OpenAI API を使った 自然言語からの KG 構築

第60回SWO研究会

2023/08/24

**いかなる損害やトラブルの責任は一切負いかねます．
あらかじめご了承ください．
OpenAI APIは有料です．そのほかコマンドなどの実行についても，内容をご理解の上，実行は自己責任でお願いいたします．**

本資料について

- ナレッジ推論チャレンジ2023 のタスクに OpenAI APIを用いて取り組んだ例です.
- チャレンジに参加される際の参考なれば幸いです.

タスク設定

FUJITSU

- 目的：
  - 自然言語から推論チャレンジ KG の triples（各シーンのみ）を生成.
  - シーン間の関係は扱わない.
- 入力：ヘレンがホームズの家にやってくる ->
- 出力：

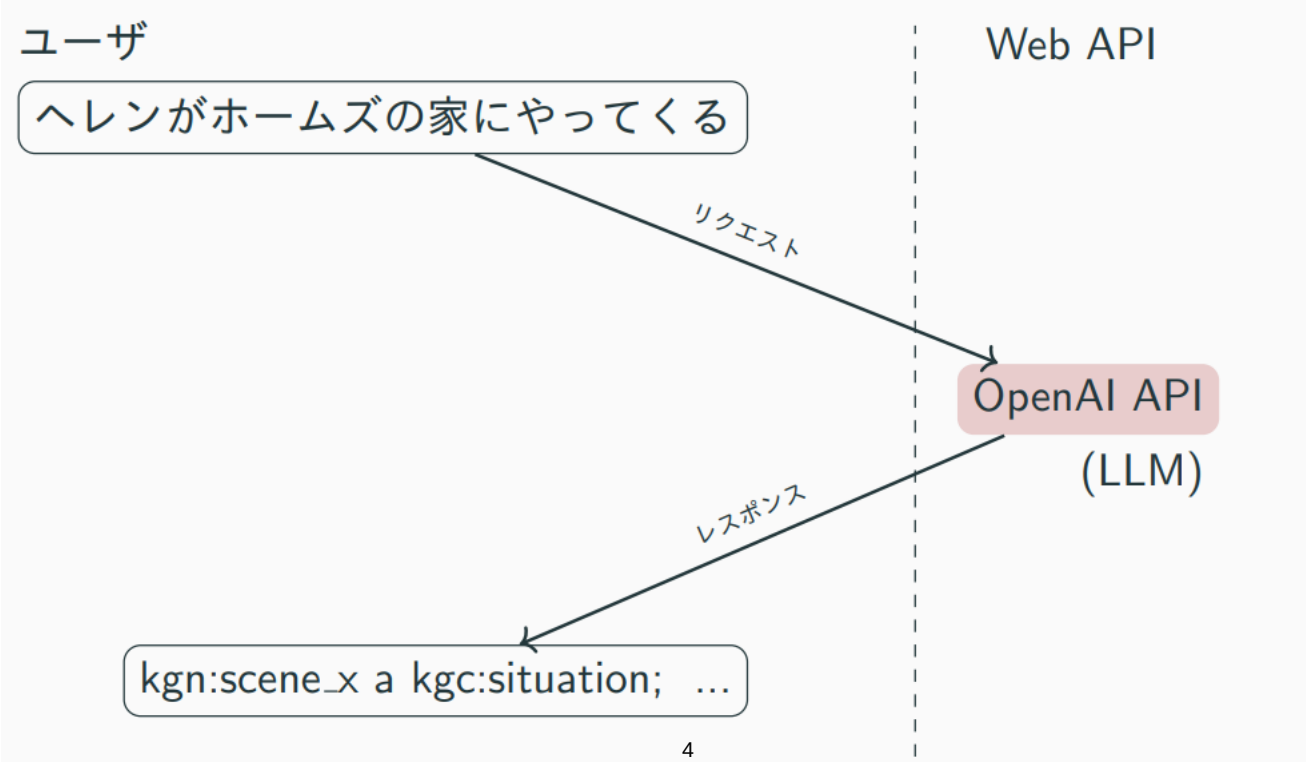
```
kgn:scene_x a kgc:situation ;
            kgc:subject kgn:helen ;
            kgc:haspredicate kgp:come ;
            kgc:to kgn:house_of_holmes .
```

- 本日の内容：
  - OpenAI API を使って上記の問題を解いてみる

OpenAI API で何ができるようになるか

FUJITSU

- 自然言語→ KG の変換を WebAPI で処理可能に！



作業の流れ

- 学習データを作成
- OpenAI API 使用準備
- モデルの学習・推論の実施（WebAPI 呼び出し）
- モデルの評価

とても簡単

学習データ作成

KGは ここ からダウンロードできます

FUJITSU

```
kgn:1
   a                    kgc:Situation;
   kgc:source           "ヘレンがホームズの家にやってくる"@ja ;   ← 入力
   kgc:source           "Helen␣comes␣to␣Holmes'␣house"@en ;
   kgc:hasPredicate     kgp:come ;
   kgc:subject          kgn:Helen ;
   kgc:to               kgn:house_of_Holmes .
```

『まだらの紐』シーン１

- 上記データを加工
  - 入力は各シーンのソース文
  - シーン ID は kgn:scene x で統一
  - シーン間の関係を表すトリプル と source は削除
  - 大文字は小文字に変換
  - すべて prefix 付きで表現（生の URI がデータ中に現れないようにする）

学習データ作成

- 入力：ヘレンがホームズの家にやってくる ->
- 出力：

```
kgn:scene_x a kgc:situation ;
            kgc:subject kgn:helen ;
            kgc:haspredicate kgp:come ;
            kgc:to kgn:house_of_holmes .
```

加工後の
『まだらの紐』シーン１

- 加工内容
  - 入力は各シーンのソース文
  - シーン ID は kgn:scene x で統一
  - シーン間の関係を表すトリプル と source は削除
  - 大文字は小文字に変換
  - すべて prefix 付きで表現（生の URI がデータ中に現れないようにする）

学習データ作成

FUJITSU

- 学習データを OpenAI API 用（JSON Lines 形式）に変換.
- 各行のフォーマット：

```
{
  "prompt": 入力文,
  "completion": 正解（出力してほしい文）
}
```

JSON Lines なので，実際には改行は含まない

- 例：

```
{
  "prompt": "ヘレンがホームズの家にやってくる␣->",
  "completion": "␣kdn:scene_x␣kgc:haspredicate␣kdp:arrive␣;␣
    kgc:subject␣kdn:helen␣;␣kgc:time␣\"1883-04-01t07:00:00\"^^
    xsd:datetime␣;␣kgc:to␣kdn:house_of_holmes␣;␣kgc:when␣kdn
    :1883-04-01t07␣;␣rdf:type␣kgc:situation␣."
}
```

JSON Lines なので，実際には改行は含まない

OpenAI API 使用準備

FUJITSU

- OpenAI への登録 と API Key の取得
  すでに記事がたくさんあるので，そちらを参考にしてください.
- API Key を設定
  - `export OPENAI_API_KEY='sk-...'`
- openai-python をインストール
  - `pip install --upgrade openai`
- 動作確認
  - `openai api models.list`
- openai コマンドの詳細は，openai-python を参照してください.

OpenAI API 使用準備

FUJITSU

- Fine-tuning API の費用．最新情報は公式ページで確認してください． 処理されたトークン数に応じて費用が発生します．

Model	Training	Usage
Ada	$0.0004 / 1K tokens	$0.0016 / 1K tokens
Babbage	$0.0006 / 1K tokens	$0.0024 / 1K tokens
Curie	$0.0030 / 1K tokens	$0.0120 / 1K tokens
Davinci	$0.0300 / 1K tokens	$0.1200 / 1K tokens

- 1k tokens の目安 → 日本語：600 文字，英語：1,000 単語
  - **テキスト次第で変化しますので，参考程度にお考え下さい．**

OpenAI API 使用準備

FUJITSU

- 2023/08/23 に 新しい Fine-tuning API が公開されました．費用が下がり，GPT-3.5 も対象となっています．最新情報は公式ページで確認してください．

Model	Training	Input usage	Output usage
babbage-002	$0.0004 / 1K tokens	$0.0016 / 1K tokens	$0.0016 / 1K tokens
davinci-002	$0.0060 / 1K tokens	$0.0120 / 1K tokens	$0.0120 / 1K tokens
GPT-3.5 Turbo	$0.0080 / 1K tokens	$0.0120 / 1K tokens	$0.0160 / 1K tokens

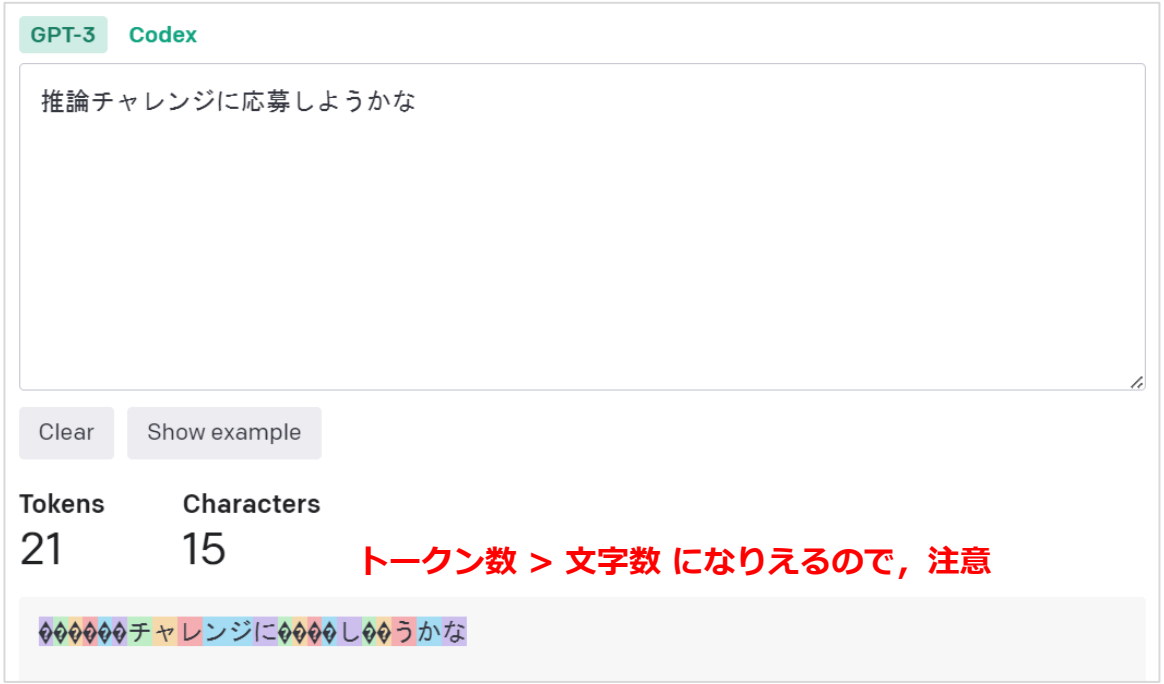
※ 今回紹介する方法では，前ページの条件で費用がかかるかもしれません．

トークン数について

OpenAI APIでこのトークナイザが使われているかどうかはわかりません．参考程度にお考え下さい．

FUJITSU

- トークン数は，OpenAI のデモ が参考になるかもしれません．



モデルの学習（WebAPI 呼び出し）

FUJITSU

- 以下のようなコマンドを呼びだし，モデルを学習.

```
openai api fine_tunes.create \
  -t path/to/your/train.data.jsonl \  # 用意した学習データへのパス
  -m davinci \                        # 学習を行うモデル
  --suffix 'ja-1epoch' \              # 学習後のモデルにつける接尾語
  --n_epochs 1 \                      # 学習データ全体を何回処理するか（費用に直結）
```

- 学習に時間がかかります．進捗は，上記コマンド実行時に得られる識別 ID を用いて，
  - `openai api fine_tunes . follow -i 識別ID`
- で確認可能です

New Finetuning API

FUJITSU

- Python スクリプトから簡単に新しい Fine-tuning API を利用できそうです.
  - 詳細は GitHub リポジトリ などをご参照ください.

```
# Create a fine-tuning job with an already uploaded file
openai.FineTuningJob.create(
  training_file="file-abc123",
  model="gpt-3.5-turbo"
)
```

モデルの学習（WebAPI 呼び出し）

FUJITSU

- `openai api fine_tunes.follow -i 識別ID` 実行後,

```
Job complete! Status: succeeded
Try out your fine-tuned model:

openai api completions.create -m 学習後モデルの名前 -p <YOUR_PROMPT>
```

- と表示されたら学習完了です.

推論の実施（WebAPI 呼び出し）

FUJITSU

- 以下のコマンドで，学習後のモデル使った自然言語→ triples の生成を実施.

```
openai api completions.create \
  --max-tokens 512 \                          # 生成する最大系列長を指定
  -m モデル名 \                                # 使用するモデルを指定
  -p "ヘレンがホームズの家にやってくる␣->" \    # 入力文を指定
  --stop '.'                                  # 指定した文字列が生成された場合，生成終了
```

- 以下のような出力が得られる.

```
ヘレンがホームズの家にやってくる
     -> kgn:scene_x kgc:infosource kgn:holmes ; a kgc:statement ; kgc:subject
       kgn:helen ; kgc:haspredicate kgp:travel ; kgc:obj kgn:house_of_holmes .
```

推論の実施（WebAPI 呼び出し）

FUJITSU

- 以下のようなコードで，Python スクリプトからも簡単に呼び出せます.

```
import openai
response = openai.Completion.create(
  engine=engine,   # モデル名
  prompt=prompt,   # 入力文
  max_tokens=512,  # 最大生成系列長
  stop='.'         # 指定した文字列が生成された場合，生成終了
)
print("生成結果：")
print(response['choices'])
```

- 引数や返り値の詳細は API Reference を参照してください.

FUJITSU

実験・評価など

実験設定

- 入力：ヘレンがホームズの家にやってくる ->
- 出力：

```
kgn:scene_x a kgc:situation ;
            kgc:subject kgn:helen ;
            kgc:haspredicate kgp:come ;
            kgc:to kgn:house_of_holmes .
```

- モデル：Davinci （古い API で，一番良いモデル）
- 学習用データ：まだらの紐以外の 7 小説 (2,632 シーン)
- テストデータ：まだらの紐 (396 シーン)
- 費用：約$7/epoch（古い API で）

評価指標

- Loadability: Turtle として読み込み可能な割合
- Precision：生成されたものが正解のシーンにも存在する割合
- Recall：正解のシーンにあるものが，生成されている割合
- F1：Precision, Recall の調和平均

Precision

FUJITSU

Generated

```
kgn:scene_x kgc:subject kgn:helen ;          o
            kgc:subject kgn:holmes ;         } 3
            kgc:haspredicate kgp:come.       o
```

Original

```
kgn:scene_x kgc:subject kgn:helen ;
            kgc:to kgn:house_of_holmes ;
            kgc:haspredicate kgp:come ;
            a kgc:situation .
```

$$\text{precision} = \frac{2}{3}$$

Recall

FUJITSU

Generated

```
kgn:scene_x kgc:subject kgn:helen ;          o
            kgc:subject kgn:holmes ;
            kgc:haspredicate kgp:come.       o
```

Original

```
kgn:scene_x kgc:subject kgn:helen ;
            kgc:to kgn:house_of_holmes ;
            kgc:haspredicate kgp:come ;
            a kgc:situation .
```

} 4

$$\text{recall} = \frac{2}{4}$$

F1

FUJITSU

Generated

```
kgn:scene_x kgc:subject kgn:helen ;          o
            kgc:subject kgn:holmes ;
            kgc:haspredicate kgp:come.       o
```

Original

```
kgn:scene_x kgc:subject kgn:helen ;
            kgc:to kgn:house_of_holmes ;
            kgc:haspredicate kgp:come ;
            a kgc:situation .
```

$3 + 4$

$$\mathrm{f1} = \frac{2 * \mathrm{precision} * \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

性能分析

日本語ソース文を入力文とし，3 epochs 学習したモデル

	F1	Precision	Recall	Loadbility
ja 3 epochs	0.356	0.410	0.325	1.00
en 3 epochs	0.405	0.471	0.367	1.00
ja 1 epoch	0.336	0.388	0.305	0.997

Loadbility: Turtle として読み込める割合.

- ほとんど Turtle として読み込める
- 日本語より英語のほうが性能がよい
- 全体的にそれほど性能は高くない

性能分析

- Predicate のみを用いて評価

	F1	Precision	Recall	Loadbility
ja 3 epochs	0.728	0.827	0.674	1.00
en 3 epochs	0.722	0.836	0.660	1.00
ja 1 epoch	0.707	0.809	0.650	0.997

Loadbility: Turtle として読み込める割合.

- Predicate 単体だと，日本語モデルと英語モデルの性能は拮抗
- それなりに性能が出ている

性能分析

- Object のみを用いて評価

	F1	Precision	Recall	Loadbility
ja 3 epochs	0.385	0.444	0.351	1.00
en 3 epochs	0.438	0.512	0.395	1.00
ja 1 epoch	0.367	0.425	0.333	0.997

Loadbility: Turtle として読み込める割合.

- 日本語モデルより英語モデルのほうが性能がよい
- 全体的にそれほど性能は高くない
- Object の生成性能が triples 生成性能のボトルネックになっている

Predicate ごとの性能評価

FUJITSU

- Precision:
  - 対象 predicate を含む triples のみ評価対象とし，シーンごとに完全一致で評価後，平均を計算
  - **生成データ**に対象 predicate がない場合は評価対象外
- Recall:
  - 対象 predicate を含む triples のみ評価対象とし，シーンごとに完全一致で評価後，平均を計算
  - **正解データ**に対象 predicate がない場合は評価対象外
- F1
  - Precision と Recall の調和平均

Predicate ごとの性能評価の例

FUJITSU

- 例：kgc:subject に対して評価を実施する場合，

 # 正解

```
kgn:scene_x
  a kgc:situation ;
  kgc:subject kgn:helen ;
  kgc:haspredicate kgp:come ;
  kgc:to kgn:house_of_holmes .
```

 # 生成

```
kgn:scene_x
  a kgc:situation ;
  kgc:subject kgn:helen ;
  kgc:subject kgn:roylott ;
  kgc:haspredicate kgp:come ;
  kgc:to kgn:house_of_holmes .
```

Predicate が **kgc:subjct である triples** のみに限定し，triples の完全一致(binary)で評価
※ この例の場合，”一致していない(0)”と評価

Predicate ごとの性能評価

各predicateを含むシーン数
#p:生成データ，#t:正解データ

FUJITSU

- Predicate と Object が評価対象

出現頻度が高く，object が入力文から生成しやすそうなものはスコアが高い

正解と生成で，出現頻度に差がある．後述する訓練データとテストデータの分布の差によるものだと考えられる

predicate	f1	precision	recall	#p	#t
rdf:type	0.717	0.717	0.717	396.000	396.000
kgc:subject	0.591	0.593	0.588	393.000	396.000
kgc:haspredicate	0.538	0.533	0.544	323.000	316.000
kgc:hasproperty	0.276	0.292	0.263	72.000	80.000
kgc:whom	0.238	0.333	0.185	15.000	27.000
kgc:what	0.257	0.242	0.274	178.000	157.000
kgc:when	0.010	0.029	0.006	70.000	318.000
kgc:time	0.000	0.000	0.000	59.000	295.000
kgc:where	0.171	0.192	0.154	73.000	91.000
kgc:from	0.157	0.200	0.129	20.000	31.000
kgc:to	0.150	0.188	0.125	32.000	48.000
kgc:how	0.175	0.250	0.135	20.000	37.000
kgc:infosource	0.016	0.013	0.020	77.000	49.000

Predicate ごとの性能評価

FUJITSU

- Predicateのみ評価対象

Predicateの生成性能は比較的高い．学習データとテストデータで出現する語彙が同じであるためだと考えられる．

#p << #t では，recallが低い（考察は後述）

predicate	f1	precision	recall	#p	#t
rdf:type	1.000	1.000	1.000	396.000	396.000
kgc:subject	0.971	0.975	0.967	393.000	396.000
kgc:haspredicate	0.914	0.904	0.924	323.000	316.000
kgc:hasproperty	0.618	0.653	0.588	72.000	80.000
kgc:whom	0.381	0.533	0.296	15.000	27.000
kgc:what	0.687	0.646	0.732	178.000	157.000
kgc:when	0.211	0.586	0.129	70.000	318.000
kgc:time	0.215	0.644	0.129	59.000	295.000
kgc:where	0.549	0.616	0.495	73.000	91.000
kgc:from	0.392	0.500	0.323	20.000	31.000
kgc:to	0.500	0.625	0.417	32.000	48.000
kgc:how	0.386	0.550	0.297	20.000	37.000
kgc:infosource	0.127	0.104	0.163	77.000	49.000

Predicate ごとの性能評価

FUJITSU

- Objectのみ評価対象

predicate	f1	precision	recall	#p	#t
rdf:type	0.717	0.717	0.717	396.000	396.000
kgc:subject	0.591	0.593	0.588	393.000	396.000
kgc:haspredicate	0.538	0.533	0.544	323.000	316.000
kgc:hasproperty	0.276	0.292	0.263	72.000	80.000
kgc:whom	0.238	0.333	0.185	15.000	27.000
kgc:what	0.257	0.242	0.274	178.000	157.000
kgc:when	0.010	0.029	0.006	70.000	318.000
kgc:time	0.000	0.000	0.000	59.000	295.000
kgc:where	0.171	0.192	0.154	73.000	91.000
kgc:from	0.157	0.200	0.129	20.000	31.000
kgc:to	0.150	0.188	0.125	32.000	48.000
kgc:how	0.175	0.250	0.135	20.000	37.000
kgc:infosource	0.016	0.013	0.020	77.000	49.000

#p << #t であっても，Object 生成の Precisionは低い.

Predcate の出現頻度について

FUJITSU

- `kgc:whem`, `kgc:time`, `kgc:inforsource` などは学習・テストデータでの出現頻度に差がある
- これらの Predicate の生成では，Recall のみ低いことから訓練データに含まれる限定的なパターンのみ学習できていると考えられる
- 対応する Object の生成では，Precision も Recall も低く，そもそも学習がうまくできていないと考えられる

出現シーン数 / 全シーン数

predicate	train	test
kgc:subject	2522 / 2632	396 / 396
kgc:when	409 / 2632	295 / 396
kgc:time	305 / 2632	318 / 396
kgc:inforsource	1317 / 2632	49 / 396

エラー例

- 語彙にない Entity が生成されてしまう
- 日本語入力の場合，自然言語→ Entity の変換時に，ロイロット→ royllot などの翻訳が必要であり，期待しない語彙の生成要因であると考えられる

```
# 入力: ロイロット博士が、化粧着を着ていた ->

# 正解:
kdn:scene_x a kgc:situation ;
    kgc:hasproperty kdp:wear ;
    kgc:subject kdn:roylott ;
    kgc:time "1883-04-02t04:00:00"^^xsd:datetime ;
    kgc:what kdn:decorative_wear ;
    kgc:when kdn:1883-04-02t04 .

# 生成された trurle:
kdn:scene_x a kgc:situation ;
    kgc:haspredicate kdp:wear ;
    kgc:subject kdn:doctor_ross ;
    kgc:what kdn:dress_of_the_beauty .
```

エラー例

- hasPredicate と hasProperty が置き換わってしまう
  - 「一時的な性質は，hasPredicate で表現」という推論チャレンジKGの特殊な条件がうまく学習されなかった可能性がある

#入力分："ヘレンは怖がっている ->"

```
# 正解：
kdn:scene_x
  kgc:haspredicate kdp:bescared ;
  kgc:subject kdn:helen ;
  rdf:type kgc:situation ;
  ...

# 生成：
kdn:scene_x
  kgc:hasproperty kdp:scared ;
  kgc:subject kdn:helen ;
  rdf:type kgc:situation .
```

エラー例

- 入力に十分な情報がない・誤った情報を生成
  - 以下の例では，時間や情報発信者の情報がなく，それらに関する triples を正しく生成するのは困難．誤った情報を生成してしまっている．

```
# 入力："ロイロット博士は今日の日中は留守にしている。 ->",
```

```
# 正解：
kdn:scene_x
  kgc:haspredicate kdp:notexist ;
  kgc:infosource kdn:helen ;
  kgc:subject kdn:roylott ;
  kgc:time "1883-04-01t14:00:00";
  kgc:when kdn:1883-04-01t14:00 ;
  kgc:when kdn:today ;
  kgc:where kdn:mansion_of_roylott ;
  rdf:type kgc:statement .
```

```
# 生成：
kdn:scene_x
  kgc:haspredicate kdp:absent ;
  kgc:subject kdn:doctor_cottage ;
  kgc:time "1898-08-13t12:00:00";
  kgc:when kdn:1898-08-13t12:00 ;
  kgc:when kdn:today ;
  rdf:type kgc:situation .
```

タスク提案

タスク提案 1

- 生成された語彙をオリジナル KG にある語彙にマッピング

```
# 入力: ロイロット博士が、化粧着を着ていた ->
# 生成された trurle:
kdn:scene_x a kgc:situation ;
    kgc:haspredicate kdp:wear ;
    kgc:subject kdn:doctor_ross <- kdn:roylott で置き換え ;
    kgc:what kdn:dress_of_the_beauty .
```

タスク提案２

- 生成されたシーン間の関係を補完

```
kdn:scene_x
  kgc:haspredicate kdp:gain ;
  kgc:subject kdn:roylott ;
  kgc:what kdn:money ;
  kgc:time "1875-04-01t10:00:00" ;
  kgc:when kdn:1875-04-01t10 ;
  kgc:if kdn:scene_y ;
  rdf:type kgc:situation .
```

```
kdn:scene_y
  kgc:haspredicate kdp:live ;
  kgc:subject kdn:julia ;
  kgc:where kdn:mansion_of_roylott ;
  rdf:type kgc:situation .
```

タスク提案３

- メタ情報を付与

オリジナルの文：

「**折よく父も大事な用でロンドンへ出ると申しておりました。日中は留守にしているでしょうから、大丈夫でございましょう。**家政婦がひとりおりますが、耄碌しているお婆さんですから、外へ出てもらうのは訳ないかと。」
（青空文庫『まだらのひも』）

```
# 入力："ロイロット博士は今日の日中は留守にしている。 ->",

# 正解：
kdn:scene_x
  kgc:haspredicate kdp:notexist ;
  kgc:infosource kdn:helen ;
  kgc:subject kdn:roylott ;
  kgc:time "1883-04-01t14:00:00";
  kgc:when kdn:1883-04-01t14:00 ;
  kgc:when kdn:today ;
  kgc:where kdn:mansion_of_roylott ;
  rdf:type kgc:statement .
```

シーンに対応する小説内の自然言語にもメタ情報がない場合があり，かなり難しいタスクだと思われる

Thank you