fi, en)	0.352416	0.438676	0.716517	25.816301
7	(ru, en)	0.385352	0.463256	0.732696	29.879974

	subword_Sacrebleu	character_Sacrebleu
0	38.603239	61.893634
1	43.765782	65.751398
2	50.653509	58.592193
3	47.515898	65.445066
4	54.494192	65.461260
5	51.368148	60.359567
6	36.122189	60.487375
7	36.549705	61.986108

How can you explain these results ?

From the above data we can conclude that the scores of both Bleu and sacreBleu increase when the tokenized units become smaller, this is because when we split the sentences into smaller units, we are able to match more n-grams and thus increase the scores. In model comparisons, MarianMT slightly better than mBART on word and subword, especially on word_sacreBLEU where the gap is obvious, e.g., on (en, fi) mBART scores 17.07 while MarianMT reaches 27.60. In general, MarianMT scores higher than mBART on most language pairs, especially on subword and character levels, which indicates that it is more compatible with different tokenization levels when dealing with multilingual translation tasks.