Pairwise String Comparison

Often you will want to compare predictions of an LLM, Chain, or Agent for a given input. The `StringComparison` evaluators facilitate this so you can answer questions like:

- Which LLM or prompt produces a preferred output for a given question?
- Which examples should I include for few-shot example selection?
- Which output is better to include for fintetuning?

The simplest and often most reliable automated way to choose a preferred prediction for a given input is to use the `pairwise_string` evaluator.

Check out the reference docs for the PairwiseStringEvalChain for more info.

```
from langchain.evaluation import load_evaluator

evaluator = load_evaluator("labeled_pairwise_string")
```

API Reference:

- load_evaluator from `langchain.evaluation`

```
evaluator.evaluate_string_pairs(
    prediction="there are three dogs",
    prediction_b="4",
    input="how many dogs are in the park?",
    reference="four",
)
```

```
    {'reasoning': 'Both responses are relevant to the question asked,
as they both provide a numerical answer to the question about the
number of dogs in the park. However, Response A is incorrect according
to the reference answer, which states that there are four dogs.
Response B, on the other hand, is correct as it matches the reference
answer. Neither response demonstrates depth of thought, as they both
simply provide a numerical answer without any additional information or
context. \n\nBased on these criteria, Response B is the better
response.\n',
```

```
    'value': 'B',
    'score': 0}
```

Methods

The pairwise string evaluator can be called using evaluate_string_pairs (or async aevaluate_string_pairs) methods, which accept:

- prediction (str) – The predicted response of the first model, chain, or prompt.
- prediction_b (str) – The predicted response of the second model, chain, or prompt.
- input (str) – The input question, prompt, or other text.
- reference (str) – (Only for the labeled_pairwise_string variant) The reference response.

They return a dictionary with the following values:

- value: 'A' or 'B', indicating whether `prediction` or `prediction_b` is preferred, respectively
- score: Integer 0 or 1 mapped from the 'value', where a score of 1 would mean that the first `prediction` is preferred, and a score of 0 would mean `prediction_b` is preferred.
- reasoning: String "chain of thought reasoning" from the LLM generated prior to creating the score

Without References

When references aren't available, you can still predict the preferred response. The results will reflect the evaluation model's preference, which is less reliable and may result in preferences that are factually incorrect.

```
from langchain.evaluation import load_evaluator

evaluator = load_evaluator("pairwise_string")
```

API Reference:

- load_evaluator from `langchain.evaluation`

```
evaluator.evaluate_string_pairs(
    prediction="Addition is a mathematical operation.",
```

```
    prediction_b="Addition is a mathematical operation that adds two
numbers to create a third number, the 'sum'.",
    input="What is addition?",
)
```

```
    {'reasoning': 'Both responses are correct and relevant to the
question. However, Response B is more helpful and insightful as it
provides a more detailed explanation of what addition is. Response A is
correct but lacks depth as it does not explain what the operation of
addition entails. \n\nFinal Decision: [[B]]',
     'value': 'B',
     'score': 0}
```

Defining the Criteria

By default, the LLM is instructed to select the 'preferred' response based on helpfulness, relevance, correctness, and depth of thought. You can customize the criteria by passing in a `criteria` argument, where the criteria could take any of the following forms:

- `Criteria` enum or its string value - to use one of the default criteria and their descriptions
- Constitutional principal - use one any of the constitutional principles defined in langchain
- Dictionary: a list of custom criteria, where the key is the name of the criteria, and the value is the description.
- A list of criteria or constitutional principles - to combine multiple criteria in one.

Below is an example for determining preferred writing responses based on a custom style.

```
custom_criteria = {
    "simplicity": "Is the language straightforward and unpretentious?",
    "clarity": "Are the sentences clear and easy to understand?",
    "precision": "Is the writing precise, with no unnecessary words or
details?",
    "truthfulness": "Does the writing feel honest and sincere?",
    "subtext": "Does the writing suggest deeper meanings or themes?",
}
evaluator = load_evaluator("pairwise_string", criteria=custom_criteria)
```

```
evaluator.evaluate_string_pairs(
    prediction="Every cheerful household shares a similar rhythm of
joy; but sorrow, in each household, plays a unique, haunting melody.",
    prediction_b="Where one finds a symphony of joy, every domicile of
happiness resounds in harmonious,"
    " identical notes; yet, every abode of despair conducts a dissonant
orchestra, each"
    " playing an elegy of grief that is peculiar and profound to its
own existence.",
    input="Write some prose about families.",
)
```

```
    {'reasoning': 'Response A is simple, clear, and precise. It uses
straightforward language to convey a deep and sincere message about
families. The metaphor of joy and sorrow as music is effective and easy
to understand.\n\nResponse B, on the other hand, is more complex and
less clear. The language is more pretentious, with words like
"domicile," "resounds," "abode," "dissonant," and "elegy." While it
conveys a similar message to Response A, it does so in a more
convoluted way. The precision is also lacking due to the use of
unnecessary words and details.\n\nBoth responses suggest deeper
meanings or themes about the shared joy and unique sorrow in families.
However, Response A does so in a more effective and accessible
way.\n\nTherefore, the better response is [[A]].',
     'value': 'A',
     'score': 1}
```

Customize the LLM

By default, the loader uses `gpt-4` in the evaluation chain. You can customize this when loading.

```
from langchain.chat_models import ChatAnthropic

llm = ChatAnthropic(temperature=0)

evaluator = load_evaluator("labeled_pairwise_string", llm=llm)
```

API Reference:

- ChatAnthropic from `langchain.chat_models`

```
evaluator.evaluate_string_pairs(
    prediction="there are three dogs",
    prediction_b="4",
    input="how many dogs are in the park?",
    reference="four",
)
```

```
    {'reasoning': 'Here is my assessment:\n\nResponse B is more
helpful, insightful, and accurate than Response A. Response B simply
states "4", which directly answers the question by providing the exact
number of dogs mentioned in the reference answer. In contrast, Response
A states "there are three dogs", which is incorrect according to the
reference answer. \n\nIn terms of helpfulness, Response B gives the
precise number while Response A provides an inaccurate guess. For
relevance, both refer to dogs in the park from the question. However,
Response B is more correct and factual based on the reference answer.
Response A shows some attempt at reasoning but is ultimately incorrect.
Response B requires less depth of thought to simply state the factual
number.\n\nIn summary, Response B is superior in terms of helpfulness,
relevance, correctness, and depth. My final decision is: [[B]]\n',
     'value': 'B',
     'score': 0}
```

Customize the Evaluation Prompt

You can use your own custom evaluation prompt to add more task-specific instructions or to instruct the evaluator to score the output.

*Note: If you use a prompt that expects generates a result in a unique format, you may also have to pass in a custom output parser (`output_parser=your_parser()`) instead of the default `PairwiseStringResultOutputParser`

```
from langchain.prompts import PromptTemplate

prompt_template = PromptTemplate.from_template(
    """Given the input context, which do you prefer: A or B?
Evaluate based on the following criteria:
{criteria}
Reason step by step and finally, respond with either [[A]] or [[B]] on
its own line.
```

```
DATA
----
input: {input}
reference: {reference}
A: {prediction}
B: {prediction_b}
---
Reasoning:

"""
)
evaluator = load_evaluator(
    "labeled_pairwise_string", prompt=prompt_template
)
```

API Reference:

- PromptTemplate from `langchain.prompts`

```
# The prompt was assigned to the evaluator
print(evaluator.prompt)
```

```
    input_variables=['prediction', 'reference', 'prediction_b',
'input'] output_parser=None partial_variables={'criteria':
'helpfulness: Is the submission helpful, insightful, and appropriate?
\nrelevance: Is the submission referring to a real quote from the text?
\ncorrectness: Is the submission correct, accurate, and factual?
\ndepth: Does the submission demonstrate depth of thought?'}
template='Given the input context, which do you prefer: A or B?
\nEvaluate based on the following criteria:\n{criteria}\nReason step by
step and finally, respond with either [[A]] or [[B]] on its own
line.\n\nDATA\n----\ninput: {input}\nreference: {reference}\nA:
{prediction}\nB: {prediction_b}\n---\nReasoning:\n\n'
template_format='f-string' validate_template=True
```

```
evaluator.evaluate_string_pairs(
    prediction="The dog that ate the ice cream was named fido.",
    prediction_b="The dog's name is spot",
    input="What is the name of the dog that ate the ice cream?",
    reference="The dog's name is fido",
)
```

```
    {'reasoning': 'Helpfulness: Both A and B are helpful as they
provide a direct answer to the question.\nRelevance: A is relevant as
it refers to the correct name of the dog from the text. B is not
relevant as it provides a different name.\nCorrectness: A is correct as
it accurately states the name of the dog. B is incorrect as it provides
a different name.\nDepth: Both A and B demonstrate a similar level of
depth as they both provide a straightforward answer to the
question.\n\nGiven these evaluations, the preferred response is:\n',
     'value': 'A',
     'score': 1}
```