# GPT系モデルの系列ラベリングによる品詞付与

安岡孝一*

## 1 はじめに

2019年5月15日リリースのUniversal Dependencies (UD) 2.4において、われわれの研究グループは、四書 (孟子・論語・大学・中庸) の依存構造コーパス (11176文、55026語、56768字) を製作[1]し、UDにおける開発の一翼を担うことにした[2]。開発は現在も継続しており、2024年5月15日リリースのUD 2.14は、『孟子』『論語』『禮記』『十八史略』『楚辭』『唐詩三百首』『摩訶般若波羅蜜大明呪經』『金剛般若波羅蜜經』『佛説阿彌陀經』『戰國策』の依存構造コーパス (89239文、433168語、450257字) を収録[3]している。これら依存構造コーパスの開発過程において、われわれは、UDとTransformers[4]による古典中国語 (漢文) 形態素解析・係り受け解析モデルを製作した。具体的には、古典中国語事前学習モデルRoBERTa-Classical-Chinese[5]に対して、系列ラベリングによるファインチューニングをおこない、形態素解析と係り受け解析を同時におこなう手法を開発[6]した。

ただし、この手法はBERT・RoBERTa・DeBERTaなど双方向モデル (bidirectional transformers) に特化して設計しており、GPT・LLaMA・Mistral・Qwenなど単方向モデル (unidirectional transformers) に適用可能かどうかは謎である。単方向モデルでは、順方向 (文頭から文末へ) の情報の流れはあるが、逆方向の流れはモデル内部には存在しない。この点が双方向モデルとは大きく異なっており、GPT系の単方向モデルは系列ラベリングにおいて不利となる可能性が高い。とは思うのだが、本当だろうか。ちょっとした工夫で、うまくいったりしないだろうか。

本稿では、GPT系モデルの系列ラベリングにおけるちょっとした工夫を、Universal Part-Of-Speech (UPOS)[7]品詞付与を題材に述べる。なお、本稿での各手法は、学際大規模情報基盤共同利用・共同研究拠点公募型共同研究『単語間に区切りのない書写言語における係り受け解析エンジンの開発』の成果である。

*京都大学人文科学研究所附属人文情報学創新センター

[1]Koichi Yasuoka: Universal Dependencies Treebank of the Four Books in Classical Chinese, DADH2019: 10th International Conference of Digital Archives and Digital Humanities (December 2019), pp.20-28.

[2]安岡孝一, ウィッテルン クリスティアン, 守岡知彦, 池田巧, 山崎直樹, 二階堂善弘, 鈴木慎吾, 師茂樹, 藤田一乘: 古典中国語 (漢文) Universal Dependencies とその応用, 情報処理学会論文誌, Vol.63, No.2 (2022年2月), pp.355-363.

[3]https://github.com/UniversalDependencies/UD_Classical_Chinese-Kyoto

[4]Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, Alexander M. Rush: Transformers: State-of-the-Art Natural Language Processing, Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (October 2020): System Demonstrations, pp.38-45.

[5]https://huggingface.co/KoichiYasuoka/roberta-classical-chinese-base-char

[6]Koichi Yasuoka: Sequence-Labeling RoBERTa Model for Dependency-Parsing in Classical Chinese and Its Application to Vietnamese and Thai, ICBIR 2023: 8th International Conference on Business and Industrial Research (May 2023), pp.169-173.

[7]https://universaldependencies.org/u/pos/

## 2　古典中国語 Xunzi モデルの系列ラベリング

XunziALLM (荀子系列大語言模型)[8]は、南京農業大学・南京理工大学・南京師範大学が共同で製作した[9]古典中国語の生成モデルで、2023 年 12 月 23 日に Xunzi-Qwen-7B と Xunzi-Qwen-7B-CHAT がリリースされた。その後、2024 年 1 月 13 日に Xunzi-GLM-6B と Xunzi-Baichuan-7B が、2024 年 5 月 6 日に Xunzi-Qwen1.5-4B と Xunzi-Qwen1.5-7B と Xunzi-Qwen1.5-14B と Xunzi-Qwen1.5-7B_chat が追加されている。

これら 8 つのモデルのうち、サイズが最も小さい Xunzi-Qwen1.5-4B に対し、UD 2.14 の古典中国語[3]UPOS で系列ラベリングに挑戦した。具体的には、mdx 上の Transformers 4.41.2 に Qwen2ForTokenClassification[10]を追加し、`lzh_kyoto-ud-train.conllu` で訓練したところ、NVIDIA A100-SXM4-40GB × 8 枚で 2 時間 17 分を要した。出来上がったモデルの系列ラベリング精度を、`lzh_kyoto-ud-test.conllu` の UPOS でテストしてみたところ、F1 値が 64.36 だった。正直、かなり低い。中身を見ると「四十而不惑」が

NUM　NUM　NOUN
四十　而不　惑

と品詞付与されていたりする。トークナイザが「而不」を 1 トークンにまとめている上に、情報の流れが順方向しかないので、「四十」の NUM に、後続の品詞が引きずられているようだ。さて、どうするか。

出来上がったモデルはそのままに、入力部と出力部をいじる方策を考えてみよう。入力部については、「而不」を「而」「不」の 2 トークンにバラすような形で、トークナイザの単文字化をおこなうことにする。端的には、モデルの `merges.txt` から「而 不」に対応する「èĢĮ ä¸į」[11]の行を削除すればよく、この手法を全漢字トークンに適用して単文字化をおこなう。出力部については、全てのラベルに対する logits[12]を出力した上で、逆方向 (文末から文頭へ) に Bellman-Ford[13]を適用 (図 1) して、logit の合計が最大となるようなラベルを選び出す。

これらの改良をおこなった結果、F1 値は 80.00 となった (表 1)。もう少し精度が上がってほしい気もするが、モデルそのものをいじってない以上、まあまあのセンだろう。ち

表 1: Xunzi-Qwen1.5-4B の系列ラベリング精度 (Precision / Recall / F1)

| | 逆方向の流れあり | なし |
|---|---|---|
| 単文字化あり | 79.10 / 80.91 / 80.00 | 72.68 / 67.80 / 70.16 |
| なし | 75.41 / 72.94 / 74.15 | 68.77 / 60.48 / 64.36 |

[8] `https://github.com/Xunzi-LLM-of-Chinese-classics/XunziALLM`

[9] Bin Li, Bolin Chang, Zhixing Xu, Minxuan Feng, Chao Xu, Weiguang Qu, Si Shen, Dongbo Wang: Overview of EvaHan2024: The First International Evaluation on Ancient Chinese Sentence Segmentation and Punctuation, Proceedings of the Third Workshop on Language Technologies for Historical and Ancient Languages (LT4HALA) @ LREC-COLING-2024 (May 2024), pp.229-236.

[10] `https://github.com/huggingface/transformers/pull/29878`

[11]「而」と「不」を UTF-8 のバイト列で表し、0x00〜0x20 を `U+0100`〜`U+0120` へ、0x21〜0x7e を `U+0021`〜`U+007E` へ、0x7f〜0x9f を `U+0121`〜`U+0141` へ、0xa0〜0xff を `U+00A0`〜`U+00FF` へそれぞれマッピングして、間に `U+0020` を挟み`<U+00E8 U+0122 U+012E U+0020 U+00E4 U+00B8 U+012F>`とした文字列。

[12]ラベルの生起確率を $p$ とおくとき、logit (対数オッド) は $\log \frac{p}{1-p}$ となる。

[13] Richard Bellman: On a Routing Problem, Quarterly of Applied Mathematics, Vol.16, No.1 (April 1958), pp.87-90.

```
from transformers import TokenClassificationPipeline
class BellmanFordTokenClassificationPipeline(TokenClassificationPipeline):
  def __init__(self,**kwargs):
    import numpy
    super().__init__(**kwargs)
    x=self.model.config.label2id
    y=[k for k in x if not k.startswith("I-")]
    self.transition=numpy.full((len(x),len(x)),numpy.nan)
    for k,v in x.items():
      if k.startswith("B-"):
        self.transition[v,x["I-"+k[2:]]]=0
      else:
        for j in [k]+y if k.startswith("I-") else y:
          self.transition[v,x[j]]=0
  def check_model_type(self,supported_models):
    pass
  def postprocess(self,model_outputs,**kwargs):
    import numpy
    if "logits" not in model_outputs:
      return self.postprocess(model_outputs[0],**kwargs)
    m=model_outputs["logits"][0].numpy()
    e=numpy.exp(m-numpy.max(m,axis=-1,keepdims=True))
    z=e/e.sum(axis=-1,keepdims=True)
    for i in range(m.shape[0]-1,0,-1):
      m[i-1]+=numpy.nanmax(m[i]+self.transition,axis=1)
    k=[numpy.nanargmax(m[0])]
    for i in range(1,m.shape[0]):
      k.append(numpy.nanargmax(m[i]+self.transition[k[-1]]))
    m=model_outputs["offset_mapping"][0].tolist()
    w=[{"entity":self.model.config.id2label[j],"start":s,"end":e,"score":z[i,j]}
      for i,((s,e),j) in enumerate(zip(m,k)) if s<e]
    if kwargs.get("aggregation_strategy") not in {"none",None}:
      for i,t in reversed(list(enumerate(w))):
        p=t.pop("entity")
        if p.startswith("I-"):
          w[i-1]["score"]=min(w[i-1]["score"],t["score"])
          w[i-1]["end"]=w.pop(i)["end"]
        elif p.startswith("B-"):
          t["entity_group"]=p[2:]
        else:
          t["entity_group"]=p
    for t in w:
      t["text"]=model_outputs["sentence"][t["start"]:t["end"]]
    return w
```

図 1: BellmanFordTokenClassificationPipeline

なみに、改良後の「四十而不惑」は


と品詞付与されている。「四」は B-NUM、「十」は I-NUM と品詞付与してほしいところだが、それにはモデルの再訓練が必要だと考えられる。

同様の手法で Xunzi-Qwen1.5-7B に対し、古典中国語 UPOS で系列ラベリングをおこなった。訓練は NVIDIA A100-SXM4-40GB × 8 枚で 6 時間 10 分を要した。入力部と出力部の改良も含めた結果を、表 2 に示す。悲しいかな 4B と 7B で、結果にあまり差がない。

表 2: Xunzi-Qwen1.5-7B の系列ラベリング精度 (Precision / Recall / F1)

| | 逆方向の流れあり | なし |
|---|---|---|
| 単文字化あり | 79.33 / 80.70 / 80.01 | 72.95 / 68.28 / 70.54 |
| なし | 75.46 / 72.59 / 74.00 | 69.02 / 60.76 / 64.63 |

なお、単文字化ありモデルを以下の URL で公開した。

`https://huggingface.co/KoichiYasuoka/Xunzi-Qwen1.5-4B-upos`
`https://huggingface.co/KoichiYasuoka/Xunzi-Qwen1.5-7B-upos`

Bellman-Ford の有無は、pipeline の`"upos"`タスクと`"token-classification"`タスクで切り替え可能である。また、単文字化なしにしたい場合は、以下の手順で `oldmerges.txt` を使えば、トークナイザを入れ替え可能である。ぜひ確かめてみてほしい。

```
from transformers import pipeline,AutoTokenizer
from transformers.utils import cached_file
task,model="upos","KoichiYasuoka/Xunzi-Qwen1.5-4B-upos"
nlp=pipeline(task,model,trust_remote_code=True,aggregation_strategy="simple")
m=cached_file(model,"oldmerges.txt")
nlp.tokenizer=AutoTokenizer.from_pretrained(model,merges_file=m)
print(nlp("四十而不惑"))
```

# 3 日本語 Swallow モデルの系列ラベリング

東京工業大学と産業技術総合研究所が共同で製作した Swallow-MS-7b-v0.1[14] に対し、MistralForTokenClassification (図 2) と `ja_gsd_modern.conllu`[15] の UPOS で、系列ラベリングに挑戦した[16]。出来上がったモデルの系列ラベリング精度を、UD 2.14 の国語研短単位[17] `ja_gsd-ud-test.conllu` の UPOS でテストしてみたところ、F1 値が 63.38 しかなかった。原因を探ってみたところ「どこの村でも当然だった」という文が


[14] `https://huggingface.co/tokyotech-llm/Swallow-MS-7b-v0.1`
[15] `https://github.com/KoichiYasuoka/SuPar-UniDic/blob/main/suparunidic/suparmodels`
[16] `https://qiita.com/KoichiYasuoka/items/c458e3b81a62ea0a2607`
[17] `https://github.com/UniversalDependencies/UD_Japanese-GSD`

表 3: 日本語 Swallow モデルの系列ラベリング精度 (Precision / Recall / F1)

(a) Swallow-MS-7b-v0.1

| | 逆方向の流れあり | なし |
|---|---|---|
| 単文字化あり | 76.99 / 77.61 / 77.30 | 59.18 / 59.13 / 59.15 |
| なし | 79.05 / 76.78 / 77.90 | 68.05 / 59.31 / 63.38 |

(b) Swallow-7b-plus-hf

| | 逆方向の流れあり | なし |
|---|---|---|
| 単文字化あり | 76.04 / 77.52 / 76.78 | 58.45 / 58.45 / 58.45 |
| なし | 79.05 / 76.78 / 77.90 | 68.05 / 59.31 / 63.38 |

(c) Llama-3-Swallow-8B-v0.1

| | 逆方向の流れあり | なし |
|---|---|---|
| 単文字化あり | 57.85 / 50.25 / 53.78 | 40.98 / 41.22 / 41.10 |
| なし | 58.39 / 44.08 / 50.24 | 40.93 / 36.60 / 38.64 |

と品詞付与されてしまっていて、非常にマズイ。「どこ」「の」「村」「で」「も」とトークナイズすべきところを、どういう風の吹き回しか「ど」「この」「村」「でも」とトークナイズしてしまうようだ。

そこで、トークナイザの単文字化を、ひらがなに対しておこなってみた。出力部の logits には、逆方向に Bellman-Ford を適用してみた。しかし、結果は

SCONJ B-DET I-DET NOUN ADP B-NOUN I-NOUN B-AUX I-AUX AUX
ど こ の 村 で も 当然 だ っ た

となってしまい、どうも単文字化はダメらしい (表 3(a))。また、Swallow-7b-plus-hf[18]や Llama-3-Swallow-8B-v0.1[19]に対し、LlamaForTokenClassification[20]で同様の手法を適用してみたが、あまり良い結果は得られなかった (表 3(b)(c))。なお、単文字化ありモデルを

```
https://huggingface.co/KoichiYasuoka/Swallow-MS-7b-char-upos
https://huggingface.co/KoichiYasuoka/Swallow-7b-plus-char-upos
https://huggingface.co/KoichiYasuoka/Llama-3-Swallow-8B-char-upos
```

で、単文字化なしモデルを

```
https://huggingface.co/KoichiYasuoka/Swallow-MS-7b-upos
https://huggingface.co/KoichiYasuoka/Swallow-7b-plus-upos
https://huggingface.co/KoichiYasuoka/Llama-3-Swallow-8B-upos
```

で公開した。いずれも pipeline の"upos"タスクと"token-classification"タスクで、Bellman-Ford の有無を切り替え可能である。ぜひ確かめてみてほしい。

[18]https://huggingface.co/tokyotech-llm/Swallow-7b-plus-hf
[19]https://huggingface.co/tokyotech-llm/Llama-3-Swallow-8B-v0.1
[20]https://github.com/huggingface/transformers/issues/26521

```
from transformers import MistralModel,MistralPreTrainedModel
from transformers.modeling_outputs import TokenClassifierOutput
class MistralForTokenClassification(MistralPreTrainedModel):
  def __init__(self,config):
    from torch import nn
    super().__init__(config)
    self.num_labels=config.num_labels
    self.model=MistralModel(config)
    if getattr(config,"classifier_dropout",None) is not None:
      classifier_dropout=config.classifier_dropout
    elif getattr(config,"hidden_dropout",None) is not None:
      classifier_dropout=config.hidden_dropout
    else:
      classifier_dropout=0.1
    self.dropout=nn.Dropout(classifier_dropout)
    self.classifier=nn.Linear(config.hidden_size,config.num_labels)
    self.post_init()
  def get_input_embeddings(self):
    return self.model.embed_tokens
  def set_input_embeddings(self,value):
    self.model.embed_tokens=value
  def forward(self,input_ids=None,past_key_values=None,attention_mask=None,
    position_ids=None,inputs_embeds=None,labels=None,use_cache=None,
    output_attentions=None,output_hidden_states=None,return_dict=None):
    if return_dict is None:
      return_dict=self.config.use_return_dict
    transformer_outputs=self.model(input_ids,past_key_values=past_key_values,
      attention_mask=attention_mask,position_ids=position_ids,
      inputs_embeds=inputs_embeds,use_cache=use_cache,
      output_attentions=output_attentions,
      output_hidden_states=output_hidden_states,return_dict=return_dict)
    hidden_states=transformer_outputs[0]
    hidden_states=self.dropout(hidden_states)
    logits=self.classifier(hidden_states)
    loss=None
    if labels is not None:
      from torch import nn
      loss_fct=nn.CrossEntropyLoss()
      loss=loss_fct(logits.view(-1,self.num_labels),labels.view(-1))
    if not return_dict:
      output=(logits,)+transformer_outputs[2:]
      return ((loss,)+output) if loss is not None else output
    return TokenClassifierOutput(loss=loss,logits=logits,
      hidden_states=transformer_outputs.hidden_states,
      attentions=transformer_outputs.attentions)
```

図 2: MistralForTokenClassification

# 4　日本語単文字GPT2モデルの系列ラベリング

ならば、ku-nlp/gpt2-{small,medium,large}-japanese-char[21]のように、トークナイザが単文字で設計されている日本語モデルはどうだろう。GPT2ForTokenClassification[22]で同様の手法を適用してみた (表 4)。逆方向の Bellman-Ford が効いており、全体に F1 値が高い。

表 4: ku-nlp/gpt2-∗-japanese-char の系列ラベリング精度 (Precision / Recall / F1)

| | 逆方向の流れあり | なし |
|---|---|---|
| small (90M) | 92.33 / 92.31 / 92.32 | 76.27 / 75.60 / 75.93 |
| medium (310M) | 92.92 / 92.99 / 92.95 | 79.27 / 77.32 / 78.28 |
| large (717M) | 93.52 / 93.59 / 93.55 | 81.07 / 78.45 / 79.74 |

small モデルで「どこの村でも当然だった」に品詞付与すると、逆方向の流れなしでは

| B-PRON | I-PRON | ADP | NOUN | ADP | ADP | B-NOUN | I-ADJ | AUX | I-AUX | AUX |
|---|---|---|---|---|---|---|---|---|---|---|
| ど | こ | の | 村 | で | も | 当 | 然 | だ | っ | た |

逆方向の流れありでは

| B-PRON | I-PRON | ADP | NOUN | ADP | ADP | B-NOUN | I-NOUN | B-AUX | I-AUX | AUX |
|---|---|---|---|---|---|---|---|---|---|---|
| ど | こ | の | 村 | で | も | 当 | 然 | だ | っ | た |

となった。単文字モデルを用いた品詞付与では、B-・I-間で矛盾が起こりやすいのだが、順方向の情報の流れだけでは矛盾の解消は難しく、逆方向の流れが必要になるということだろう。なお、以下の URL で各モデルを公開した。

```
https://huggingface.co/KoichiYasuoka/gpt2-small-japanese-upos
https://huggingface.co/KoichiYasuoka/gpt2-medium-japanese-upos
https://huggingface.co/KoichiYasuoka/gpt2-large-japanese-upos
```

Bellman-Ford の有無は、pipeline の`"upos"`タスクと`"token-classification"`タスクで切り替え可能である。ぜひ使ってみてほしい。

# 5　おわりに

Qwen2・Mistral・LLaMA・GPT2 などの単方向モデルに対し、系列ラベリングによるファインチューニングをおこない、UPOS 品詞付与をおこなうモデルを開発した。いずれのモデルにおいても、出力部に逆方向 (文末から文頭へ) の情報の流れを入れることで、品詞付与の精度が向上することが明らかになった。一方、トークナイザを単文字化した場合の効果は、モデルごとに異なっているようである。

GPT 系モデルでの品詞付与に関しては、おおよその目途が立ったことから、さて、次の課題は係り受け解析である。ただ、モデルが巨大すぎて、Biaffine[23]の隣接行列が、なかなか

[21]村脇有吾: 文字言語モデルからの単語言語モデルの教師なし合成, 情報処理学会研究報告, Vol.2024-NL-260, No.2 (2024 年 6 月 28 日), pp.1-14.

[22]`https://github.com/huggingface/transformers/pull/13290`

[23]Timothy Dozat, Christopher D. Manning: Deep Biaffine Attention for Neural Dependency Parsing, 5th International Conference on Learning Representations (April 2017), C25.

GPU に乗り切らない。何とか日本語単文字 GPT2 モデル (逆方向の流れあり) を、esupar[24] と国語研短単位 UD[17] でチューニングしてみたところ、UAS (Unlabeled Attachment Score)・LAS (Labeled Attachment Score)・MLAS (Morphology-aware Labeled Attachment Score)[25] ともに、まずまず良い値となった (表 5)。

今後、他の GPT 系モデルによる係り受け解析も、頑張ってモノにしていきたい。

表 5: gpt2-∗-japanese-upos の係り受け解析精度 (Precision / Recall / F1)

| | UAS | LAS | MLAS |
|---|---|---|---|
| small | 93.95 / 93.93 / 93.94 | 93.13 / 93.11 / 93.12 | 78.00 / 78.11 / 78.05 |
| medium | 94.12 / 94.19 / 94.16 | 93.15 / 93.23 / 93.19 | 78.80 / 78.99 / 78.90 |
| large | 94.91 / 94.97 / 94.94 | 94.05 / 94.12 / 94.08 | 80.74 / 80.92 / 80.83 |

[24] https://github.com/KoichiYasuoka/esupar

[25] Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov: CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, Proceedings of the CoNLL 2018 Shared Task (October 2018), pp.1-21.