LLM Qualitative Sort

Pythonパッケージ 要件定義書 v2.0

作成日: 2025年12月9日

変更履歴

1. パッケージ概要

1.1 パッケージ名

llm-qualitative-sort

1.2 目的

LLMを用いて、定量的に比較できない評価観点（文章の良さ、キャラクターの強さなど）に基づき、複数のテキストデータをマルチイリミネーション方式で順位付けする。

1.3 利用形態

Pythonライブラリとしてプログラムから呼び出す形式

2. コア機能

2.1 ペア比較機能

LLMに2つのテキストを提示し、どちらが優れているかを評価させる。

2.2 マルチイリミネーショントーナメント

ダブルイリミネーション（**N=2**）

ウィナーズブラケット（無敗）とルーザーズブラケット（1敗）の2つのトーナメント

2回負けると敗退

n人参加時: 2n-2 〜 2n-1 試合

マルチイリミネーション（**N=**任意）

N回負けるまで敗退しない

負け数ごとにブラケットが存在（0敗、1敗、2敗、...、N-1敗）

n人参加時: Nn-N 〜 Nn-1 試合

初期配置はランダム（シード機能なし）

2.3 順位決定ルール

最終順位は勝利数で決定

同じ勝利数の場合は同順位

上位ほど細かく順位が決まり、下位ほど大きな塊になる

この方式は上位の順位付け精度が高い特徴を持つ

3. 非機能要件

3.1 並列処理

asyncioベースの非同期処理

セマフォによる同時リクエスト数制御

max_concurrent_requests パラメータで上限を指定

3.2 進捗コールバック

トーナメントの進行状況を通知するコールバック機能を提供する。

イベント種別:

MATCH_START: 試合開始

MATCH_END: 試合終了（結果含む）

ROUND_END: ラウンド終了

BRACKET_CHANGE: ブラケット移動

3.3 キャッシング

同一ペア・同一評価基準の場合はキャッシュを再利用

順序（A→B または B→A）も区別してキャッシュ

キャッシュキー: (item_a_hash, item_b_hash, criteria_hash, order)

ファイルキャッシュとメモリキャッシュの両方をサポート

4. LLMプロバイダー対応

4.1 対応プロバイダー（初期リリース）

4.2 プロバイダー設定パラメータ

4.3 デフォルトエンドポイント

OpenAI: https://api.openai.com/v1

Google: https://generativelanguage.googleapis.com/v1beta

4.4 プロバイダーインターフェース

class LLMProvider(ABC):

def __init__(self, api_key: str,

```
base_url: str | None = None,
model: str = "default"):
...
@abstractmethod
async def compare(self, item_a: str, item_b: str,
criteria: str) -> ComparisonResult:
pass
```

4.5 使用例

```
# OpenAI
provider = OpenAIProvider(
api_key="sk-...",
model="gpt-5.1"
)
# Google
provider = GoogleProvider(
api_key="AIza...",
model="gemini-3.0"
)
# カスタムエンドポイント
provider = OpenAIProvider(
api_key="...",
base_url="https://my-proxy.example.com/v1",
model="gpt-5.1"
)
```

5. API設計

5.1 メインクラス: QualitativeSorter

6. データ構造

6.1 ComparisonResult

@dataclass

class ComparisonResult:

winner: str | None # "A", "B", or None (error)

reasoning: str

raw_response: dict

6.2 MatchResult

@dataclass

class MatchResult:

item_a: str

item_b: str

winner: str | None # "A", "B", or None (draw)

rounds: list[RoundResult]

6.3 RoundResult

@dataclass

class RoundResult:

order: str # "AB" or "BA"

winner: str # "A" or "B"

reasoning: str

cached: bool

6.4 SortResult

@dataclass

class SortResult:

rankings: list[tuple[int, list[str]]] # [(rank, [items])]

match_history: list[MatchResult]

statistics: Statistics

6.5 Statistics

@dataclass

class Statistics:

total_matches: int

total_api_calls: int

cache_hits: int

elapsed_time: float

6.6 ProgressEvent

@dataclass

class ProgressEvent:

type: EventType # MATCH_START, MATCH_END, etc.

message: str

completed: int

total: int

data: dict | None

7. テスト要件

7.1 テスト方針

LLM部分をモック化し、決定論的かつ再現可能なテストを実施する。定性的評価の不確実性をシミュレートするため、確率的なノイズを導入する。

7.2 モックLLMプロバイダー

基本仕様

テストデータ: "0" 〜 "999" の数値テキスト（1000件）

比較方法: テキストを整数にパースして数値比較

期待される正解順位: 999 > 998 > ... > 1 > 0（降順）

ノイズシミュレーション

定性的評価において、必ずしも「強いもの」が勝つわけではないことをシミュレートするため、比較時にノイズを加える。

比較ロジック

def mock_compare(item_a: str, item_b: str) -> str:

value_a = int(item_a) + random.gauss(0, 3.33)

value_b = int(item_b) + random.gauss(0, 3.33)

標準偏差3.33で約99.7%が±10の範囲に収まる

return "A" if value_a > value_b else "B"

結果: 数値差が小さいペア（例: 500 vs 502）は逆転しやすく、数値差が大きいペア（例: 100 vs 900）は逆転しにくい

7.3 精度測定指標

7.4 N値による精度比較テスト

elimination_count（N）を1〜10まで変化させ、ソーティング精度が向上することを確認する。

テスト手順

テストデータ（"0"〜"999"）を準備

各N値（1〜10）でソーティングを実行

各実行について精度指標を計算

N値と精度の関係をグラフ化

N増加に伴い精度が向上することを確認

期待される結果

N=1（シングルイリミネーション）: 最も低い精度

N=2（ダブルイリミネーション）: N=1より向上

N増加: 精度が単調増加（収束する傾向）

上位の精度 > 下位の精度（Top-10がTop-100より高精度）

テストコード例

```
import pytest
from scipy.stats import kendalltau
@pytest.mark.parametrize("n", range(1, 11))
async def test_accuracy_improves_with_n(n: int):
provider = MockLLMProvider(seed=42)
sorter = QualitativeSorter(
provider=provider,
elimination_count=n,
criteria="larger is better"
```

```
)
items = [str(i) for i in range(1000)]
result = await sorter.sort(items)
# 精度計算
actual_order = flatten_rankings(result.rankings)
expected_order = [str(i) for i in range(999, -1, -1)]
tau, _ = kendalltau(actual_order, expected_order)
return {"n": n, "kendall_tau": tau}
```

7.5 再現性

モックプロバイダーにシード値を設定可能

同じシードで同じ結果が得られることを保証

CIで再現可能なテストを実現

8. 使用例

```
import asyncio
from llm_qualitative_sort import (
QualitativeSorter, OpenAIProvider,
FileCache, ProgressEvent
)
async def main():
provider = OpenAIProvider(
api_key="sk-...",
model="gpt-5.1"
)
def on_progress(event: ProgressEvent):
print(f"[{event.type}] {event.message}")
```

```
sorter = QualitativeSorter(
provider=provider,
elimination_count=3,
comparison_rounds=4,
criteria="どちらの文章がより説得力があるか",
max_concurrent_requests=10,
cache=FileCache("./cache"),
on_progress=on_progress
)
items = ["テキストA", "テキストB", "テキストC"]
result = await sorter.sort(items)
for rank, items in result.rankings:
print(f"{rank}位: {items}")
asyncio.run(main())
```

9. 補足: マルチイリミネーション方式について

9.1 概要

マルチイリミネーション方式は、シングルイリミネーション（1回負けで敗退）を拡張したトーナメント方式である。N回負けるまで敗退しないため、運の要素が減り、より実力を反映した結果が得られる。

9.2 ダブルイリミネーションの構造

ウィナーズブラケット（Winners Bracket）：まだ1度も負けていない参加者のトーナメント

ルーザーズブラケット（Losers Bracket）：1度負けた参加者のトーナメント

グランドファイナル: 両ブラケットの勝者が対戦。ウィナーズ側が負けた場合はリセット（再戦）となる場合がある

9.3 本パッケージでの適用

本パッケージでは、完全なダブルイリミネーションのトーナメント構造ではなく、「勝利数」を基準とした簡略化されたマルチイリミネーション方式を採用する。これにより:

実装の複雑さを軽減

上位の順位付け精度を維持

下位は大まかなグループ分けで十分という要件に対応

— End of Document —

版	日付	変更内容
v1.0	2025/12/9	初版作成
v2.0	2025/12/9	LLMプロバイダー更新、テスト要件追加

項目	仕様
評価回数	偶数回（デフォルト: 2回）
順序バイアス対策	提示順を入れ替えた独立セッションで評価
勝敗判定	多く勝った方が勝利、同数なら引き分け
引き分け時	両者に1敗を追加（敗退ではない）
評価軸	ユーザーがカスタマイズ可能
プロンプトテンプレート	パッケージ内部で固定（カスタマイズ不可）

プロバイダー	クラス名	最新モデル
OpenAI	OpenAIProvider	GPT-5.1
Google	GoogleProvider	Gemini 3.0

パラメータ	型	説明

api_key	str	APIキー（必須）
base_url	str | None	エンドポイントURL（オプション、デフォルト値あり）
model	str	使用するモデル名

パラメータ	型	説明
provider	LLMProvider	LLMプロバイダーインスタンス
elimination_count	int	N回負けで敗退（デフォルト: 2）
comparison_rounds	int	各ペアの評価回数（偶数、デフォルト: 2）
criteria	str	評価基準（例: 「どちらが説得力があるか」）
max_concurrent_requests	int	同時リクエスト上限
cache	Cache | None	キャッシュインスタンス（オプション）
on_progress	Callable | None	進捗コールバック関数（オプション）

項目	仕様
ノイズ分布	正規分布（ガウス分布）
ノイズ範囲	-10 〜 +10
適用対象	比較される2つの値それぞれに独立して適用

指標	説明

Kendall's τ	順位相関係数。-1〜1の値で、1が完全一致。全体的な順位の正確さを測定
Top-K 精度	上位K件が正しく含まれている割合。K=10, 50, 100で測定
正順ペア率	全ペアのうち、正しい順序になっているペアの割合