2022 년 k-ium 의료인공지능경진대회 기술 문서

팀명 : 박천복(개인)

◇ 요약

1. 1 차 데이터셋의 주제 Column 중 'Conclusion₩n'에서 불필요한 문자(₩n) 수정.

- 프로그램 수행 중에 오류가 발생하지 않도록 ['Findings', 'Conclusion', 'AcuteInfarction']으로 Column 이름을 변환 후 데이터 처리를 진행합니다.

2. 데이터셋 내에 존재하는 결측값(NaN)을 공백문자('')로 처리(변환).

3. 학습 내용에 필요하지 않은 문자와 구조를 삭제.

- 항목 번호: 1), 2), 1], 2] 등
- 특수 문자와 이스케이프 문자: '-', '<>', '()', ':', '[]', '{}', '&', '₩r(개행문자)', '₩n(줄바꿈)' 등

4. 대표적인 자연어 처리 모델인 BERT 의 사전학습 모델에 적용하기 위한 데이터 전처리.

- BERT 의 사전학습 모델에 적용하기 위해 원본 데이터 구조를 변경합니다.
- 문장 데이터를 토큰화, 토큰 데이터에 대한 고유 식별번호 생성, 문자 존재 여부를 판별하는 Attention Mask 생성의 작업을 수행합니다.

5. 전처리한 데이터를 사전모델에 추가 학습시키기 위해 Pytorch 라이브러리 사용.

- 전처리한 DataFrame 을 Pytorch Tensor 로 변환 후, DataLoader 를 구성합니다.
- device(CPU/GPU) 설정 후, DataLoader 의 각 배치를 device 에 넣어 학습을 수행합니다.

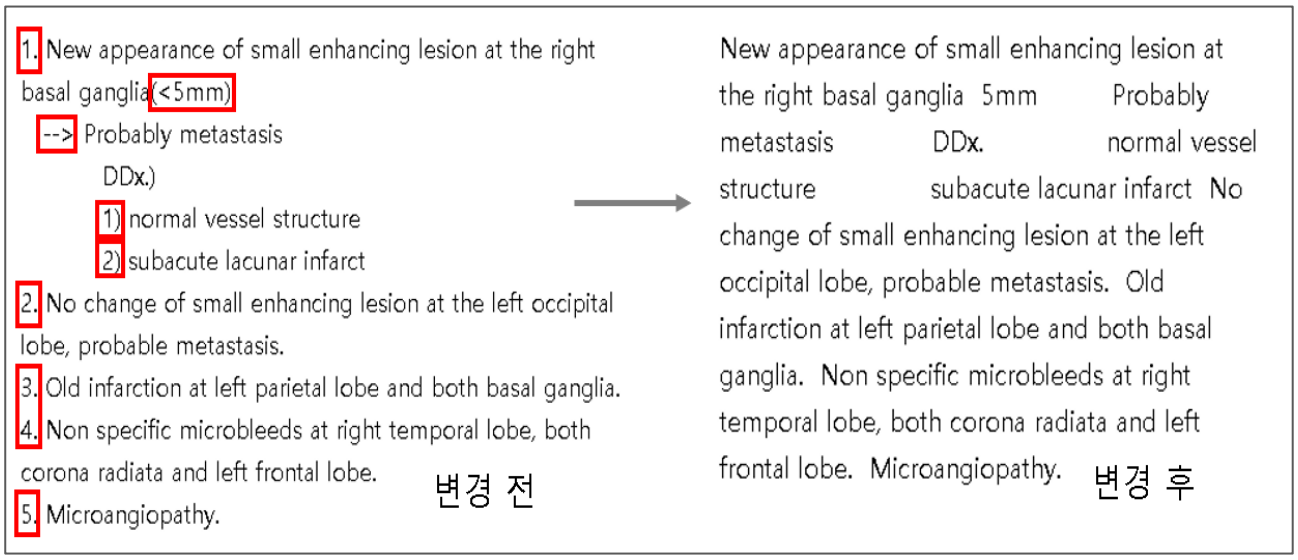
6. 검증 데이터를 이용해 검증 과정을 수행하고 정확도 계산.

- 검증에 사용할 데이터도 학습 모델이 적절히 판독할 수 있도록 Pytorch DataLoader 구조로 데이터를 변환해야 합니다. 그 과정은 위에서 DataLoader 를 만드는 것과 같습니다.
- 검증을 위한 device(CPU/GPU)를 선택하고, 학습 모델을 Pytorch 모델로 가져옵니다.
- 뇌졸중이 아닌 값=0, 뇌졸중인 값=1 일 때, 검증 과정에서 [1 로 예측한 값, 0 으로 예측한 값]이 결과로 출력됩니다. Softmax 함수를 사용해 출력 값을 [0,1] 범위로 잡았습니다.

◇ 구현 과정 (방법)

★ 자연어 데이터가 어떻게 처리되는지 예시 캡처 이미지를 보여주면서 추가 설명했습니다.

1. 문장 구조에서 학습에 불필요한 요소들을 제거합니다.



2. ['Findings', 'Conclusion'] Column 의 문장을 합치고, 사전학습 모델인 BERT 의 구조에 맞게 변환합니다. BERT 는 특수목적의 토큰 중 처음 문장의 시작을 의미하는 '**[CLS]**', 문장의 종료를 의미하는 '**[SEP]**'가 있습니다. BERT 에서 문장 인식을 할 때 필수로 참고하는 요소입니다.

   아래와 같이 변경된 데이터가 이후의 처리 과정에서 하나의 '**행 입력**'이 됩니다.

[CLS] Clinical information Lung cancer patient. **[SEP]** Axial T1WI, sagittal T1WI, axial T2WI, axial FLAIR, axial T2* GRE image 획득하였으며 조영증강을 시행함. **[SEP]** Right basal ganglia에 5mm 미만의 새롭게 enhancing되는 병변이 있고 probably metastasis로 생각됨. **[SEP]** 그러나 DDx.로 normal vessel structure와 subacute lacunar infarct을 고려해야 함. **[SEP]** 그 외 이전과 큰 변화 없음. **[SEP]** New appearance of small enhancing lesion at the right basal ganglia 5mm Probably metastasis DDx. **[SEP]** normal vessel structure subacute lacunar infarct No change of small enhancing lesion at the left occipital lobe, probable metastasis. **[SEP]** Old infarction at left parietal lobe and both basal ganglia. **[SEP]** Non specific microbleeds at right temporal lobe, both corona radiata and left frontal lobe. **[SEP]** Microangiopathy. **[SEP]**

3. BERT 의 사전학습 모델을 기반으로 문장 데이터를 토큰화합니다.

BERT 의 사전학습 모델에서 등록된 단어를 찾아냅니다. 만약, 사전에 포함된 단어가 아니라면 사전에 포함된 단어까지 인식한 다음, 뒤에 추가로 붙는 단어를 '##'으로 구분해 이해합니다.

예시로, 'Lung' 단어가 사전에 없는 경우, 사전에 있는 Lu 까지 'Lu' 토큰으로 구분하고, 이후의 단어 중 사전에 있는 ng 를 '##ng'토큰으로 구분해 서로 이어진 단어라고 이해합니다.

```
['[CLS]', 'Clinical', 'information', 'Lu', '##ng', 'cancer',
'patient', '.', '[SEP]', 'A', '##xia', '##l', 'T1', '##W', '##I', ',',
'sa', '##gitt', '##al', 'T1', '##W', '##I', ',', 'a', '##xia', '##l',
'T2', '##W', '##I', ',', 'a', '##xia', '##l', 'FL', '##A', '##IR', ',',
'a', '##xia', '##l', 'T2', '*', 'GR', '##E', 'image', '획', '##득',
'##하였으며', '조', '##영', '##증', '##강', '##을', '시', '##행',
'##함', '.', '[SEP]', 'Right', 'basal', 'gang', '##lia', '##에', '5',
'##mm', '미', '##만', '##의', '새', '##롭', '##게', 'en', '##han',
'##cing', '##되는', '병', '##변', '##이', '있고', 'probably',
```

처리의 효율을 높이기 위해서 각 토큰을 고유한 번호로 Mapping 합니다. 단어가 같은 토큰은 모두 같은 번호를 가지고, 단어가 다른 토큰은 서로 다른 번호를 가집니다. BERT 에서 사용하는 특수한 토큰 중 **[CLS]**는 **101**, **[SEP]**는 **102** 의 고정된 번호를 가집니다.

앞에서 채워지는 0 의 값은 Padding 입니다. 학습을 위해 사용할 행 입력의 사이즈를 512 로 잡았습니다. 그래서 한 행을 구성하는 토큰의 개수가 512 를 넘지 않으면 남은 공간을 0 으로 채웁니다.

Padding

```
[    0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0   101   140 11281   117 11649 12396   119   102
   138 30967 10161 94459 13034 11281   117 10148 78108 10415
 94459 13034 11281   117   169 30967 10161 102172 13034 11281
   117   169 30967 10161 83243 10738 73522   117   169 30967
 10161 102172   115 58787 11259 18170   117   169 30967 10161
   141 13034 11281   117 69617 40333 74116 58573 11565 52157
 10738  9999 118813 53529 63938 52157 10738 10530 33378  9678
 30858 119230 47181 10622  9485 25549 48533   119   102 40586
   117 42818 11565 16199   117 10111 16382 79441   138 103411
 10114 30156 103411 11649 12396 10160 10105 45684 24289 23178
 45846 10406 16216   119   102 11982 17437 73995 10161 23773
```

생성된 토큰에서 실제 토큰이 존재하는 구간과 패딩으로 채워진 구간이 존재합니다. 이를 BERT 에 명확하게 전달하기 위해 Attention Mask 를 사용합니다. 이를 통해 불필요한 토큰의 연산을 최소화할 수 있습니다. 패딩 구간은 0 으로, 단어 토큰이 있는 구간은 1 로 구성합니다.

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

4. [학습, 검증]에 사용할 Pytorch DataLoader 구조로 변환합니다. 위의 과정까지는 Pandas 의 DataFrame 구조입니다. DataFrame 을 학습에 적용하기 위한 구조인 Pytorch Tensor 로 변환합니다. Tensor 데이터를 [학습, 검증]에 사용하기 위해 세트로 구성한 데이터 묶음이 DataLoader 입니다. 아래는 DataLoader 의 하나의 batch(set) 데이터입니다.

   1 번 tensor: 행 입력의 단어 토큰

   2 번 tensor: 1 번 tensor 에 대응하는 Attention mask

   3 번 tensor: 정답지(Label)

```
torch.Size([512])
tensor([[    0,     0,     0,  ..., 13890,   119,   102],
        [    0,     0,     0,  ..., 10157,   119,   102],
        [    0,     0,     0,  ..., 29731, 17530,   102],
        ...,
        [    0,     0,     0,  ..., 15851,   119,   102],
        [    0,     0,     0,  ..., 10251,   119,   102],
        [    0,     0,     0,  ..., 43977,   119,   102]], dtype=torch.int32)
tensor([[0, 0, 0,  ..., 1, 1, 1],
        [0, 0, 0,  ..., 1, 1, 1],
        [0, 0, 0,  ..., 1, 1, 1],
        ...,
        [0, 0, 0,  ..., 1, 1, 1],
        [0, 0, 0,  ..., 1, 1, 1],
        [0, 0, 0,  ..., 1, 1, 1]])
tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        1, 0, 0, 0, 0, 0, 0, 0])
```

DataLoader 를 이용해 학습과 검증하는 과정에서 각 문장(CLS, SEP 로 구분)을 구성하는 토큰의 연관성, 인접 토큰의 영향도 등을 판단합니다.

◇ 검증 결과 (AUROC)

```
정확도(Accuracy): 0.97121
```