Evaluation

May 27, 2020

1 Evaluation of BERT and DistilBERT

1.1 METEOR and GLEU scores

1. Functions for calculating GLEU and METEOR for two outputs at the same time
2. Calculating GLEU scores
3. Calculating METEOR scores
4. Appending to dataframe, getting 50 samples

[7]:

```python
import nltk
import json
import re
import numpy
import pandas as pd
import string

from nltk.translate import gleu_score
from nltk.stem.porter import PorterStemmer
from nltk.corpus import wordnet
from itertools import chain, product
```

1.2 1. Functions for calculating GLEU and METEOR for two outputs at the same time

The original GLEU function compares one translation to several golden standards, but we want to compare the performance of both BERT and DistilBERT at the same time to the golden standard.

GLEU takes an input of a split string. METEOR takes just strings.

1.2.1 1.1 GLEU functions

[8]:

```python
# Function that expands the gleu sentence calculations to be done on more than␣
 ↪1 input texts
def gleu_lists_no_df(golden_standard, text1, text2):

    # n-gram lists for text1 and text2
    ngram_list = [[],[]]

    # Append the calculated gleu scores to a list for both text1 and text2
    ngram_list[0].append(nltk.translate.gleu_score.
 ↪sentence_gleu([golden_standard], text1))
    ngram_list[1].append(nltk.translate.gleu_score.
 ↪sentence_gleu([golden_standard], text2))

    # This one returns ngram_list directly for the function that makes a row␣
 ↪per answer
    return(ngram_list)

# Function that takes columns as input and outputs a dataframe of the results␣
 ↪per model
def gleu_more(gold1, gold2, berts, distilberts):
    # Initiate results dataframe
    results = pd.DataFrame()

    # Loop over the rows in the dataset
    for i in range(0,len(gold1)):
        # Define the inputs to the bleu function from the dataset
        # One row per input
        gold_1 = gold1[i]
        # Clean of punctuation
        gold_1 = gold_1.translate(str.maketrans('', '', string.punctuation))
        gold_2 = gold2[i]
        #gold_2 = gold_2.translate(str.maketrans('', '', string.punctuation))
        bert = berts[i]
        bert = bert.translate(str.maketrans('', '', string.punctuation))
        distil = distilberts[i]
        distil = distil.translate(str.maketrans('', '', string.punctuation))

        # Get the gleu scores per line in dataframe with the predefined␣
 ↪function for 1st golden answers
        ngram_list1 = gleu_lists_no_df(gold_1.split(), bert.split(), distil.
 ↪split())

        # Now get the glue scores for 2nd golden answer in case there are more␣
 ↪than 1 ways to answer the question
        # For missing values in Gold2, return all 0-s
        if pd.isnull(gold_2):
            ngram_list2 = [[0],[0]]
        else:
            ngram_list2 = gleu_lists_no_df(gold_2.split(), bert.split(), distil.
 ↪split())

        # Get the highest value per model in terms of them matching best to␣
 ↪gold1 or gold2
        ngram_max = [max(i, j) for i, j in zip(ngram_list1, ngram_list2)]
        # Append to dataframe, index is model name + iteration
        dff = pd.DataFrame(ngram_max, index =['0', '1'])

        # Append results to the dataframe
        results = results.append(dff)

    return(results)
```

1.2.2 1.2 METEOR functions

We are using other parameter values for meteor score calculations so I will add the whole code on compliling meteor scores together with how the parameter values were changed. Meteor score code retrieved from: https://www.nltk.org/_modules/nltk/translate/meteor_score.html

[20]:

```
from nltk.stem.porter import PorterStemmer
from nltk.corpus import wordnet
from itertools import chain, product


def _generate_enums(hypothesis, reference, preprocess=str.lower):
    """
    Takes in string inputs for hypothesis and reference and returns
    enumerated word lists for each of them

    :param hypothesis: hypothesis string
    :type hypothesis: str
    :param reference: reference string
    :type reference: str
    :preprocess: preprocessing method (default str.lower)
    :type preprocess: method
    :return: enumerated words list
    :rtype: list of 2D tuples, list of 2D tuples
    """
    hypothesis_list = list(enumerate(preprocess(hypothesis).split()))
    reference_list = list(enumerate(preprocess(reference).split()))
    return hypothesis_list, reference_list


def exact_match(hypothesis, reference):
    """
    matches exact words in hypothesis and reference
    and returns a word mapping based on the enumerated
    word id between hypothesis and reference

    :param hypothesis: hypothesis string
    :type hypothesis: str
    :param reference: reference string
    :type reference: str
    :return: enumerated matched tuples, enumerated unmatched hypothesis tuples,
             enumerated unmatched reference tuples
    :rtype: list of 2D tuples, list of 2D tuples,  list of 2D tuples
    """
    hypothesis_list, reference_list = _generate_enums(hypothesis, reference)
    return _match_enums(hypothesis_list, reference_list)



def _match_enums(enum_hypothesis_list, enum_reference_list):
    """
    matches exact words in hypothesis and reference and returns
    a word mapping between enum_hypothesis_list and enum_reference_list
    based on the enumerated word id.

    :param enum_hypothesis_list: enumerated hypothesis list
    :type enum_hypothesis_list: list of tuples
    :param enum_reference_list: enumerated reference list
    :type enum_reference_list: list of 2D tuples
    :return: enumerated matched tuples, enumerated unmatched hypothesis tuples,
             enumerated unmatched reference tuples
    :rtype: list of 2D tuples, list of 2D tuples,  list of 2D tuples
    """
    word_match = []
    for i in range(len(enum_hypothesis_list))[::-1]:
        for j in range(len(enum_reference_list))[::-1]:
            if enum_hypothesis_list[i][1] == enum_reference_list[j][1]:
                word_match.append(
                    (enum_hypothesis_list[i][0], enum_reference_list[j][0])
                )
                (enum_hypothesis_list.pop(i)[1], enum_reference_list.pop(j)[1])
                break
    return word_match, enum_hypothesis_list, enum_reference_list


def _enum_stem_match(
    enum_hypothesis_list, enum_reference_list, stemmer=PorterStemmer()
):
    """
    Stems each word and matches them in hypothesis and reference
    and returns a word mapping between enum_hypothesis_list and
    enum_reference_list based on the enumerated word id. The function also
    returns a enumerated list of unmatched words for hypothesis and reference.

    :param enum_hypothesis_list:
```

```
    :type enum_hypothesis_list:
    :param enum_reference_list:
    :type enum_reference_list:
    :param stemmer: nltk.stem.api.StemmerI object (default PorterStemmer())
    :type stemmer: nltk.stem.api.StemmerI or any class that implements a stem␣
↪method
    :return: enumerated matched tuples, enumerated unmatched hypothesis tuples,
             enumerated unmatched reference tuples
    :rtype: list of 2D tuples, list of 2D tuples,  list of 2D tuples
    """
    stemmed_enum_list1 = [
        (word_pair[0], stemmer.stem(word_pair[1])) for word_pair in␣
↪enum_hypothesis_list
    ]

    stemmed_enum_list2 = [
        (word_pair[0], stemmer.stem(word_pair[1])) for word_pair in␣
↪enum_reference_list
    ]

    word_match, enum_unmat_hypo_list, enum_unmat_ref_list = _match_enums(
        stemmed_enum_list1, stemmed_enum_list2
    )

    enum_unmat_hypo_list = (
        list(zip(*enum_unmat_hypo_list)) if len(enum_unmat_hypo_list) > 0 else␣
↪[]
    )

    enum_unmat_ref_list = (
        list(zip(*enum_unmat_ref_list)) if len(enum_unmat_ref_list) > 0 else []
    )

    enum_hypothesis_list = list(
        filter(lambda x: x[0] not in enum_unmat_hypo_list, enum_hypothesis_list)
    )

    enum_reference_list = list(
        filter(lambda x: x[0] not in enum_unmat_ref_list, enum_reference_list)
    )

    return word_match, enum_hypothesis_list, enum_reference_list


def stem_match(hypothesis, reference, stemmer=PorterStemmer()):
    """
    Stems each word and matches them in hypothesis and reference
    and returns a word mapping between hypothesis and reference

    :param hypothesis:
    :type hypothesis:
    :param reference:
    :type reference:
    :param stemmer: nltk.stem.api.StemmerI object (default PorterStemmer())
    :type stemmer: nltk.stem.api.StemmerI or any class that
                   implements a stem method
    :return: enumerated matched tuples, enumerated unmatched hypothesis tuples,
             enumerated unmatched reference tuples
    :rtype: list of 2D tuples, list of 2D tuples,  list of 2D tuples
    """
    enum_hypothesis_list, enum_reference_list = _generate_enums(hypothesis,␣
 ↪reference)
    return _enum_stem_match(enum_hypothesis_list, enum_reference_list,␣
 ↪stemmer=stemmer)



def _enum_wordnetsyn_match(enum_hypothesis_list, enum_reference_list,␣
 ↪wordnet=wordnet):
    """
    Matches each word in reference to a word in hypothesis
    if any synonym of a hypothesis word is the exact match
    to the reference word.

    :param enum_hypothesis_list: enumerated hypothesis list
    :param enum_reference_list: enumerated reference list
    :param wordnet: a wordnet corpus reader object (default nltk.corpus.wordnet)
    :type wordnet: WordNetCorpusReader
    :return: list of matched tuples, unmatched hypothesis list, unmatched␣
 ↪reference list
    :rtype:  list of tuples, list of tuples, list of tuples

    """
    word_match = []
    for i in range(len(enum_hypothesis_list))[::-1]:
        hypothesis_syns = set(
            chain(
                *[
                    [
                        lemma.name()
                        for lemma in synset.lemmas()
                        if lemma.name().find("_") < 0
                    ]
                    for synset in wordnet.synsets(enum_hypothesis_list[i][1])
                ]
            )
        ).union({enum_hypothesis_list[i][1]})
        for j in range(len(enum_reference_list))[::-1]:
            if enum_reference_list[j][1] in hypothesis_syns:
                word_match.append(
                    (enum_hypothesis_list[i][0], enum_reference_list[j][0])
                )
                enum_hypothesis_list.pop(i), enum_reference_list.pop(j)
                break
    return word_match, enum_hypothesis_list, enum_reference_list


def wordnetsyn_match(hypothesis, reference, wordnet=wordnet):
    """
    Matches each word in reference to a word in hypothesis if any synonym
    of a hypothesis word is the exact match to the reference word.

    :param hypothesis: hypothesis string
    :param reference: reference string
    :param wordnet: a wordnet corpus reader object (default nltk.corpus.wordnet)
    :type wordnet: WordNetCorpusReader
    :return: list of mapped tuples
    :rtype: list of tuples
    """
    enum_hypothesis_list, enum_reference_list = _generate_enums(hypothesis,␣
 ↪reference)
    return _enum_wordnetsyn_match(
        enum_hypothesis_list, enum_reference_list, wordnet=wordnet
    )



def _enum_allign_words(
    enum_hypothesis_list, enum_reference_list, stemmer=PorterStemmer(),␣
 ↪wordnet=wordnet
):
    """
    Aligns/matches words in the hypothesis to reference by sequentially
    applying exact match, stemmed match and wordnet based synonym match.
    in case there are multiple matches the match which has the least number
    of crossing is chosen. Takes enumerated list as input instead of
    string input

    :param enum_hypothesis_list: enumerated hypothesis list
    :param enum_reference_list: enumerated reference list
    :param stemmer: nltk.stem.api.StemmerI object (default PorterStemmer())
    :type stemmer: nltk.stem.api.StemmerI or any class that implements a stem␣
 ↪method
    :param wordnet: a wordnet corpus reader object (default nltk.corpus.wordnet)
    :type wordnet: WordNetCorpusReader
    :return: sorted list of matched tuples, unmatched hypothesis list,
             unmatched reference list
    :rtype: list of tuples, list of tuples, list of tuples
    """
    exact_matches, enum_hypothesis_list, enum_reference_list = _match_enums(
        enum_hypothesis_list, enum_reference_list
    )

    stem_matches, enum_hypothesis_list, enum_reference_list = _enum_stem_match(
        enum_hypothesis_list, enum_reference_list, stemmer=stemmer
    )

    wns_matches, enum_hypothesis_list, enum_reference_list =␣
 ↪_enum_wordnetsyn_match(
        enum_hypothesis_list, enum_reference_list, wordnet=wordnet
    )

    return (
        sorted(
            exact_matches + stem_matches + wns_matches, key=lambda wordpair:␣
 ↪wordpair[0]
        ),
        enum_hypothesis_list,
        enum_reference_list,
    )


def allign_words(hypothesis, reference, stemmer=PorterStemmer(),␣
 ↪wordnet=wordnet):
    """
    Aligns/matches words in the hypothesis to reference by sequentially
    applying exact match, stemmed match and wordnet based synonym match.
    In case there are multiple matches the match which has the least number
    of crossing is chosen.

    :param hypothesis: hypothesis string
    :param reference: reference string
    :param stemmer: nltk.stem.api.StemmerI object (default PorterStemmer())
    :type stemmer: nltk.stem.api.StemmerI or any class that implements a stem␣
 ↪method
    :param wordnet: a wordnet corpus reader object (default nltk.corpus.wordnet)
    :type wordnet: WordNetCorpusReader
    :return: sorted list of matched tuples, unmatched hypothesis list, unmatched reference list
    :rtype: list of tuples, list of tuples, list of tuples
    """
    enum_hypothesis_list, enum_reference_list = _generate_enums(hypothesis, reference)
    return _enum_allign_words(
        enum_hypothesis_list, enum_reference_list, stemmer=stemmer, wordnet=wordnet
    )



def _count_chunks(matches):
    """
    Counts the fewest possible number of chunks such that matched unigrams
    of each chunk are adjacent to each other. This is used to caluclate the
    fragmentation part of the metric.

    :param matches: list containing a mapping of matched words (output of allign_words)
    :return: Number of chunks a sentence is divided into post allignment
    :rtype: int
    """
    i = 0
    chunks = 1
    while i < len(matches) - 1:
        if (matches[i + 1][0] == matches[i][0] + 1) and (
            matches[i + 1][1] == matches[i][1] + 1
        ):
            i += 1
            continue
        i += 1
        chunks += 1
    return chunks


def single_meteor_score(
    reference,
    hypothesis,
    preprocess=str.lower,
    stemmer=PorterStemmer(),
    wordnet=wordnet,
    alpha=0.9,
    beta=3,
    gamma=0.5,
):
    """
    Calculates METEOR score for single hypothesis and reference as per
    "Meteor: An Automatic Metric for MT Evaluation with HighLevels of
    Correlation with Human Judgments" by Alon Lavie and Abhaya Agarwal,
    in Proceedings of ACL.
    http://www.cs.cmu.edu/~alavie/METEOR/pdf/Lavie-Agarwal-2007-METEOR.pdf


    >>> hypothesis1 = 'It is a guide to action which ensures that the military always obeys the commands of the party'

    >>> reference1 = 'It is a guide to action that ensures that the military will forever heed Party commands'


    >>> round(single_meteor_score(reference1, hypothesis1),4)
    0.7398

        If there is no words match during the alignment the method returns the
        score as 0. We can safely  return a zero instead of raising a
        division by zero error as no match usually implies a bad translation.

    >>> round(meteor_score('this is a cat', 'non matching hypothesis'),4)
    0.0

    :param references: reference sentences
    :type references: list(str)
    :param hypothesis: a hypothesis sentence
    :type hypothesis: str
    :param preprocess: preprocessing function (default str.lower)
    :type preprocess: method
    :param stemmer: nltk.stem.api.StemmerI object (default PorterStemmer())
    :type stemmer: nltk.stem.api.StemmerI or any class that implements a stem method
    :param wordnet: a wordnet corpus reader object (default nltk.corpus.wordnet)
    :type wordnet: WordNetCorpusReader
    :param alpha: parameter for controlling relative weights of precision and recall.
    :type alpha: float
    :param beta: parameter for controlling shape of penalty as a
                 function of as a function of fragmentation.
    :type beta: float
    :param gamma: relative weight assigned to fragmentation penality.
    :type gamma: float
    :return: The sentence-level METEOR score.
    :rtype: float
    """
    enum_hypothesis, enum_reference = _generate_enums(
        hypothesis, reference, preprocess=preprocess
    )
    translation_length = len(enum_hypothesis)
    reference_length = len(enum_reference)
    matches, _, _ = _enum_allign_words(enum_hypothesis, enum_reference, stemmer=stemmer)
    matches_count = len(matches)
    try:
        precision = float(matches_count) / translation_length
        recall = float(matches_count) / reference_length
        fmean = (precision * recall) / (alpha * precision + (1 - alpha) * recall)
        chunk_count = float(_count_chunks(matches))
        frag_frac = chunk_count / matches_count
    except ZeroDivisionError:
        return 0.0
    penalty = gamma * frag_frac ** beta
    return (1 - penalty) * fmean



def meteor_score(
    references,
    hypothesis,
    preprocess=str.lower,
    stemmer=PorterStemmer(),
    wordnet=wordnet,
    # Original parameter values:
    #alpha=0.9,
    #beta=3,
    #gamma=0.5,
    # Parameter values we chose based on the paper: https://www.cs.cmu.edu/~alavie/METEOR/pdf/Lavie-Agarwal-2007-METEOR.pdf
    alpha=0.81,
    beta=0.83,
    gamma=0.28,
):
    """
    Calculates METEOR score for hypothesis with multiple references as
    described in "Meteor: An Automatic Metric for MT Evaluation with
    HighLevels of Correlation with Human Judgments" by Alon Lavie and
    Abhaya Agarwal, in Proceedings of ACL.
    http://www.cs.cmu.edu/~alavie/METEOR/pdf/Lavie-Agarwal-2007-METEOR.pdf


    In case of multiple references the best score is chosen. This method
    iterates over single_meteor_score and picks the best pair among all
    the references for a given hypothesis

    >>> hypothesis1 = 'It is a guide to action which ensures that the military␣
↪always obeys the commands of the party'
    >>> hypothesis2 = 'It is to insure the troops forever hearing the activity␣
↪guidebook that party direct'

    >>> reference1 = 'It is a guide to action that ensures that the military␣
↪will forever heed Party commands'
    >>> reference2 = 'It is the guiding principle which guarantees the military␣
↪forces always being under the command of the Party'
    >>> reference3 = 'It is the practical guide for the army always to heed the␣
↪directions of the party'

    >>> round(meteor_score([reference1, reference2, reference3], hypothesis1),4)
    0.7398

        If there is no words match during the alignment the method returns the
        score as 0. We can safely  return a zero instead of raising a
        division by zero error as no match usually implies a bad translation.

    >>> round(meteor_score(['this is a cat'], 'non matching hypothesis'),4)
    0.0

    :param references: reference sentences
    :type references: list(str)
    :param hypothesis: a hypothesis sentence
    :type hypothesis: str
    :param preprocess: preprocessing function (default str.lower)
    :type preprocess: method
    :param stemmer: nltk.stem.api.StemmerI object (default PorterStemmer())
    :type stemmer: nltk.stem.api.StemmerI or any class that implements a stem␣
↪method
    :param wordnet: a wordnet corpus reader object (default nltk.corpus.wordnet)
    :type wordnet: WordNetCorpusReader
    :param alpha: parameter for controlling relative weights of precision and␣
↪recall.
    :type alpha: float
    :param beta: parameter for controlling shape of penalty as a function
                 of as a function of fragmentation.
    :type beta: float
    :param gamma: relative weight assigned to fragmentation penality.
    :type gamma: float
    :return: The sentence-level METEOR score.
    :rtype: float
    """
    return max(
        [
            single_meteor_score(
                reference,
                hypothesis,
                stemmer=stemmer,
                wordnet=wordnet,
                alpha=alpha,
                beta=beta,
                gamma=gamma,
            )
            for reference in references
        ]
    )
```

[21]:

```
# Functiones defined for calculating the meteor scores

# Define the function to calculate BLEU scores for more than one inputs
def meteor_lists_no_df(golden_standard, text1, text2):
    # n-gram lists for text1 and text2
    meteor_list = [[],[]]

    # Append meteor scores - call the meteor function on text1 and text2
    meteor_list[0].append(meteor_score([golden_standard], text1))
    meteor_list[1].append(meteor_score([golden_standard], text2))

    # This one returns ngram_list directly for the function that makes a row␣
 ↪per answer
    return(meteor_list)

def meteor_more(gold1, berts, distilberts):
    # Initiate results dataframe
    results = pd.DataFrame()

    # Loop over rows in the dataframe
    for i in range(0,len(gold1)):

        # Define the inputs to the bleu function from the dataset
        gold_1 = gold1[i]
        bert = berts[i]
        distil = distilberts[i]

        # Get the bleu scores per line in dataframe with the predefined function
        meteor_list1 = meteor_lists_no_df(gold_1, bert, distil)

        # Append to dataframe, index is model name + iteration
        dff = pd.DataFrame(meteor_list1, index =['0', '1'], columns =␣
↪["METEOR"])

        # Append results to the dataframe
        results = results.append(dff)

    return(results)
```

[22]:

```
meteor_more(df['Gold_1'], df['BERT'], df['DistilBERT'])
```

[22]:

```
      METEOR
0   0.842492
1   0.842492
0   0.000000
1   0.000000
0   0.000000
..       …
1   0.000000
0   0.720000
1   0.720000
0   0.328767
1   0.328767

[22130 rows x 1 columns]
```

1.3 2. Calculate GLEU scores

1.3.1 2.1 Prepare the dataframe

[58]:

```
import pandas as pd
# Read in data
data = pd.read_csv("../tweetQA_bothpresent.csv")
```

[59]:

```
# Select columns relevant
df = data[['Answer', "L_BERT_answer", "DistilBERT_answer"]]
df = df.rename(columns={'Answer': 'Answers', "L_BERT_answer": "BERT",␣
↪"DistilBERT_answer": "DistilBERT"})
```

[60]:

```
df
```

[60]:

```
                                           Answers  \
0                                   ['w nj', 'w nj']
1       ['#endangeredriver', '#endangereddriver']
2                           ['wiggins', 'wiggins']
3          ['the game is tied at 106', '106-106']
4                   ["kemba's", "kemba's floater"]
…                                                …
```

```
11060                                   ['guns']
11061                        ['president obama']
11062                   ['our best, whole foods']
11063                                ['january']
11064                                ['shed it']

                                             BERT  \
0                                            w nj
1                               # endangeredriver
2      monstars basketball @ m0nstarsbballwiggins
3                                       106 - 106
4                                           kemba
…                                           …
11060                                        guns
11061                             president obama
11062                    it happens to the best of us
11063                                     january
11064                             shed their skin

                                       DistilBERT
0                                            w nj
1                                        jdsutter
2      monstars basketball @ m0nstarsbballwiggins
3                               106 - 106 . 8 . 9
4                                           kemba
…                                           …
11060                                        guns
11061                             president obama
11062                    it happens to the best of us
11063                                     january
11064                             shed their skin

[11065 rows x 3 columns]
```

[61]:

```
df.describe()
```

[61]:

```
                 Answers   BERT        DistilBERT
count              11065  11065             11065
unique              9432   8875              8991
top     ['donald trump']  trump  donald j . trump
freq                  33     30                26
```

The answers have sometimes more than one correct option: make the answers into two columns, Gold_1 and Gold_2.

[62]:

```
import ast
```

```
# Function to split the rows
def split_column(row):
    # Cuts the answers row into two if possible
    initial_gold = ast.literal_eval(row['Answers'])
    # Gold_1 will always have an input
    row['Gold_1'] = initial_gold[0]
    # If the list has more than 1 element, Gold_2 gets the second input
    if len(initial_gold) > 1:
        row['Gold_2'] = initial_gold[1]

    return(row)

# Apply on the dataframe
df = df.apply(split_column, axis = 1)
```

1.3.2 2.2 Apply the GLEU function

[27]:

```
import math
gleus = gleu_more(df['Gold_1'], df['Gold_2'], df['BERT'], df['DistilBERT'])
```

[28]:

```
#Define the bert and distilbert results based on the indexes given to them in␣
 ↪the function above
bert_results = gleus.loc["0"]
distil_results = gleus.loc["1"]
```

[97]:

```
# inspect
bert_results.describe()
```

[97]:

```
                  0
count  11065.000000
mean       0.542009
std        0.436460
min        0.000000
25%        0.055556
50%        0.500000
75%        1.000000
max        1.000000
```

[98]:

```
distil_results.describe()
```

[98]:

```
                  0
count  11065.000000
mean       0.472871
std        0.439373
min        0.000000
25%        0.000000
50%        0.333333
```

```
75%          1.000000
max          1.000000
```

```
[345]: bert_results.to_csv("bert_gleus.csv", index = False)
       distil_results.to_csv("distil_gleus.csv", index = False)
```

1.4 3. Calculate METEOR scores

```
[63]: meteors = meteor_more(df['Gold_1'], df['BERT'], df['DistilBERT'])
```

```
[67]: # Get the scores for bert and distilbert
      bert_meteors = meteors.loc["0"]
      distil_meteors = meteors.loc["1"]
```

```
[81]: bert_meteors.describe()
```

```
[81]:              METEOR
      count  11065.000000
      mean       0.460381
      std        0.337903
      min        0.000000
      25%        0.000000
      50%        0.547954
      75%        0.720000
      max        1.000000
```

The METEOR scores are sometimes above 1. Inspection below in part 4 shows that the score being one usually means the model was correct. Thus, the scores above 1 will be replaced with a score of 1.

```
[89]: bert_meteors['METEOR'] = np.where(bert_meteors['METEOR'] > 1, 1,␣
       ↪bert_meteors['METEOR'])
      bert_meteors.describe()
```

```
[89]:              METEOR
      count  11065.000000
      mean       0.460381
      std        0.337903
      min        0.000000
      25%        0.000000
      50%        0.547954
      75%        0.720000
      max        1.000000
```

```
[90]: distil_meteors['METEOR'] = np.where(distil_meteors['METEOR'] > 1, 1,␣
       ↪distil_meteors['METEOR'])
      distil_meteors.describe()
```

```
[90]:              METEOR
      count  11065.000000
      mean       0.415755
      std        0.343811
      min        0.000000
      25%        0.000000
      50%        0.397790
      75%        0.720000
      max        1.000000
```

```
[83]: bert_meteors.to_csv("bert_meteors.csv", index = False)
      distil_meteors.to_csv("distil_meteors.csv", index = False)
```

1.5 4. Appending to dataframe, getting 50 samples

```
[35]: df
```

```
[35]:                                          Answers  \
      0                               ['w nj', 'w nj']
      1      ['#endangeredriver', '#endangereddriver']
      2                         ['wiggins', 'wiggins']
      3         ['the game is tied at 106', '106-106']
      4                 ["kemba's", "kemba's floater"]
      …                                            …
      11060                                   ['guns']
      11061                        ['president obama']
      11062                  ['our best, whole foods']
      11063                                ['january']
      11064                                ['shed it']

                                                   BERT  \
      0                                            w nj
      1                               # endangeredriver
      2      monstars basketball @ m0nstarsbballwiggins
      3                                       106 - 106
      4                                           kemba
      …                                            …
      11060                                        guns
      11061                             president obama
      11062                it happens to the best of us
      11063                                     january
      11064                             shed their skin

                                             DistilBERT                   Gold_1  \
      0                                            w nj                     w nj
      1                                        jdsutter         #endangeredriver
      2      monstars basketball @ m0nstarsbballwiggins                  wiggins
```

```
3                                 106 - 106 . 8 . 9  the game is tied at 106
4                                             kemba                  kemba's
…                                            …                      …
11060                                          guns                     guns
11061                               president obama          president obama
11062                  it happens to the best of us    our best, whole foods
11063                                       january                  january
11064                               shed their skin                  shed it

                  Gold_2
0                   w nj
1      #endangereddriver
2                wiggins
3                106-106
4        kemba's floater
…                   …
11060                NaN
11061                NaN
11062                NaN
11063                NaN
11064                NaN

[11065 rows x 5 columns]
```

[84]:
```
# Reset indexes for results
bert_meteors = bert_meteors.reset_index(drop=True)
distil_meteors = distil_meteors.reset_index(drop=True)
bert_results = bert_results.reset_index(drop=True)
distil_results = distil_results.reset_index(drop=True)

# Append to dataframe
df['BERT_METEOR'] = bert_meteors['METEOR']
df['DistilBERT_METEOR'] = distil_meteors['METEOR']
df['BERT_GLEU'] = bert_results[0]
df['DistilBERT_GLEU'] = distil_results[0]
```

[85]:
```
df
```

[85]:
```
                                          Answers  \
0                                ['w nj', 'w nj']
1       ['#endangeredriver', '#endangereddriver']
2                          ['wiggins', 'wiggins']
3          ['the game is tied at 106', '106-106']
4                  ["kemba's", "kemba's floater"]
…                                            …
11060                                    ['guns']
11061                         ['president obama']
```

```
11062                  ['our best, whole foods']
11063                                ['january']
11064                                ['shed it']

                                             BERT  \
0                                            w nj   
1                               # endangeredriver   
2      monstars basketball @ m0nstarsbballwiggins   
3                                       106 - 106   
4                                           kemba   
…                                           …       
11060                                        guns   
11061                             president obama   
11062                it happens to the best of us   
11063                                     january   
11064                             shed their skin   

                                       DistilBERT                   Gold_1  \
0                                            w nj                     w nj   
1                                        jdsutter         #endangeredriver   
2      monstars basketball @ m0nstarsbballwiggins                  wiggins   
3                               106 - 106 . 8 . 9  the game is tied at 106   
4                                           kemba                  kemba's   
…                                           …                   …          
11060                                        guns                     guns   
11061                             president obama          president obama   
11062                it happens to the best of us    our best, whole foods   
11063                                     january                  january   
11064                             shed their skin                  shed it   

                  Gold_2  BERT_METEOR  DistilBERT_METEOR  BERT_GLEU  \
0                   w nj     0.842492           0.842492   1.000000   
1      #endangereddriver     0.000000           0.000000   1.000000   
2                wiggins     0.000000           0.000000   0.000000   
3                106-106     0.132597           0.116317   0.055556   
4        kemba's floater     0.000000           0.000000   0.000000   
…                  …          …                 …        …            
11060                NaN     0.720000           0.720000   1.000000   
11061                NaN     0.842492           0.842492   1.000000   
11062                NaN     0.000000           0.000000   0.045455   
11063                NaN     0.720000           0.720000   1.000000   
11064                NaN     0.328767           0.328767   0.166667   

       DistilBERT_GLEU  
0             1.000000  
1             0.000000  
2             0.000000  
```

```
3             0.055556
4             0.000000
…                …
11060         1.000000
11061         1.000000
11062         0.045455
11063         1.000000
11064         0.166667

[11065 rows x 9 columns]
```

[102]:

```python
# Random sample 50 sentences
samples = df.sample(n=100)

# Read out samples to inspect them in Excel
samples.to_csv("df_samples_scores100.csv")
```

[87]:

```python
# Make the two dataframes for short and long answers
import numpy as np

short_answers = df.loc[np.array(list(map(len,df["Gold_1"].str.split()))) < 3]
long_answers = df.loc[np.array(list(map(len,df["Gold_1"].str.split()))) >= 3]
```

[88]:

```python
short_answers.to_csv("df_short_answers.csv")
long_answers.to_csv("df_long_answers.csv")
```

[52]:

```python
# Inspecting the dataframe where meteor scores were bigger than 1
df[df["BERT_METEOR"] => 1]
```

[52]:

```
                                   Answers                              BERT  \
449                    ['shark', 'sharks']                            sharks
1027                     ['steal from us']                  stealing from us
1099                       ['paul walker']                      paul walkers
1418                     ['cairnes store']                     cairns stores
1581   ['hundred of thousands of dollars']  hundreds of thousands of dollars
2365                             ['smile']                            smiled
2422              ['transgendered people']                transgender people
3731                  ['play racist call']                plays racist calls
4320                        ['stereotype']                       stereotypes
4769                  ['work really hard']                 works really hard
5068                ['female action star']               female action stars
5332                    ['stop to listen']                stopping to listen
5765                               ['end']                             ended
7066                               ['gun']                              guns
7373                 ['42 degree celsius']                42 degrees celsius
7936               ['help deliver babies']             helps delivers babies
8102               ['black makeup artist']              black makeup artists
```

```
8362                     ['dodge bullets']                    dodged bullets
8611                       ['paul walker']                      paul walkers
9685                        ['killing me']                 trying to kill me
10666                       ['masturbate']                 masturbated at me
10922                          ['pull it']                        pulling it

                                           DistilBERT  \
449    jason demers ( @ jasondemers5 ) july 16 , 2014
1027                                         stealing
1099                                     paul walkers
1418                                           cairns
1581                 hundreds of thousands of dollars
2365                                           smiled
2422                               transgender people
3731                               plays racist calls
4320                                      stereotypes
4769                                work really hard
5068                              female action stars
5332                               stopping to listen
5765                                            ended
7066                              kim kardashian west
7373                               42 degrees celsius
7936                            helps delivers babies
8102                             black makeup artists
8362                                   dodged bullets
8611                                       kevin hart
9685                                trying to kill me
10666                                     masturbated
10922                                         pulling

                                Gold_1  Gold_2  BERT_METEOR  \
449                              shark  sharks     1.440000
1027                     steal from us     NaN     1.123322
1099                       paul walker     NaN     1.200019
1418                     cairnes store     NaN     1.200019
1581   hundred of thousands of dollars     NaN     1.065002
2365                             smile     NaN     1.440000
2422              transgendered people     NaN     1.200019
3731                  play racist call     NaN     1.361264
4320                        stereotype     NaN     1.440000
4769                  work really hard     NaN     1.123322
5068                 female action star     NaN     1.123322
5332                    stop to listen     NaN     1.123322
5765                               end     NaN     1.440000
7066                               gun     NaN     1.440000
7373                 42 degree celsius     NaN     1.123322
7936               help deliver babies     NaN     1.361264
```

```
8102                 black makeup artist     NaN     1.123322
8362                       dodge bullets     NaN     1.200019
8611                         paul walker     NaN     1.200019
9685                          killing me     NaN     1.008419
10666                         masturbate     NaN     1.043478
10922                            pull it     NaN     1.200019

       DistilBERT_METEOR  BERT_GLEU  DistilBERT_GLEU
449             0.000000   1.000000         0.000000
1027            0.549618   0.500000         0.000000
1099            1.200019   0.333333         0.333333
1418            0.397790   0.000000         0.000000
1581            1.065002   0.714286         0.714286
2365            1.440000   0.000000         0.000000
2422            1.200019   0.333333         0.333333
3731            1.361264   0.166667         0.166667
4320            1.440000   0.000000         0.000000
4769            1.123322   0.500000         0.500000
5068            1.123322   0.500000         0.500000
5332            1.123322   0.500000         0.500000
5765            1.440000   0.000000         0.000000
7066            0.000000   0.000000         0.000000
7373            1.123322   0.333333         0.333333
7936            1.361264   0.166667         0.166667
8102            1.123322   0.500000         0.500000
8362            1.200019   0.333333         0.333333
8611            0.000000   0.333333         0.000000
9685            1.008419   0.100000         0.100000
10666           1.440000   0.000000         0.000000
10922           0.795580   0.333333         0.000000
```

[99]:
```
short_answers.describe()
```

[99]:
```
       BERT_METEOR  DistilBERT_METEOR    BERT_GLEU  DistilBERT_GLEU
count  7245.000000        7245.000000  7245.000000      7245.000000
mean      0.474589           0.430672     0.614320         0.534981
std       0.343093           0.351143     0.447432         0.459766
min       0.000000           0.000000     0.000000         0.000000
25%       0.000000           0.000000     0.043478         0.000000
50%       0.657534           0.521739     1.000000         0.333333
75%       0.720000           0.720000     1.000000         1.000000
max       1.000000           1.000000     1.000000         1.000000
```

[100]:
```
long_answers.describe()
```

[100]:
```
       BERT_METEOR  DistilBERT_METEOR    BERT_GLEU  DistilBERT_GLEU
count  3820.000000        3820.000000  3820.000000      3820.000000
```

```
mean      0.433435           0.387465     0.404865         0.355074
std       0.326184           0.327642     0.378705         0.370297
min       0.000000           0.000000     0.000000         0.000000
25%       0.146568           0.000000     0.071429         0.000000
50%       0.426683           0.350308     0.300000         0.200000
75%       0.724701           0.665626     0.714286         0.600000
max       1.000000           1.000000     1.000000         1.000000
```

1.6 Additional code: how GLEU compares to METEOR in short examples

[39]:

```python
# Meteor is lower for an exact short match
print(meteor_score(["one"], "one"))
print(nltk.translate.gleu_score.sentence_gleu(["one"], "one"))
```

```
0.72
1.0
```

[40]:

```python
# Meteor is higher for when there are more words in exact match
a = "one and seventeen"
print(meteor_score([a], a))
print(nltk.translate.gleu_score.sentence_gleu([a.split()], a.split()))
```

```
0.8875013295249914
1.0
```