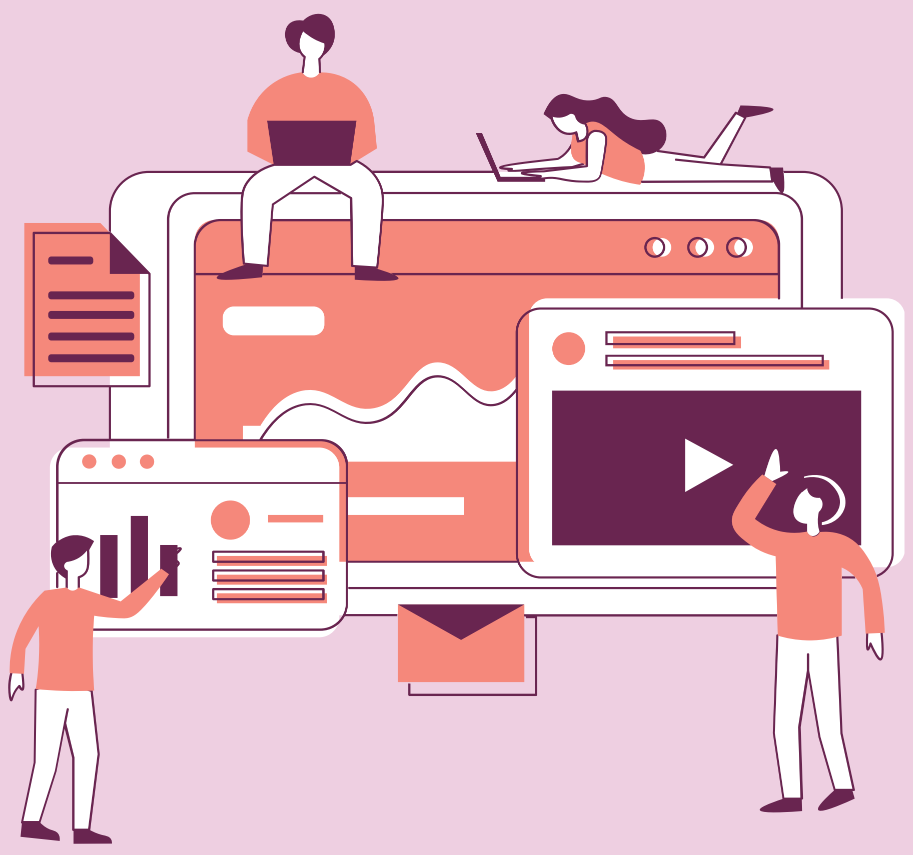
LLM Evaluation Metrics in Detail - When & How To Use Them



This document explores various evaluation metrics and recent methods essential for assessing the performance of Large Language Models (LLMs) across different Natural Language Processing (NLP) tasks.

Text similarity

Text Similarity Metrics:

- BLEU
- ROUGE, ROUGE-N, ROUGE-L
- METEOR
- FUZZY
- LEVENSHTEIN SIMILARITY RATIO

Semantic similarity

Semantic Similarity Metrics:

- BERTScore
- MoverScore
- Cosine Similarity

Text summarization

Text summarization Metrics:

- SUPERT
- BLANC
- FactCC

Others

Other Metrics:

- Perplexity
- Rule-based Metrics
- Functional Correctness
- Big Bench, GLUE Benchmark, MMLU, SQUAD etc

-Amrita Rath

TEXT SIMILARITY METRICS

1. BLEU SCORE (Reference : https://huggingface.co/spaces/evaluate-metric/bleu)

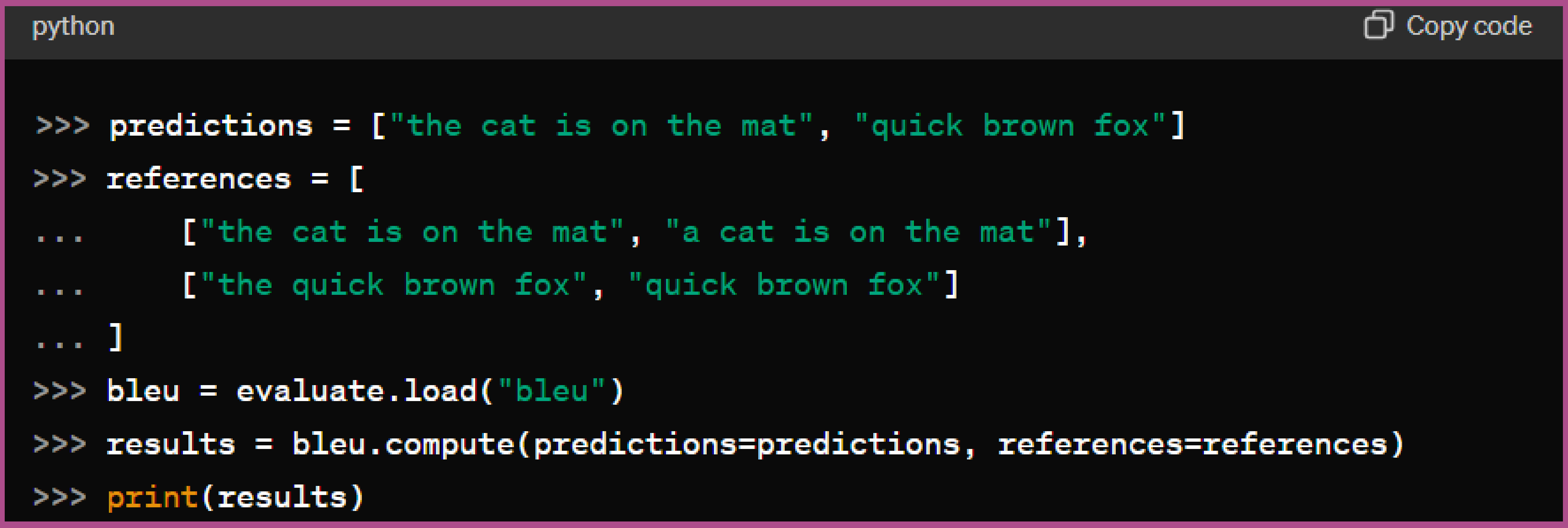
- BLEU (Bilingual Evaluation Understudy) is a widely-used algorithm for evaluating the quality of machine-translated text by comparing it to professional human translations.

- BLEU scores individual translated segments against reference translations and averages them to estimate the overall quality, focusing on correspondence rather than intelligibility or grammatical correctness.

- Despite its limitations, BLEU remains popular due to its high correlation with human judgments of translation quality and its cost-effectiveness.

HOW TO USE

This metric takes as input a list of predicted sentences and a list of lists of reference sentences (since each predicted sentence can have multiple references):



```python
>>> predictions = ["the cat is on the mat", "quick brown fox"]
>>> references = [
...     ["the cat is on the mat", "a cat is on the mat"],
...     ["the quick brown fox", "quick brown fox"]
... ]
>>> bleu = evaluate.load("bleu")
>>> results = bleu.compute(predictions=predictions, references=references)
>>> print(results)
```

Results :

{'bleu': 0.7598356856515925, 'precisions': [0.9166666666666666, 0.75, 0.6666666666666666, 0.5], 'brevity_penalty': 1.0, 'length_ratio': 1.0, 'translation_length': 8, 'reference_length': 8}

BLEU Input :

- predictions (list of strs): Translations to score.
- references (list of lists of strs): references for each translation.

BLEU Output :

BLEU's output is always a number between 0 and 1. This value indicates how similar the candidate text is to the reference texts, with values closer to 1 representing more similar texts.

BLEU Output Values : **bleu (float)**: bleu score, **precisions (list of floats)**: geometric mean of n-gram precisions, **brevity_penalty (float)**: brevity penalty, **length_ratio (float)**: ratio of lengths, **translation_length (int)**: translation_length, **reference_length (int)**: reference_length

*The brevity penalty penalizes generated translations that are too short compared to the closest reference length with an exponential decay.

2. ROUGE SCORE (Reference : https://huggingface.co/spaces/evaluate-metric/rouge)

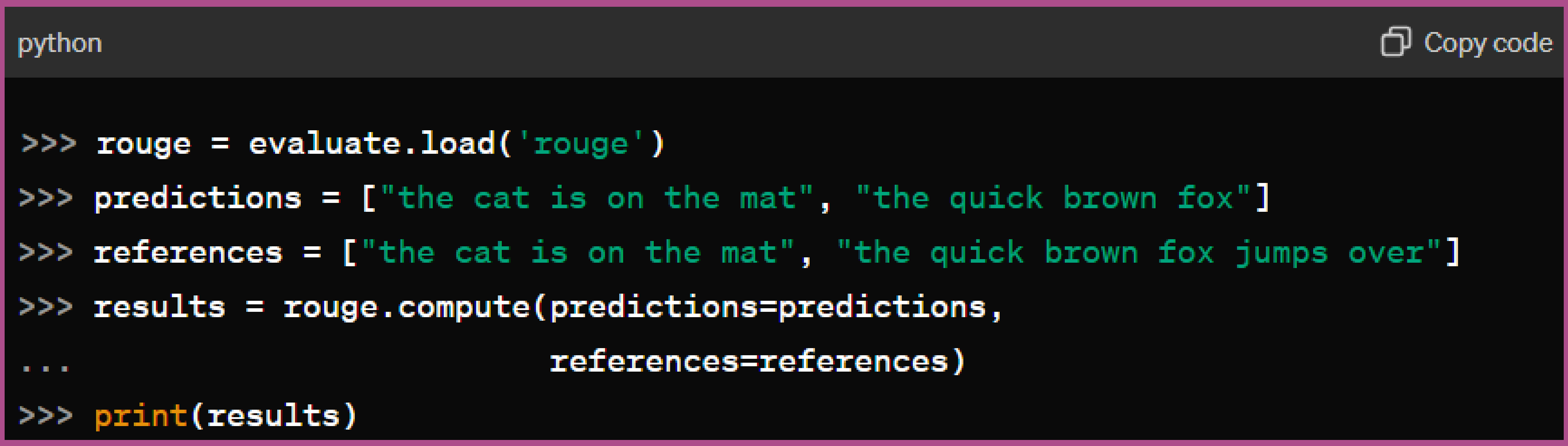
- ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics and software used to evaluate automatic summarization and machine translation by comparing machine outputs to human-produced references.

- ROUGE metrics assess the quality of summaries or translations by measuring the overlap of n-grams, word sequences, and word pairs between the machine-generated and reference texts.

- ROUGE is case insensitive, treating upper and lower case letters equivalently in the evaluation process.

HOW TO USE

At minimum, this metric takes as input a list of predictions and a list of references.



```python
>>> rouge = evaluate.load('rouge')
>>> predictions = ["the cat is on the mat", "the quick brown fox"]
>>> references = ["the cat is on the mat", "the quick brown fox jumps over"]
>>> results = rouge.compute(predictions=predictions,
...                         references=references)
>>> print(results)
```

Results :

{'rouge1': 0.9333333333333333, 'rouge2': 0.8, 'rougeL': 0.9333333333333333, 'rougeLsum': 0.9333333333333333}

ROUGE Input :

- predictions (list): list of predictions to score. Each prediction should be a string with tokens separated by spaces.
- references (list or list[list]): list of reference for each prediction or a list of several references per prediction. Each reference should be a string with tokens separated by spaces.
- rouge_types (list): A list of rouge types to calculate. Defaults to ['rouge1', 'rouge2', 'rougeL', 'rougeLsum']. Valid rouge types: "rouge1": unigram (1-gram) based scoring, "rouge2": bigram (2-gram) based scoring, "rougeL": Longest common subsequence based scoring, "rougeLSum": splits text using "\n"
- use_aggregator (boolean): If True, returns aggregates. Defaults to True.
- use_stemmer (boolean): If True, uses Porter stemmer to strip word suffixes. Defaults to False.

ROUGE Output :

- The output is a dictionary with one entry for each rouge type in the input list rouge_types.
- If use_aggregator=False, each dictionary entry is a list of scores, with one score for each sentence. E.g. if rouge_types=['rouge1', 'rouge2'] and use_aggregator=False, the output is:

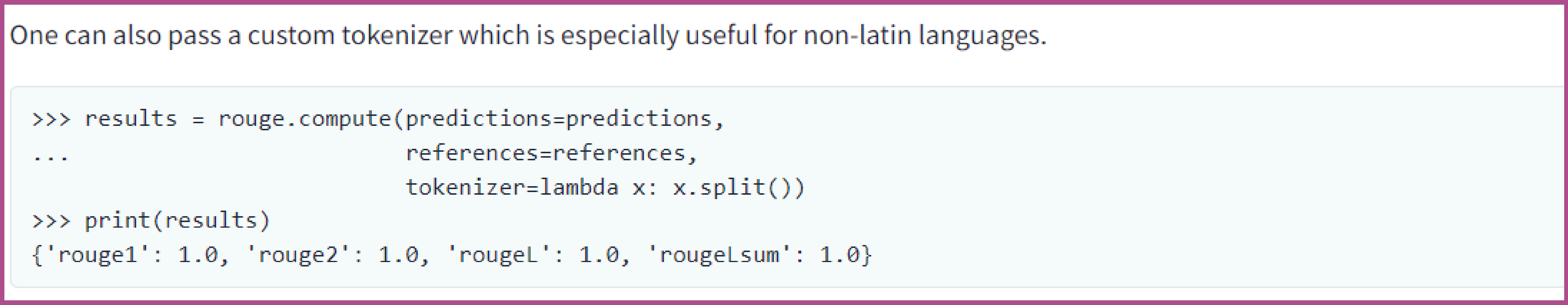
{'rouge1': [0.6666666666666666, 1.0], 'rouge2': [0.0, 1.0]}

- If rouge_types=['rouge1', 'rouge2'] and use_aggregator=True, the output is of the following format:

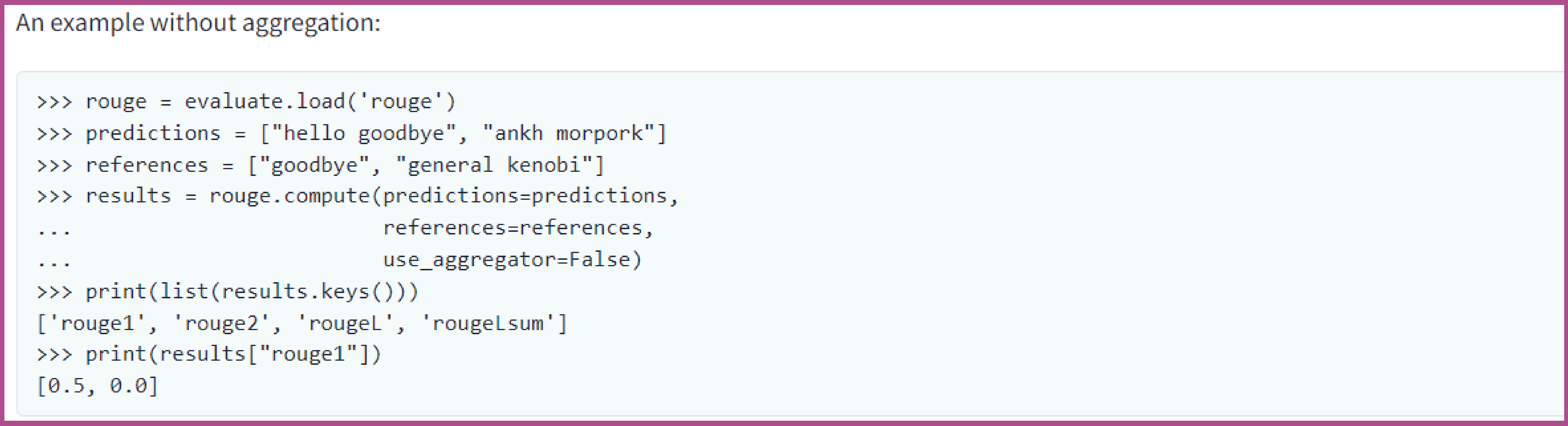
{'rouge1': 1.0, 'rouge2': 1.0}

- The ROUGE values are in the range of 0 to 1.

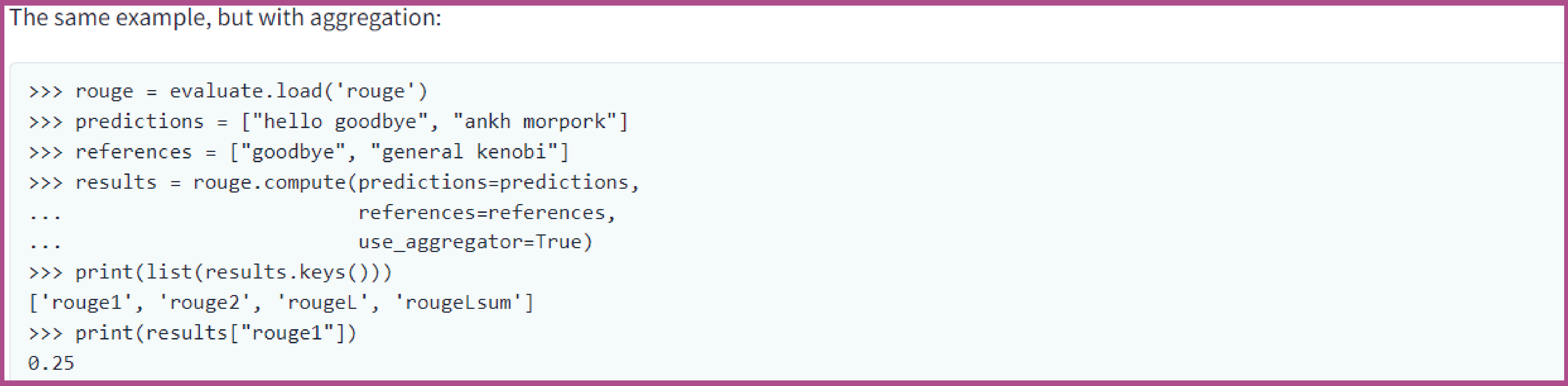
SOME EXAMPLES:


One can also pass a custom tokenizer which is especially useful for non-latin languages.

```
>>> results = rouge.compute(predictions=predictions,
...                         references=references,
                            tokenizer=lambda x: x.split())
>>> print(results)
{'rouge1': 1.0, 'rouge2': 1.0, 'rougeL': 1.0, 'rougeLsum': 1.0}
```


An example without aggregation:

```
>>> rouge = evaluate.load('rouge')
>>> predictions = ["hello goodbye", "ankh morpork"]
>>> references = ["goodbye", "general kenobi"]
>>> results = rouge.compute(predictions=predictions,
...                         references=references,
...                         use_aggregator=False)
>>> print(list(results.keys()))
['rouge1', 'rouge2', 'rougeL', 'rougeLsum']
>>> print(results["rouge1"])
[0.5, 0.0]
```


The same example, but with aggregation:

```
>>> rouge = evaluate.load('rouge')
>>> predictions = ["hello goodbye", "ankh morpork"]
>>> references = ["goodbye", "general kenobi"]
>>> results = rouge.compute(predictions=predictions,
...                         references=references,
...                         use_aggregator=True)
>>> print(list(results.keys()))
['rouge1', 'rouge2', 'rougeL', 'rougeLsum']
>>> print(results["rouge1"])
0.25
```

3. ROUGE-N/L SCORE (Reference : https://github.com/google-research/google-research/tree/master/rouge)

This package replicates the results of the original ROUGE perl script with several features:

- ROUGE-N (N-gram) scoring: Measures the overlap of n-grams between the predicted and reference texts.
- ROUGE-L (Longest Common Subsequence) scoring: Assesses the longest matching sequence of words between the texts.
- Text normalization: Standardizes text format before comparison.
- Bootstrap resampling: Provides confidence intervals for scores.
- Porter stemming: Optionally removes word suffixes (like "ing", "ion") to focus on word roots.

2 flavors of ROUGE -L:

- Sentence-level ROUGE-L: Computes the Longest Common Subsequence (LCS) between two pieces of text while ignoring newlines. This method evaluates the overlap of the longest sequence of words that appear in both the predicted and reference texts in the same order.

- Summary-level ROUGE-L: Treats newlines in the text as sentence boundaries and computes the LCS between each pair of reference and candidate sentences. It then calculates the union of these LCS results to provide an overall score for the summary. This method better handles multi-sentence texts by considering sentence boundaries.

Stopword removal is not included due to inconsistencies and licensing issues with the original stopword list. Not all perl script options are supported, but the implemented options aim to replicate the original functionality.

ROUGE-N Example

Predicted text: "the cat is on the mat"
Reference text: "the cat sat on the mat"
For bigrams (n=2):

- Predicted bigrams: "the cat", "cat is", "is on", "on the", "the mat"
- Reference bigrams: "the cat", "cat sat", "sat on", "on the", "the mat"

Overlap: "the cat", "on the", "the mat"
ROUGE-2 score: Number of overlapping bigrams / Total bigrams in reference

- Overlapping bigrams: 3
- Total bigrams in reference: 5
- ROUGE-2 score: 3/5 = 0.6

ROUGE-L Example

Predicted text: "the cat is on the mat"
Reference text: "the cat sat on the mat"
LCS: "the cat on the mat"
The length of the LCS is 5 words.
ROUGE-L score: Length of LCS / Total words in reference

- Length of LCS: 5
- Total words in reference: 6
- ROUGE-L score: $5/6 \approx 0.83$

4. METEOR SCORE (Reference : https://huggingface.co/spaces/evaluate-metric/meteor, https://aclanthology.org/W05-0909.pdf)

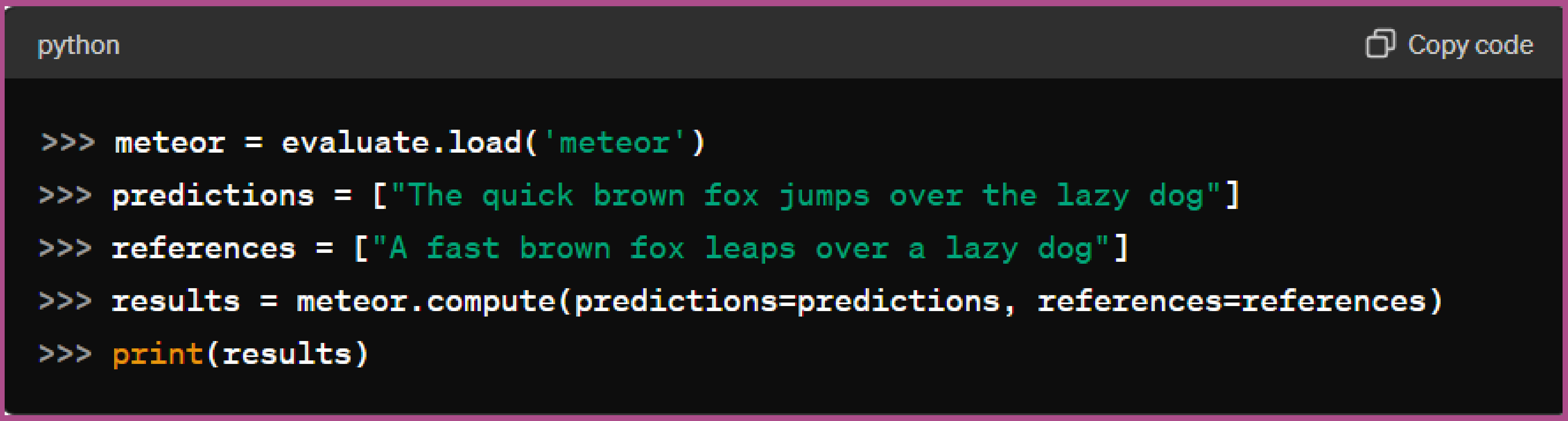
- METEOR: An automatic metric for evaluating machine translations by matching unigrams between machine-produced and human-produced translations.

- Unigram Matching: Matches unigrams based on surface forms, stemmed forms, and meanings, with potential for more advanced matching strategies.

- Scoring: Combines unigram-precision, unigram-recall, and a fragmentation measure to assess the order of matched words. Calculated based on the harmonic mean of precision and recall, with recall weighted more than precision.

- Correlation with Human Evaluation: Achieves R values of 0.347 for Arabic and 0.331 for Chinese, improving upon simple unigram-precision, unigram-recall, and their F1 combination.

HOW TO USE



```python
>>> meteor = evaluate.load('meteor')
>>> predictions = ["The quick brown fox jumps over the lazy dog"]
>>> references = ["A fast brown fox leaps over a lazy dog"]
>>> results = meteor.compute(predictions=predictions, references=references)
>>> print(results)
```

METEOR input:
- predictions: a list of predictions to score. Each prediction should be a string with tokens separated by spaces.
- references: a list of references (in the case of one reference per prediction), or a list of lists of references (in the case of multiple references per prediction. Each reference should be a string with tokens separated by spaces.
- It also has several optional parameters:
- alpha: Parameter for controlling relative weights of precision and recall. The default value is 0.9.
- beta: Parameter for controlling shape of penalty as a function of fragmentation. The default value is 3.
- gamma: The relative weight assigned to fragmentation penalty. The default is 0.5

Results :
- {'meteor': 0.9999142661179699}. METEOR Output: The metric outputs a dictionary containing the METEOR score. Its values range from 0 to 1

Fragmentation Penalty:
*The penalty is based on the number of chunks of contiguous matched words in the hypothesis and the reference. More chunks mean more fragmentation, which leads to a higher penalty.

5. Levenshtein distance in Python (Reference : https://learn.microsoft.com/en-us/ai/playbook/technology-guidance/generative-ai/working-with-llms/eval-metrics)

- Levenshtein Similarity Ratio: Measures the similarity between two sequences based on the Levenshtein Distance, which is the minimum number of single-character edits needed to transform one sequence into the other.

- A few different methods are derived from Simple Levenshtein Similarity Ratio:

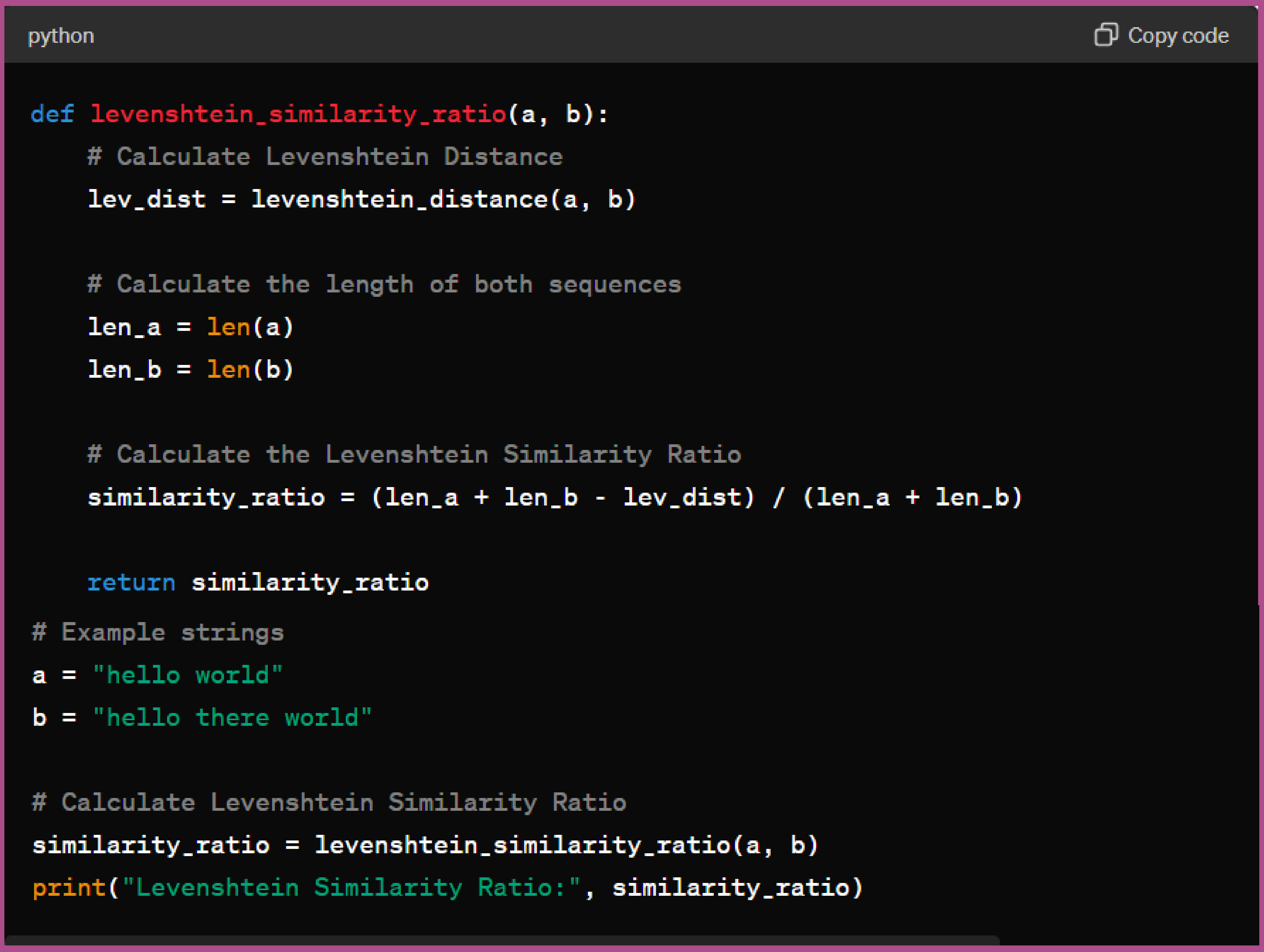
1. Simple Ratio: Computes similarity as the ratio of the sum of the lengths of the sequences minus the Levenshtein Distance, divided by the sum of the lengths of the sequences.

2. Partial Ratio: Determines similarity by comparing the shortest sequence against sub-strings of the same length in the longer sequence.

3.Token-sort Ratio: Splits sequences into tokens, sorts them alphabetically, and then compares the resulting strings using the simple ratio method.

4.Token-set Ratio: Splits sequences into tokens and computes similarity based on the intersection and union of the token sets between the two sequences.

HOW TO USE



```python
def levenshtein_similarity_ratio(a, b):
    # Calculate Levenshtein Distance
    lev_dist = levenshtein_distance(a, b)

    # Calculate the length of both sequences
    len_a = len(a)
    len_b = len(b)

    # Calculate the Levenshtein Similarity Ratio
    similarity_ratio = (len_a + len_b - lev_dist) / (len_a + len_b)

    return similarity_ratio
# Example strings
a = "hello world"
b = "hello there world"

# Calculate Levenshtein Similarity Ratio
similarity_ratio = levenshtein_similarity_ratio(a, b)
print("Levenshtein Similarity Ratio:", similarity_ratio)
```

Results:
Levenshtein Similarity Ratio: 0.9230769230769231

SEMANTIC SIMILARITY METRICS

1. BERT SCORE (Reference : https://openreview.net/pdf?id=SkeHuCVFDr, https://huggingface.co/spaces/evaluate-metric/bertscore)

- Leverages BERT Embeddings: BERTScore utilizes pre-trained contextual embeddings from BERT models to compare words in candidate and reference sentences using cosine similarity, allowing for more nuanced evaluation of text generation.

- Correlates with Human Judgment: It has demonstrated strong correlations with human judgment on both sentence-level and system-level evaluation tasks, indicating its effectiveness in capturing the quality of generated text.

- Computes Precision, Recall, and F1: BERTScore goes beyond mere similarity computation by also providing precision, recall, and F1 measures, offering a comprehensive evaluation approach for various language generation tasks.

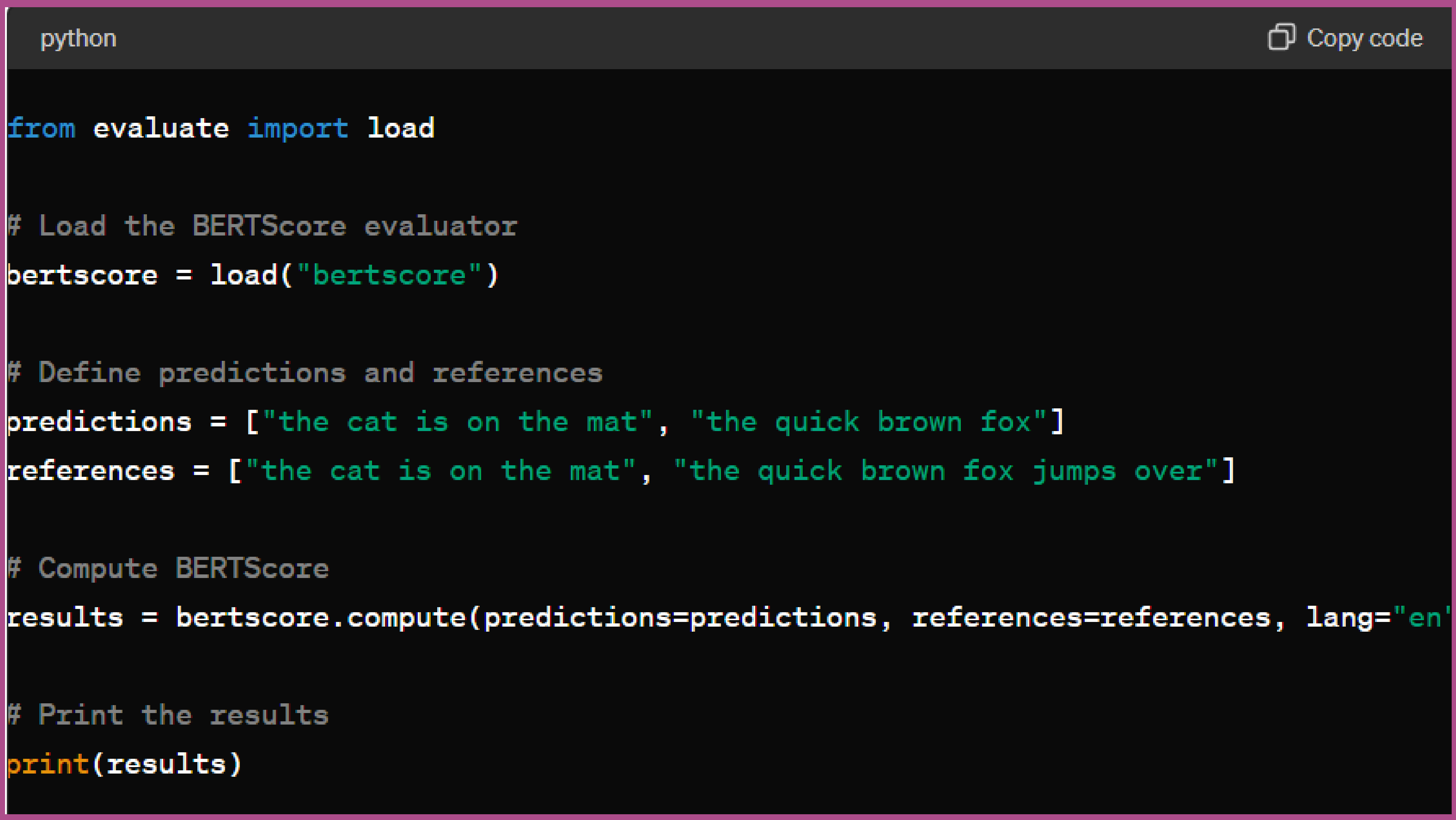
HOW TO USE

BERTScore takes 3 mandatory arguments :

- predictions (a list of string of candidate sentences)
- references (a list of strings or list of list of strings of reference sentences)
- either lang (a string of two letters indicating the language of the sentences, in ISO 639-1 format) or model_type (a string specififying which model to use, according to the BERT specification). The default behavior of the metric is to use the suggested model for the target language when one is specified, otherwise to use the model_type indicated.

OTHER OPTIONAL ARGUMENTS

- num_layers: Specifies the layer of representation to use, defaulting to the number of layers tuned on WMT16 correlation data based on the model_type.
- verbose: Enables intermediate status updates if set to True, defaulting to False.
- idf: Allows for IDF weighting, either as a boolean or a precomputed idf_dict.
- device: Specifies the device for allocation of the contextual embedding model. If None, the model is allocated on cuda:0 if available.
- nthreads: Sets the number of threads used for computation, defaulting to 4.
- rescale_with_baseline: Enables rescaling of BERTScore with a pre-computed baseline if set to True, defaulting to False.
- batch_size: Sets the processing batch size for BERTScore, required when rescale_with_baseline is True.
- baseline_path: Specifies a custom baseline file path.
- use_fast_tokenizer: Specifies whether to use the fast tokenizer, defaulting to False.



```python
from evaluate import load

# Load the BERTScore evaluator
bertscore = load("bertscore")

# Define predictions and references
predictions = ["the cat is on the mat", "the quick brown fox"]
references = ["the cat is on the mat", "the quick brown fox jumps over"]

# Compute BERTScore
results = bertscore.compute(predictions=predictions, references=references, lang="en"

# Print the results
print(results)
```

Output values-
BERTScore outputs a dictionary with the following values:

- precision: The precision for each sentence from the predictions + references lists, which ranges from 0.0 to 1.0.
- recall: The recall for each sentence from the predictions + references lists, which ranges from 0.0 to 1.0.
- f1: The F1 score for each sentence from the predictions + references lists, which ranges from 0.0 to 1.0.
- hashcode: The hashcode of the library.

Results:
{'precision': [0.7217950825691223, 0.8274509906768799], 'recall': [0.8125, 1.0], 'f1': [0.7643851633071899, 0.9056603908538818], 'hashcode': '82bd3582ddc424b1bf0b13cf9bf3f10785f7c40b1b2f6d1d500ffde9c65eb2e3'}

LIMITATIONS:

- Correlation with Human Judgment: BERTScore demonstrates strong correlation with human judgment on both sentence-level and system-level evaluation tasks, though the extent of this correlation may vary depending on the chosen model and language pair.

- Language Support: BERTScore does not support all languages, and it's important to refer to the supported language list provided by BERTScore for compatibility information.

- **Model Selection**: Choosing the appropriate BERT model is crucial, as it impacts both the accuracy of the score and the storage space required. Selecting a smaller model, such as distilbert-base-uncased, can mitigate storage and download time concerns while still providing effective evaluation.

2.MOVER SCORE (Reference : https://paperswithcode.com/paper/moverscore-text-generation-evaluating-with)

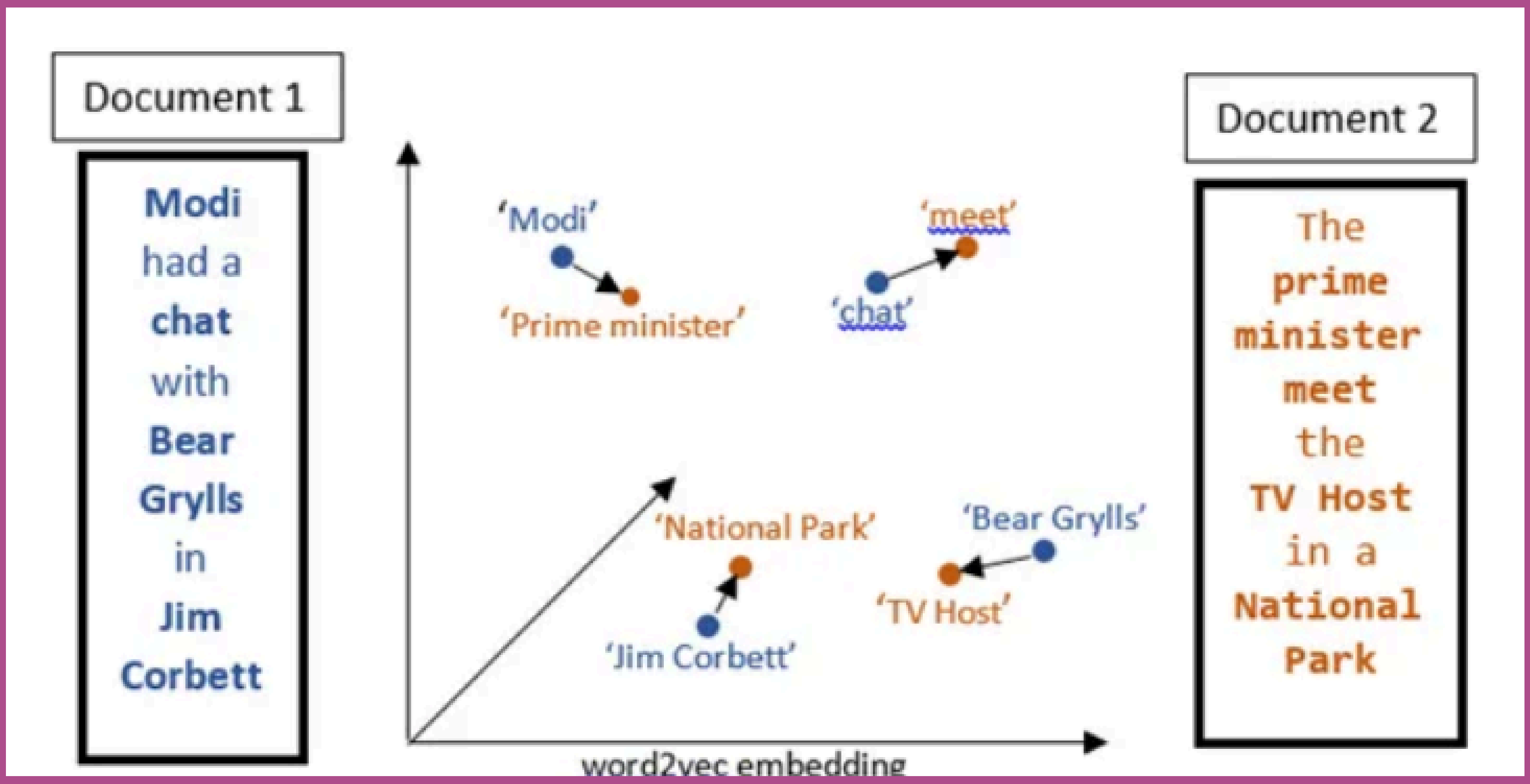
- Semantic Comparison: MoverScore, a novel evaluation metric, emphasizes comparing system outputs with references based on semantic similarity rather than surface forms, contributing to more robust evaluation of text generation systems.

- Validation Across Tasks: MoverScore is validated across various text generation tasks including summarization, machine translation, image captioning, and data-to-text generation, showcasing its versatility and effectiveness in capturing text quality.

- Contextualized Representations: Metrics that combine contextualized representations with a distance measure, such as MoverScore, demonstrate superior performance and strong generalization capability across different text generation tasks, enhancing their utility and reliability for evaluation purposes.

HOW TO USE

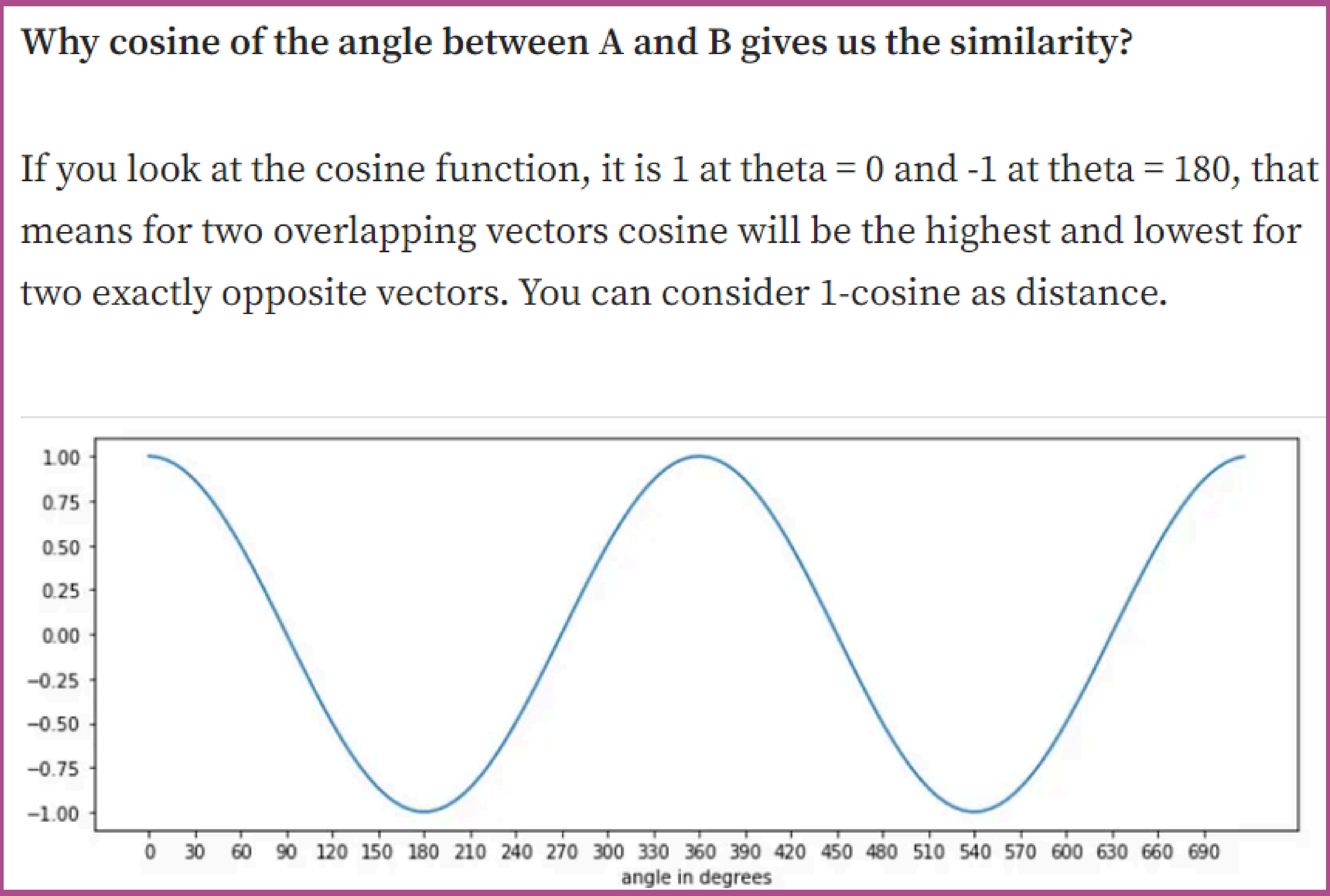
- Word Mover's Distance (WMD) leverages the semantic meaning encoded in word vector embeddings and treats text documents as weighted point clouds of embedded words.
- The distance between two text documents is computed by finding the minimum cumulative distance that words from one document need to travel to match exactly the point cloud of the other document.
- This approach allows for a semantic comparison between documents, considering the semantic meaning of individual words rather than just their surface forms.



Reference :https://medium.com/@nihitextra/word-movers-distance-for-text-similarity-7492aeca71b0

3. COSINE SIMILARITY (Reference : https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db

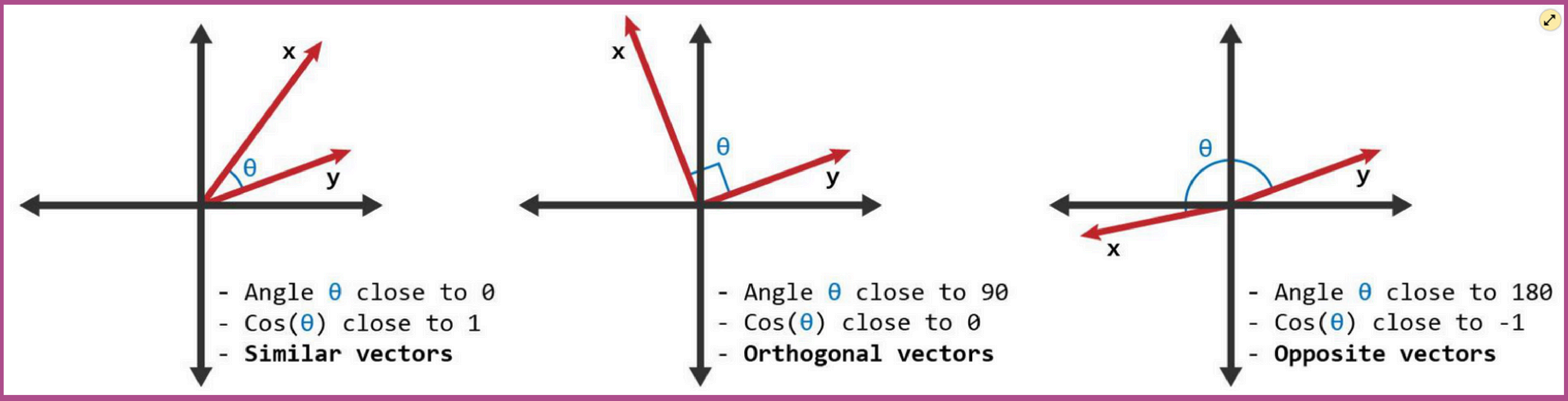
- Cosine similarity measures the similarity between two vectors by calculating the cosine of the angle between them.
- Widely used in Data Science, it finds applications across various domains including NLP for document similarity, information retrieval, bioinformatics for DNA sequence comparison, and plagiarism detection, owing to its effectiveness and versatility in capturing similarity between vectors.



Why cosine of the angle between A and B gives us the similarity?

If you look at the cosine function, it is 1 at theta = 0 and -1 at theta = 180, that means for two overlapping vectors cosine will be the highest and lowest for two exactly opposite vectors. You can consider 1-cosine as distance.

Reference : https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db



Reference : https://www.learndatasci.com/glossary/cosine-similarity/

- This metric measures the cosine of the angle between two non-zero vectors and ranges from -1 to 1. 1 means the two vectors are identical and -1 means they are dissimilar.

TEXT SUMMARIZATION METRICS

1. SUPERT (Reference : https://github.com/danieldeutsch/SUPERT, https://aclanthology.org/2020.acl-main.124/)

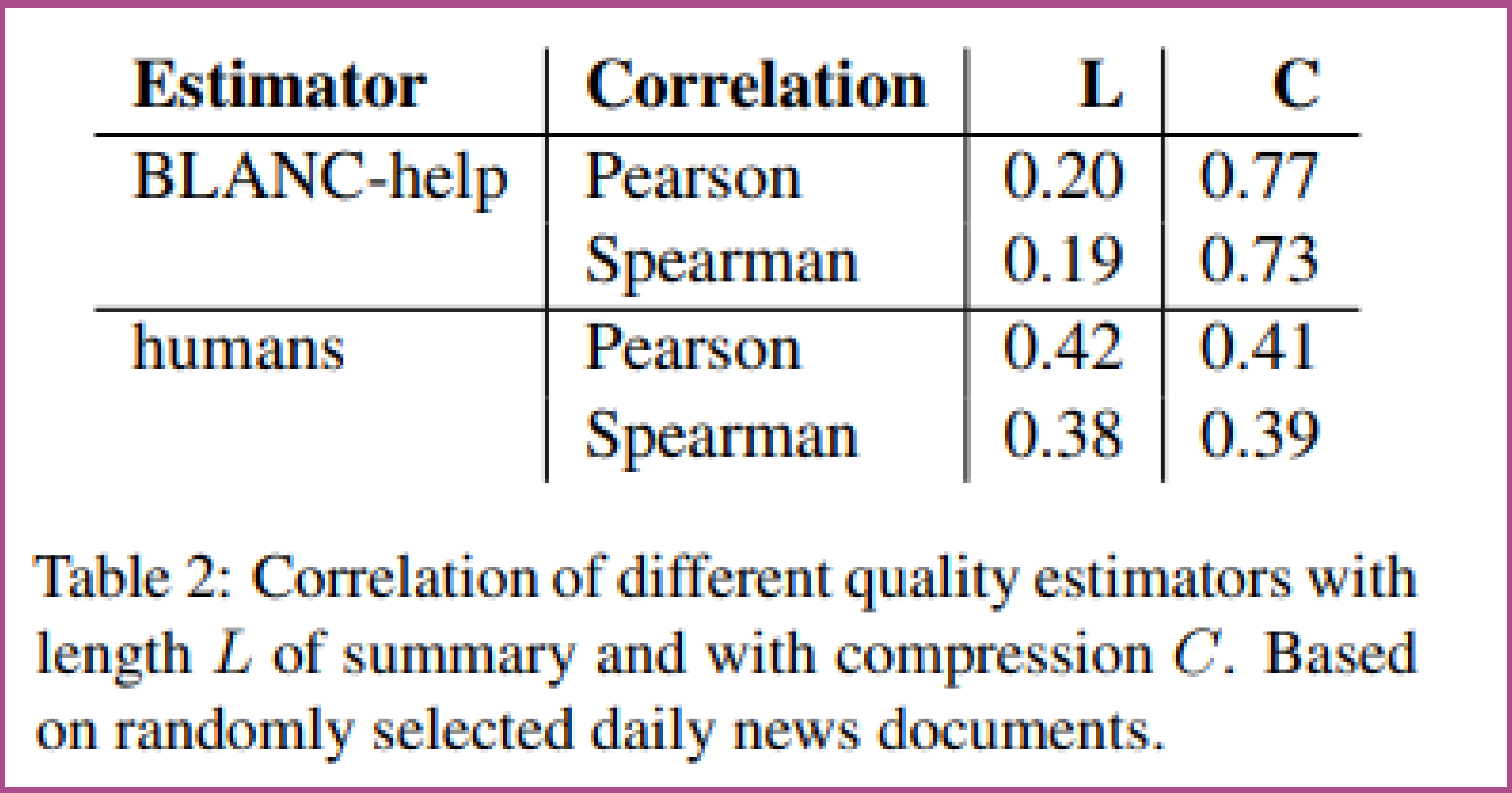
- **Unsupervised Evaluation**: SUPERT is an unsupervised multi-document summarization evaluation metric that does not require human-written reference summaries or annotations, making it efficient and scalable for evaluating summaries.

- **Semantic Similarity with Pseudo Reference**: SUPERT rates the quality of a summary by measuring its semantic similarity with a pseudo reference summary, which consists of salient sentences selected from the source documents. This approach utilizes contextualized embeddings and soft token alignment techniques to capture semantic similarity.

- **Improved Correlation with Human Ratings**: Compared to state-of-the-art unsupervised evaluation metrics, SUPERT demonstrates better correlation with human ratings, showing an improvement of 18-39%. Additionally, it can be used as rewards to guide neural-based reinforcement learning summarizers, leading to favorable performance compared to existing unsupervised summarizers.

2. BLANC (Reference : https://paperswithcode.com/method/blanc, https://arxiv.org/pdf/2002.09836v2)

- **Introduction of BLANC**: BLANC is introduced as an alternative to the ROUGE family of summary quality estimators, aiming to directly assess how well a summary aids a pre-trained language model in its language understanding task.
- **Definition of BLANC**: BLANC measures the effectiveness of a summary by evaluating how well it assists an independent, pre-trained language model, specifically focusing on the masked token task (Cloze task) where the model reconstructs obscured spans of text. BERT, a well-known language model pre-trained for predicting masked text tokens, is utilized for this purpose.
- **Two Versions of BLANC**: BLANC is presented in two versions: BLANC-help and BLANC-tune. BLANC-help involves directly concatenating the summary text with each document sentence during inference, while BLANC-tune utilizes the summary text to fine-tune the language model before processing the entire document.
- **BLANC-help**: This version of BLANC incorporates the summary text into the inference process by directly concatenating it with each document sentence, allowing the language model to benefit from the summary's information.
- **BLANC-tune**: In contrast, BLANC-tune leverages the summary text to fine-tune the language model before processing the entire document, potentially enhancing the model's understanding and performance on the document.



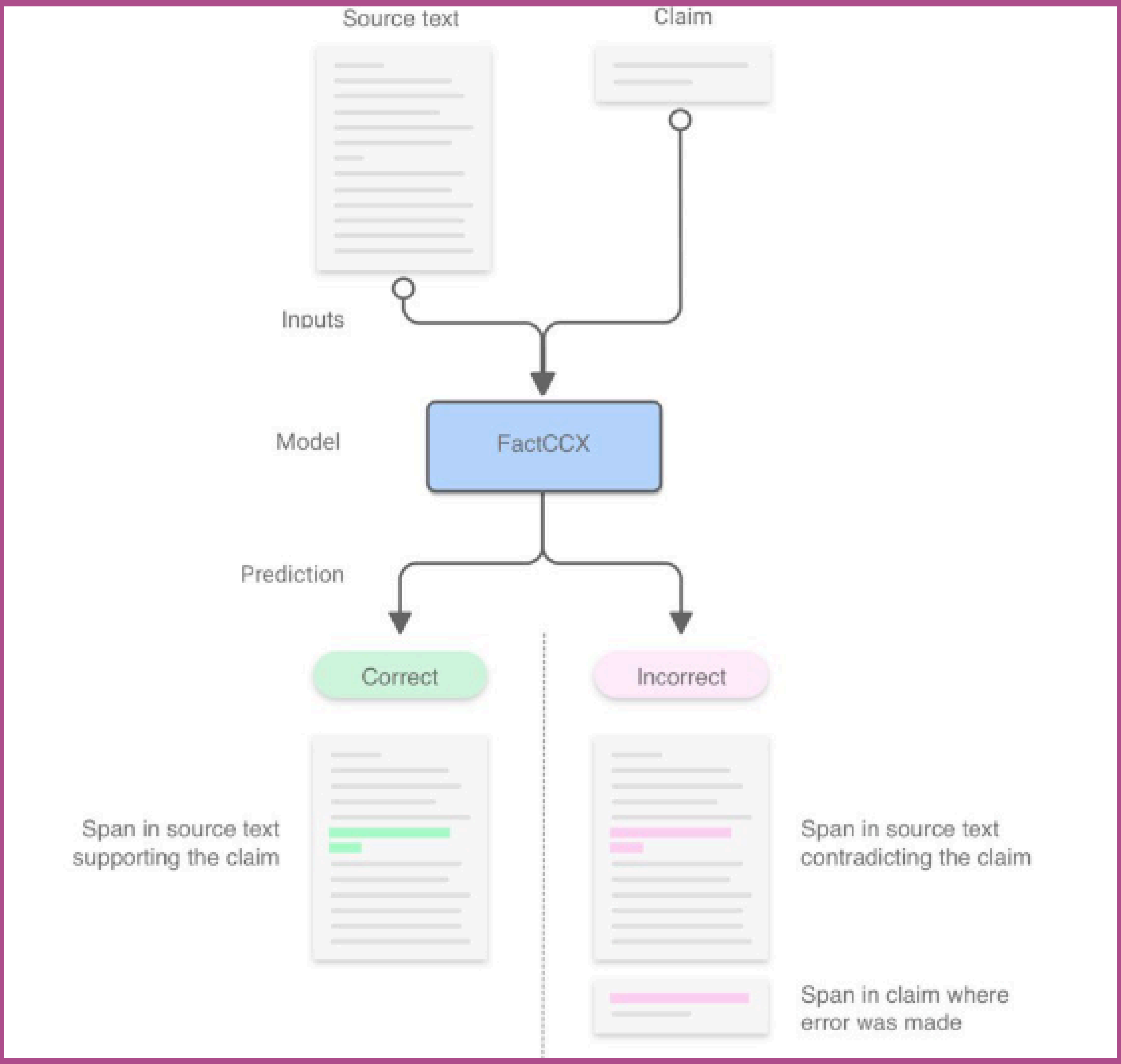
Estimator	Correlation	L	C
BLANC-help	Pearson	0.20	0.77
	Spearman	0.19	0.73
humans	Pearson	0.42	0.41
	Spearman	0.38	0.39

Table 2: Correlation of different quality estimators with length L of summary and with compression C. Based on randomly selected daily news documents.

Reference :https://arxiv.org/pdf/2002.09836v2

3. FACTCC (Reference : https://github.com/salesforce/factCC)

- **Approach Introduction**: The paper proposes a new method to check if summaries accurately represent information from source documents, which is often overlooked in current evaluation metrics.
- **Training Data Creation**: They create training data by applying rules to transform sentences in source documents.
- **Joint Training for Multiple Tasks**: Their model is trained to do three things together: decide if sentences stay accurate after transformation, find supporting evidence in source documents, and locate inconsistencies in summaries.
- **Improved Performance**: Testing shows their method works better than previous ones, even those trained with more supervision and using standard datasets.
- **Validation Through Human Assessment**: Human review confirms the helpfulness of additional tasks in accurately checking factual consistency, proving the effectiveness of their method.


Source text
Claim
Inputs
Model
FactCCX
Prediction
Correct
Incorrect
Span in source text supporting the claim
Span in source text contradicting the claim
Span in claim where error was made

Reference : https://arxiv.org/abs/1910.12840

OTHER METRICS

1. PERPLEXITY (Reference : https://huggingface.co/spaces/evaluate-metric/perplexity)

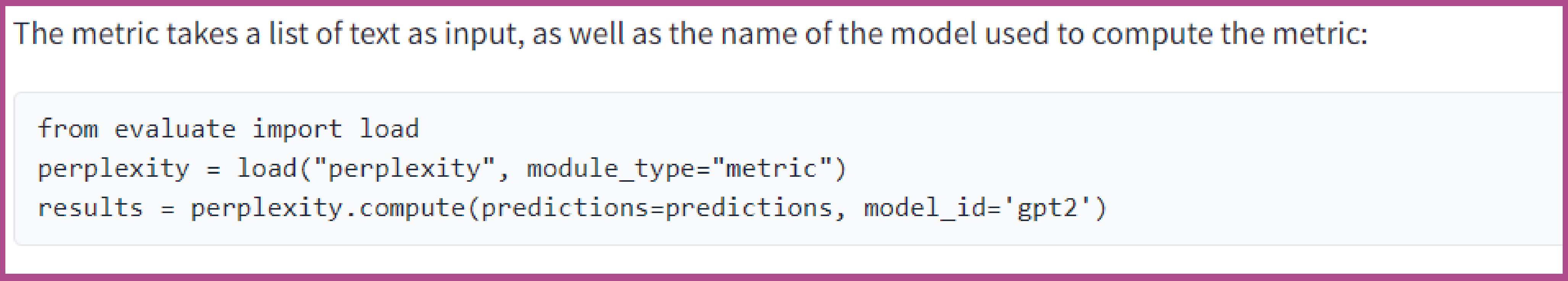
- Perplexity Definition: Perplexity (PPL) is a common metric for assessing language models, representing the average negative log-likelihood of a sequence exponentiated with base e. It measures the likelihood of a model generating a given input text sequence.

- Evaluation of Model Learning: Perplexity is utilized to evaluate how effectively a language model has learned the distribution of the text it was trained on, providing insights into the model's performance and ability to generate coherent text.

- Calculation Method: Perplexity is calculated by exponentiating the average of negative log-likelihood losses divided by the number of tokenized tokens in the input text sequence, following recent conventions in deep learning frameworks. perplexity = e**(sum(losses) / num_tokenized_tokens)

- Intuitively, perplexity means to be surprised. We measure how much the model is surprised by seeing new data. The lower the perplexity, the better the training is.

How to Use



The metric takes a list of text as input, as well as the name of the model used to compute the metric:

```
from evaluate import load
perplexity = load("perplexity", module_type="metric")
results = perplexity.compute(predictions=predictions, model_id='gpt2')
```

Inputs

- model_id (str): model used for calculating Perplexity. NOTE: Perplexity can only be calculated for causal language models.
  - This includes models such as gpt2, causal variations of bert, causal versions of t5, and more (the full list can be found in the AutoModelForCausalLM documentation here: https://huggingface.co/docs/transformers/master/en/model_doc/auto#transformers.AutoModelForCausalLM)
- predictions (list of str): input text, where each separate text snippet is one list entry.
- batch_size (int): the batch size to run texts through the model. Defaults to 16.
- add_start_token (bool): whether to add the start token to the texts, so the perplexity can include the probability of the first word. Defaults to True.
- device (str): device to run on, defaults to cuda when available

Output Values

- This metric outputs a dictionary with the perplexity scores for the text input in the list, and the average perplexity.
- If one of the input texts is longer than the max input length of the model, then it is truncated to the max length for the perplexity computation.
- {'perplexities': [8.182524681091309, 33.42122268676758, 27.012239456176758], 'mean_perplexity': 22.871995608011883}
- The range of this metric is [0, inf). A lower score is better.
- Calculation Method: Perplexity is calculated by exponentiating the average of negative log-likelihood losses divided by the number of tokenized tokens in the input text sequence, following recent conventions in deep learning frameworks. perplexity = e**(sum(losses) / num_tokenized_tokens)
- Intuitively, perplexity means to be surprised. We measure how much the model is surprised by seeing new data. The lower the perplexity, the better the training is.

ANOTHER EXAMPLE (Reference : https://medium.com/@priyankads/perplexity-of-language-models-41160427ed72)

Let's examine the perplexity of two sentences using GPT-2 and observe the level of uncertainty conveyed by the model. We'll begin by loading a tokenizer and a causal head for the GPT-2 model from Hugging Face:

```
from transformers import AutoModelForCausalLM, AutoTokenizer
model = AutoModelForCausalLM.from_pretrained("gpt2")
tokenizer = AutoTokenizer.from_pretrained("gpt2")

inputs = tokenizer("ABC is a startup based in New York City and Paris",
 return_tensors = "pt")
loss = model(input_ids = inputs["input_ids"], labels = inputs["input_ids"]).loss
ppl = torch.exp(loss)
print(ppl)

>>>Output: 29.48

inputs_wiki_text = tokenizer("Generative Pretrained Transformer is an opensource artificial
intelligence created by OpenAI in February 2019", return_tensors = "pt")
loss = model(input_ids = inputs_wiki_text["input_ids"], labels = inputs_wiki_text["input_ids"]).loss
ppl = torch.exp(loss)
print(ppl)

>>>Output: 211.81
```

As evident, the first sentence is a sequence familiar to the model, resulting in a notably lower perplexity compared to the second sentence. Conversely, the GPT-2 model encounters the second sentence for the first time, leading to a higher perplexity as the model grapples with the unfamiliar input.