Project: Toxicity Detection 보고서

20213073 AI융합학부 반세현

File 구성

소스코드	모델	inference
Nlp_mlp_100.ipynb	Model1.pth	Inference_ver1.csv
Nlp_mlp_300.ipynb	Model2.pth	Inference_ver2.csv
Nlp_mlp_300_mean.ipynb	Model3.pth	Inference_ver3.csv

최종 제출본

1. 특수 문자 없애기

- Comment를 소문자로 바꿔주고 특수 문자를 제외해준다.

```
def preprocess_text(text):
    text = ' '.join(word.lower() for word in text)
    text = re.sub(r"([.,!?])", r" #1 ", text)
    text = re.sub(r"[^a-zA-Z.,!?]+", r" ", text)
    return text

train_tokenize['comment'] = train_tokenize['comment'].apply(preprocess_text)
inference_tokenize['comment'] = inference_tokenize['comment'].apply(preprocess_text)
```

2. embedding

- Stanford/glove에서 제공되는 사전학습된 embedding을 사용했다.
- 해당 glove은 50, 100, 200, 300차원으로 구성되어 embedding의 차원을 선택하여 사용할 수 있다.

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip
!mkdir -p data/glove
!mv glove.6B.100d.txt data/glove
```

2. embedding

- Embedding dim 100보다 embedding dim 300에서 성능향상을 확인.

```
검증 손실: 0.36875443259078683
검증 정확도: 94.30408435919449
검증 f1-score: 0.9346551732332808
```

Embedding 100

- 처음에는 실수로 추출된 embedding 을 전부 합쳐줬다.
- Mean으로 바꿔주었으나 성능에서 큰 차이를 보이지 않았다.

- 의문점 : 수업시간에 입력 크기를 일관되게 만들어주면서 정보의 손실이 발생하지 않도록 Embedding의 max나 mean을 쓴다는 것을 배웠다. 그러나 문장의 길이에 영향을 받기 때문에 sum은 사용할 수 없다고 알고 있다.

```
train_vector.append(sum(embedding_vetor)/count)
```

```
검증 손실: 0.3610796568866724
검증 정확도: 94.96791201450522
검증 f1-score: 0.9454714549553819
```

Embedding 300, sum

Embedding 300, mean

```
train_vector[i] = np.sum(embedding_vetor, axis=0, keepdims=True)
```

```
검증 손실: 0.361930893513865
검증 정확도: 94.98669958965547
검증 f1-score: 0.9451090380334451
```

2. Embedding *(의문 : 왜 SUM와 MEAN의 embedding의 성능이 비슷하게 나오는지*

```
df_mean[df_mean['pred']==1].shape
```

```
(8311, 2)
```

Mean을 쓸 경우 toxicity가 8311개

```
평균 단어 개수 :  64.78582601371676
```

mean을 쓸 경우 평균 단어 64개

```
df_sum[df_sum['pred']==1].shape
```

```
(2277, 2)
```

sum을 쓸 경우 toxicity가 2277개

```
평균 단어 개수 :  48.026350461133076
```

sum을 쓸 경우 평균 단어 48개

```
sub_set = [x for x in mean_idx if x not in sum_idx]
```

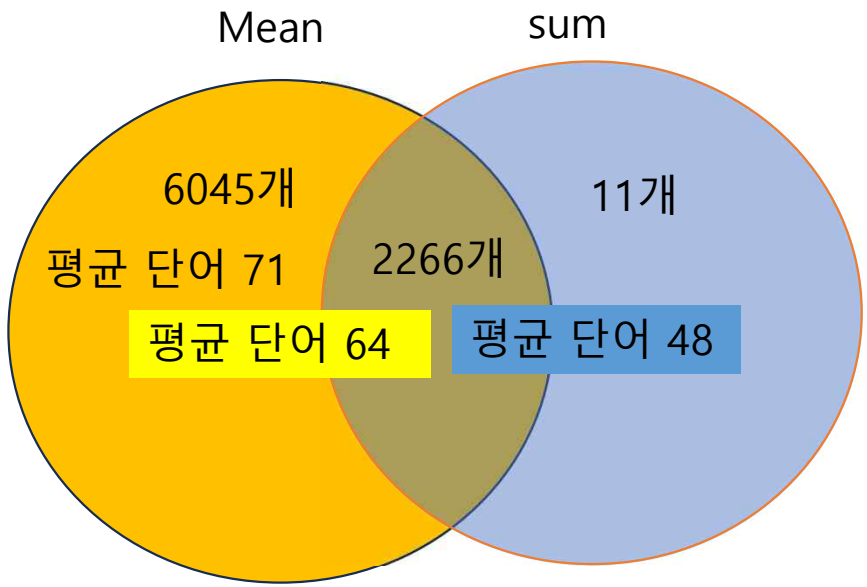
```
len(sub_set)
```

```
6045
```

두 index의 차집합은 6045개

```
평균 단어 개수 :  71.0582299421009
```

차집합의 평균 단어 개수 71개



결론 : Mean embedding을 사용하면 문장이 긴 toxicity 글을 탐지한다.

* 2266+6045 = 8311

2. Embedding *(의문 : 왜 SUM와 MEAN의 embedding의 성능이 비슷하게 나오는지*

추측 1. toxicity가 1인 comment의 경우 글의 길이가 짧은 경향이 있기 때문에
긴 글을 잡아내지 못해도 정확도가 비슷하게 나온다.

추측 2. 모델 성능이 좋지 않아 긴 글의 toxicity를 구별하지 못하고 있다.

추측 3. 문장의 길고 짧음도 유의미한 정보성을 지녔다.

```
n,_ = train_tokenize.shape
len_0 = 0
len_1 = 0
count_0 = 0
count_1 = 0
for i in range(n):
  string = train_tokenize.iloc[i,0].split(" ")
  if train_tokenize.iloc[i,1]==1:
    count_1+=1
    len_1+=len(string)
  else:
    count_0+=1
    len_0+=len(string)
```

```
print("0의 평균 단어 개수 : ", len_0/count_0)
print("1의 평균 단어 개수 : ", len_1/count_1)
```

```
0의 평균 단어 개수 :  79.594721228756
1의 평균 단어 개수 :  64.41611089316072
```

최종 선택 근거

- Train data의 평균 단어 개수 경향과 비슷한 mean을 선택하기로 했다.
- 이론적으로 옳기 때문에

3. Data split

- Label간의 불균형이 발견돼 test, validation set을 비율에 맞도록 나눴다.
- Label 1의 비율이 0의 1/10이니 정확도가 90% 이상은 나와야 유의미한 학습을 한다고 볼 수 있다.

- 입력 형식은 (data 개수, embedding dim) shape의 데이터 프레임이다.

```
train_emb[train_emb["toxicity"]==1].shape
```

```
(7647, 101)
```

```
train_emb[train_emb["toxicity"]==0].shape
```

```
(72138, 101)
```

```
[ ]  X_train[X_train['toxicity']==1].shape
```

```
(6118, 101)
```

```
[ ]  X_train[X_train['toxicity']==0].shape
```

```
(57710, 101)
```

4. 학습 및 검증

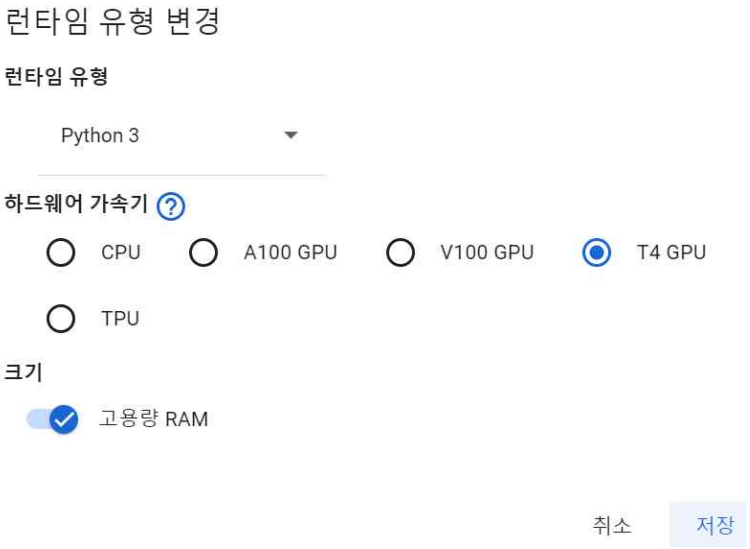
- 모델은 수업자료였던 mlp를 썼다.
- 고정된 input dim (embedding dim) 을 사용할 수 있다는 점에서 mlp를 선택했다. Mlp의 activation functio은 softmax로, output dim 각각의 dim에 속할 확률을 돌려줘 합산은 1이 된다.
- Output dim을 2로 두어 activation 결과 0의 확률이 크다면 toxicity=0, 1의 확률이 크다면 toxicity=1 로 분류하도록 inference 했다.

```
def __init__(self, input_dim, hidden_dim, output_dim):
    super(SurnameClassifier, self).__init__()
    self.fc1 = nn.Linear(input_dim,hidden_dim)
    self.fc2 = nn.Linear(hidden_dim,output_dim)

def forward(self, x_in, apply_softmax=True):
    intermediate = F.relu(self.fc1(x_in))
    prediction_vector = self.fc2(intermediate)

    if apply_softmax:
      prediction_vector = F.softmax(prediction_vector,dim=1)

    return prediction_vector
```

5. 기타

아쉬운 점

- 여러 모델과 비교해야 하는데 input으로 가공하는 과정이 어려워 mlp밖에 실험해보지 못했다. Lstm등 sequence로서 의미를 갖는 모델도 실험을 해봐야 한다.
- 첫번째 전처리가 소문자로 통일하는 일인데, 대문자/소문자에도 정보성이 담겨있다. 예를 들어 대문자로 통일된 문장은 공격성을 보인다. 대문자/소문자를 embed에 학습시킬 방법이 있는지 알아보고 싶다.
- Embedding을 sum/mean뿐만 아니라 min/max를 사용하여 시도해봐야 한다.



실험 환경 : colab pro, 실습실은 사용하지 않았습니다.