C4_W1_Ungraded_Lab_3_Bleu_Score

December 17, 2023

1 Calculating the Bilingual Evaluation Understudy (BLEU) score: Ungraded Lab

In this ungraded lab, we will implement a popular metric for evaluating the quality of machine-translated text: the BLEU score proposed by Kishore Papineni, et al. in their 2002 paper "BLEU: a Method for Automatic Evaluation of Machine Translation", the BLEU score works by comparing "candidate" text to one or more "reference" translations. The result is better the closer the score is to 1. Let's see how to get this value in the following sections.

2 Part 1: BLEU Score

2.1 1.1 Importing the Libraries

We will first start by importing the Python libraries we will use in the first part of this lab. For learning, we will implement our own version of the BLEU Score using Numpy. To verify that our implementation is correct, we will compare our results with those generated by the SacreBLEU library. This package provides hassle-free computation of shareable, comparable, and reproducible BLEU scores. It also knows all the standard test sets and handles downloading, processing, and tokenization.

[ ]:
```
import numpy as np                    # import numpy to make numerical␣
 ↪computations.
import nltk                           # import NLTK to handle simple NL tasks␣
 ↪like tokenization.
nltk.download("punkt")
import math
from nltk.util import ngrams
from collections import Counter       # import a counter.
!pip3 install 'sacrebleu'             # install the sacrebleu package.
import sacrebleu                      # import sacrebleu in order compute the␣
 ↪BLEU score.
import matplotlib.pyplot as plt       # import pyplot in order to make some␣
 ↪illustrations.
```

2.2 1.2 Defining the BLEU Score

You have seen the a way for calculating the BLEU score in this week's lectures. More formally, we can express the BLEU score as:

$$BLEU = BP \times \left(\prod_{i=1}^{4} precision_i\right)^{(1/4)}$$

We used the productory and the denominator of the exponent as 4 because we are assuming that we are going to work up to 4-grams. The Brevity Penalty defined as an exponential decay:

$$BP = min\left(1, e^{(1-(ref/cand))}\right)$$

In the previous equation, ref and $cand$ refers to the length or count of words in the reference and candidate translations. The brevity penaly BP helps to handle very short translations.

The precision is defined as :

$$precision_i = \frac{\sum_{snt \in cand} \sum_{i \in snt} min\left(m^i_{cand}, m^i_{ref}\right)}{w^i_t}$$

where:

- m^i_{cand}, is the count of i-gram in candidate matching the reference translation.
- m^i_{ref}, is the count of i-gram in the reference translation.
- w^i_t, is the total number of i-grams in candidate translation.

2.3 1.3 Explaining the BLEU score

2.3.1 Brevity Penalty (example):

[ ]:
```
ref_length = np.ones(100)
can_length = np.linspace(1.5, 0.5, 100)
x = ref_length / can_length
y = 1 - x
y = np.exp(y)
y = np.minimum(np.ones(y.shape), y)

# Code for in order to make the plot
fig, ax = plt.subplots(1)
lines = ax.plot(x, y)
ax.set(
    xlabel="Ratio of the length of the reference to the candidate text",
    ylabel="Brevity Penalty",
)
plt.show()
```

The brevity penalty penalizes generated translations that are too short compared to the closest reference length with an exponential decay. The brevity penalty compensates for the fact that the BLEU score has no recall term.

2.3.2 N-Gram Precision (example):

[ ]:
```
# Mocked dataset showing the precision for different n-grams
data = {"1-gram": 0.8, "2-gram": 0.7, "3-gram": 0.6, "4-gram": 0.5}
names = list(data.keys())
values = list(data.values())

fig, ax = plt.subplots(1)
bars = ax.bar(names, values)
ax.set(ylabel="N-gram precision")

plt.show()
```

The n-gram precision counts how many unigrams, bigrams, trigrams, and four-grams (i=1,...,4) match their n-gram counterpart in the reference translations. This term acts as a precision metric. Unigrams account for adequacy while longer n-grams account for fluency of the translation. To avoid overcounting, the n-gram counts are clipped to the maximal n-gram count occurring in the reference (m_n^{ref}). Typically precision shows exponential decay with the degree of the n-gram.

2.3.3 N-gram BLEU score (example):

[ ]:
```
# Mocked dataset showing the precision multiplied by the BP for different␣
↪n-grams
data = {"1-gram": 0.8, "2-gram": 0.77, "3-gram": 0.74, "4-gram": 0.71}
names = list(data.keys())
values = list(data.values())

fig, ax = plt.subplots(1)
bars = ax.bar(names, values)
ax.set(ylabel="Modified N-gram precision")

plt.show()
```

When the n-gram precision is multiplied by the BP, then the exponential decay of n-grams is almost fully compensated. The BLEU score corresponds to a geometric average of this modified n-gram precision.

2.4 1.4 Example Calculations of the BLEU score

In this example we will have a reference translation and 2 candidates translations. We will tokenize all sentences using the NLTK package introduced in Course 2 of this NLP specialization.

[ ]:
```
reference = "The NASA Opportunity rover is battling a massive dust storm on␣
 ↪planet Mars."
candidate_1 = "The Opportunity rover is combating a big sandstorm on planet␣
 ↪Mars."
candidate_2 = "A NASA rover is fighting a massive storm on planet Mars."

tokenized_ref = nltk.word_tokenize(reference.lower())
tokenized_cand_1 = nltk.word_tokenize(candidate_1.lower())
tokenized_cand_2 = nltk.word_tokenize(candidate_2.lower())

print(f"{reference} -> {tokenized_ref}")
print("\n")
print(f"{candidate_1} -> {tokenized_cand_1}")
print("\n")
print(f"{candidate_2} -> {tokenized_cand_2}")
```

2.4.1 STEP 1: Computing the Brevity Penalty

[ ]:
```
def brevity_penalty(candidate, reference):
    ref_length = len(reference)
    can_length = len(candidate)

    # Brevity Penalty
    if ref_length < can_length: # if reference length is less than candidate␣
 ↪length
        BP = 1 # set BP = 1
    else:
        penalty = 1 - (ref_length / can_length) # else set BP=exp(1-(ref_length/
 ↪can_length))
        BP = np.exp(penalty)

    return BP
```

2.4.2 STEP 2: Computing the Precision

[ ]:
```
def clipped_precision(candidate, reference):
    """
    Clipped precision function given a original and a machine translated␣
 ↪sentences
    """

    clipped_precision_score = []

    for i in range(1, 5):
```

```
        ref_n_gram = Counter(ngrams(reference,i))
        cand_n_gram = Counter(ngrams(candidate,i))

        c = sum(cand_n_gram.values())

        for j in cand_n_gram: # for every n-gram up to 4 in candidate text
            if j in ref_n_gram: # check if it is in the reference n-gram
                if cand_n_gram[j] > ref_n_gram[j]: # if the count of the␣
↪candidate n-gram is bigger
                                                   # than the corresponding␣
↪count in the reference n-gram,
                    cand_n_gram[j] = ref_n_gram[j] # then set the count of the␣
↪candidate n-gram to be equal
                                                   # to the reference n-gram
            else:
                cand_n_gram[j] = 0 # else set the candidate n-gram equal to zero


        clipped_precision_score.append(sum(cand_n_gram.values())/c)

    weights =[0.25]*4

    s = (w_i * math.log(p_i) for w_i, p_i in zip(weights,␣
↪clipped_precision_score))

    s = math.exp(math.fsum(s))
    return s
```

2.4.3 STEP 3: Computing the BLEU score

[ ]:
```
def bleu_score(candidate, reference):
    BP = brevity_penalty(candidate, reference)
    precision = clipped_precision(candidate, reference)
    return BP * precision
```

2.4.4 STEP 4: Testing with our Example Reference and Candidates Sentences

[ ]:
```
print(
    "Results reference versus candidate 1 our own code BLEU: ",
    round(bleu_score(tokenized_cand_1, tokenized_ref) * 100, 1),
)

print(
    "Results reference versus candidate 2 our own code BLEU: ",
```

```
    round(bleu_score(tokenized_cand_2, tokenized_ref) * 100, 1),
)
```

2.4.5 STEP 5: Comparing the Results from our Code with the Sacrebleu Library

[ ]:
```
print(
    "Results reference versus candidate 1 sacrebleu library sentence BLEU: ",
    round(sacrebleu.sentence_bleu(candidate_1, [reference]).score, 1),
)
print(
    "Results reference versus candidate 2 sacrebleu library sentence BLEU: ",
    round(sacrebleu.sentence_bleu(candidate_2, [reference]).score, 1),
)
```

3 Part 2: BLEU computation on a corpus

3.1 Loading Data Sets for Evaluation Using the BLEU Score

In this section, we will show a simple pipeline for evaluating machine translated text. Due to storage and speed constraints, we will not be using our own model in this lab (you'll get to do that in the assignment!). Instead, we will be using Google Translate to generate English to German translations and we will evaluate it against a known evaluation set. There are three files we will need:

1. A source text in English. In this lab, we will use the first 1671 words of the wmt19 evaluation dataset downloaded via SacreBLEU. We just grabbed a subset because of limitations in the number of words that can be translated using Google Translate.
2. A reference translation to German of the corresponding first 1671 words from the original English text. This is also provided by SacreBLEU.
3. A candidate machine translation to German from the same 1671 words. This is generated by feeding the source text to a machine translation model. As mentioned above, we will use Google Translate to generate the translations in this file.

With that, we can now compare the reference and candidate translation to get the BLEU Score.

[ ]:
```
# Loading the raw data
wmt19_src = open("data/wmt19_src.txt", "r")
wmt19_src_1 = wmt19_src.read()
wmt19_src.close()

wmt19_ref = open("data/wmt19_ref.txt", "r")
wmt19_ref_1 = wmt19_ref.read()
wmt19_ref.close()

wmt19_can = open("data/wmt19_can.txt", "r")
```

```
wmt19_can_1 = wmt19_can.read()
wmt19_can.close()

tokenized_corpus_src = nltk.word_tokenize(wmt19_src_1.lower())
tokenized_corpus_ref = nltk.word_tokenize(wmt19_ref_1.lower())
tokenized_corpus_cand = nltk.word_tokenize(wmt19_can_1.lower())
```

Inspecting the first sentence of the data.

[ ]:
```
print("English source text:")
print("\n")
print(f"{wmt19_src_1[0:170]} -> {tokenized_corpus_src[0:30]}")
print("\n")
print("German reference translation:")
print("\n")
print(f"{wmt19_ref_1[0:219]} -> {tokenized_corpus_ref[0:35]}")
print("\n")
print("German machine translation:")
print("\n")
print(f"{wmt19_can_1[0:199]} -> {tokenized_corpus_cand[0:29]}")
```

[ ]:
```
print(
    "Results reference versus candidate 1 our own BLEU implementation: ",
    round(bleu_score(tokenized_corpus_cand, tokenized_corpus_ref) * 100, 1),
)
```

[ ]:
```
print(
    "Results reference versus candidate 1 sacrebleu library BLEU: ",
    round(sacrebleu.sentence_bleu(wmt19_can_1, [wmt19_ref_1]).score, 1),
)
```

BLEU Score Interpretation on a Corpus

Score	Interpretation
< 10	Almost useless
10 - 19	Hard to get the gist
20 - 29	The gist is clear, but has significant grammatical errors
30 - 40	Understandable to good translations
40 - 50	High quality translations
50 - 60	Very high quality, adequate, and fluent translations
> 60	Quality often better than human

From the table above (taken here), we can see the translation is high quality (*if you see "Hard to get the gist", please open your workspace, delete* ***wmt19_can.txt*** *and get the latest version via the Lab Help button*). Moreover, the results of our coded BLEU score are almost identical to those of the SacreBLEU package.