目次

ファイルの構成と実行手順

ファイル構成：

submit.zip: 予測モデルのソースコードと学習済モデル

ner-submit-final-nishika.ipynb: 必要な環境と実行手順に関するJupyter Notebook

実行環境（Colab)：

OS：Ubuntu

GPU：16GB（これより少ない場合はStackingのデータを読み込めない可能性がある）

メモリ：32GB

実行手順：

ner-submit-final-nishika.ipynbをColabの環境で順次実行すれば良い。

ソリューションの概要

データの増強

今回の課題の本質はNERタスクのため、NERタスクでよく使う置換の方法を使った。

収集したデータ：

- 人名18,678件
  Wikipedia[1]から実在する人物の名前を3,678件収集し、さらに疑似個人情報生成のサイト[2]から人名を15,000件生成した。
- 企業名3,030件
  インタネット[3]から企業名を3,003件収集した。

置換方法：

ラベルがつけられている部分を同じラベルのテキストに置換する。人名と企業名のみ対象とした。

例えば：　収集した人名：　田中太郎

原文　→　被告榎本酉蔵らの共同不法行為

置換後　→　被告田中太郎らの共同不法行為

※　同じ人物が一文書に連続出現する場合については、同じ人名で置換したほうがより元データに近い形の疑似データができる。しかし、実装としては違う人名で置換した。それは実装の手間と人名のバラエティを考慮したためである。

作成したデータ：元データの2回分の疑似データを作成した

1. https://ja.wikipedia.org/wiki/%E6%9D%B1%E4%BA%AC%E5%A4%A7%E5%AD%A6%E3%81%AE%E4%BA%BA%E7%89%A9%E4%B8%80%E8%A6%A7
2. https://hogehoge.tk/personal/generator/
3. https://chosa.itmedia.co.jp/providers

学習データ作成　その１　Tokenization

データ作成ステップ

1. Ginza Tokenization: 入力文をまずGinzaで形態素解析を行う。
2. Bert Tokenization: Ginzaで得た各Tokenについて更にBert Tokenizerで形態素解析を行う。
3. ラベルマスク: Bert Toenizerで得た最初のTokenのラベルのみ保留、その他のラベルをマスクする。（マスクの部分についてLossを計算する際に考慮しないために、コードを-100にする）

原文	被告田中太郎らの共同不法行為						
Ginza Token	被告	田中太郎	ら	の	共同	不法行為	
Ginza Tag	O	B-PERSON	O	O	O	O	
Bert Token	被告	田中	太郎	ら	の	共同	不法	行為
Bert Tag	O	B-PERSON	[MASK]	O	O	O	O	[MASK]

学習データ作成　その２　長文の切断

Bertモデルが扱えるToken数は上限512までのため、一つの判例文を**512**Tokens以下のいつくかの部分に分解する必要がある。

また、Ginza Tokenの途中から切断されることを防ぐために、判例文をまず読点「。」で文に切ってから、512Tokens以下の複数の部分に切断した。

例：

原文：　。。。両手挫傷の傷害を負わせた。被告人松村好利は,常習として被害者に対し殴る暴行を加えた。

ここまで510 Tokens。文の途中で切断されることを防ぐため、一旦ここで切って、残りのTokensを[PAD]で埋める。

使用したモデル

合わせて6つのBert baseの学習済みのモデルを使用した。

- Hugging Faceが提供した4つのモデル
  - cl-tohoku/bert-base-Japanese
  - cl-tohoku/bert-base-japanese-whole-word-masking
  - cl-tohoku/bert-base-japanese-char
  - cl-tohoku/bert-base-japanese-char-whole-word-masking
- 情報通信研究機構が公開した2つのモデル
  - NICT_BERT-base_JapaneseWikipedia_100K
  - NICT_BERT-base_JapaneseWikipedia_32K_BPE

モデルアーキテクチャ：

学習済みのBertの最後２層を連結して線形変換して最終結果を得た。

学習の詳細

ハイパーパラメータ：

学習にHugging FaceのTransformersのTrainerを利用した。以下明記したこと以外のすべてのハイパーパラメータはデフォルト設定となる。

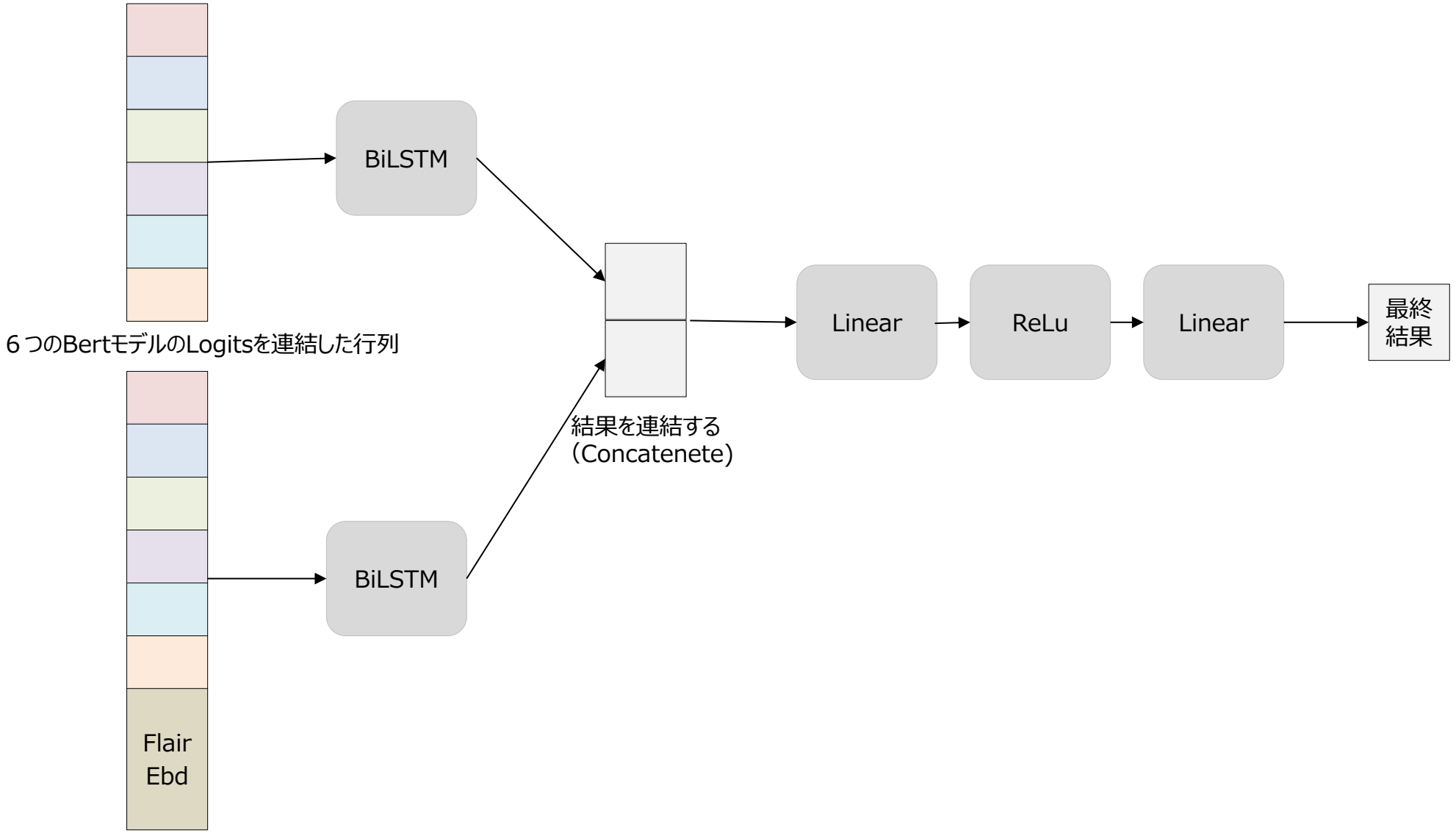
- Training Data：元データと増強した２つのデータ（合わせてモデルの３倍の量）
- Epoch： 1
- Learning Rate：3e-5
- CV: 5 folds

損失の計算：

損失関数はpytorchのCrossEntropyLossを利用した。計算する際に、MASKのTokenを考慮しなかった。

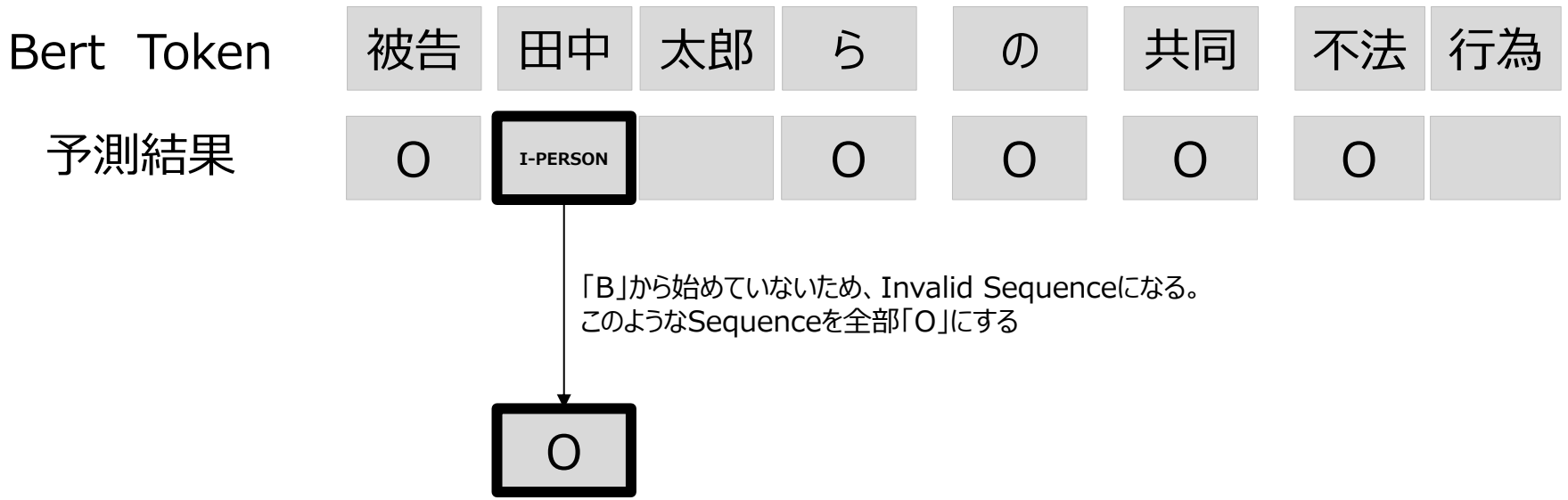
Stacking Model

Stackingモデルでは、**各判例ごと**、６つのBertモデルのLogitsとFlair Embeddingを連結した行列をインプットデータとした。
１判例１データのため、Stackingモデルに判例の全体像を把握させることが期待している。



後処理

ルールに則っていない「I－」から初めのシーケンスをすべて「O」にする。



結果比較

Stackingモデルについて2nd foldのLB結果が一番よかったため、最終的に2nd foldの結果を選んだ。実はPBから見るとやはり5 folds全部使って投票したほうが成績が良かった。

モデル	5 fold CV	Public Leadboard	Private Leadboard
NICT-100k	0.8885	0.9191	0.8719
NICT-32k	0.8854	0.9119	0.8543
cl-char	0.8635	0.8976	0.8399
cl-charwom	0.8760	0.8892	0.8364
cl-wom	0.8786	0.9029	0.8323
cl	0.8819	0.9029	0.8636
6 models simply vote	0.8916	0.9197	0.8581
Stacking (2nd fold)		0.9327	0.8697
Stacking (2nd fold) + post processing		0.9463	0.8841
Stacking (5 folds simpley volt) + post processing		0.9404	0.890384

最終に選択した結果

※Micro F1を評価基準になっているため、長文に同じエンティティが複数回出現する場合、それを検出できなかったら最終スコアに大きな影響がある。それは結果が不安定の原因だと思う。