<파일설명>

- 비교를 위해서 4가지 모델을 구현하고,각 모델에 대해 ipynb 파일과 학습된 모델 weight, 그리고 학습된 word embedding 파일을 첨부했다.
- 필수로 제출해야 하는 파일인 주피터 노트북과 학습된 임베딩 파일을 제외한 파일들은 '기타파일들' 이라는 폴더에 저장했다.

4가지 모델과 해당하는 ipynb 파일의 이름과 모델 weight는 아래와 같다.

1. Lstm with pretrained embedding : lstm_with_embedding.ipynb, lstm_with_w2v.pt
2. Lstm without pretrained embedding : lstm.ipynb, lstm_wo_w2v.pt
3. Cnn with pretrained embedding : MidTermAssignment_2019_17577_jihoojung.ipynb,cnn_with_embedding.ipynb(MidTermAssignment_2019_17577_jihoojung.ipynb과 cnn_with_embedding.ipynb는 같은 파일임),cnn_with_w2v.pt
4. Cnn without pretrained embedding : cnn.ipynb, cnn_wo_w2v.pt

핵심파일

- 주피터 노트북 : MidTermAssignment_2019_17577_jihoojung.ipynb, 4가지 모델 중 가장 설명을 상세하게 달아놓은 **Cnn with pretrained embedding** 모델에 대한 파일을 **MidTermAssignment_2019_17577_jihoojung.ipynb**로 저장해두었다.
- 학습된 임베딩 파일 : w2v.model

<개괄>

총 4가지 모델을 구현했다.이번 과제에서는 rnn계열(lstm)과 cnn의 결과를 비교해보는 것을 목표로 결정했다.

전체적인 모델과 train, evaluate 등의 함수들은 수업 자료를 활용했다. epoch은 공통적으로 10으로 사용하고 아래의 4모델의 결과를 비교해보기로 했다.

1. Lstm with pretrained embedding
2. Lstm without pretrained embedding
3. Cnn with pretrained embedding
4. Cnn without pretrained embedding

Why not RNN?

그 외에도 아래와 같은 RNN 모델을 정의해서 epoch 30으로 테스트해보았으나 아래와 같이 전혀 training이 되지 않는 것 같아 lstm과 cnn만을 사용하기로 했다. 물론, rnn을 좀 더 깊게 쌓고 정교하게 쌓았다면 결과가 달라질 것이겠지만,,, 이번 실험에서는 rnn계열(lstm)과 cnn의 결과를 비교해보는 것으로 결정했다,

```python
import torch.nn as nn


class RNN(nn.Module):
def __init__(self, input_dim, embedding_dim, hidden_dim, output_dim):
super().__init__()
self.embedding = nn.Embedding(input_dim, embedding_dim)
self.rnn = nn.RNN(embedding_dim, hidden_dim)
self.fc = nn.Linear(hidden_dim, output_dim)
def forward(self, text):


#text = [sent len, batch size]
# print("text in model",text)
embedded = self.embedding(text)
#embedded = [sent len, batch size, emb dim]
output, hidden = self.rnn(embedded)
#output = [sent len, batch size, hid dim]
#hidden = [1, batch size, hid dim]
assert torch.equal(output[-1,:,:], hidden.squeeze(0))
```

```
return self.fc(hidden.squeeze(0))
```

```
end of epoch 10 | time: 39.38s | valid accuracy 49.893 |
end of epoch 20 | time: 39.97s | valid accuracy 49.913 |
end of epoch 30 | time: 39.95s | valid accuracy 49.847
```

Why epoch 10?

LSTM을 epoch 30으로 실험해본 결과, epoch 10 이후에는 눈에 띄는 변화가 없었다.
CNN에 대해 epoch 30으로 실험해본 결과, 아래와 같이 지속적인 accuracy 향상이 존재했다.
그러나, 시간이 너무 오래 걸리는 등의 현실적인 이유로 이후 실험은 epoch 10으로 진행했다.

```
end of epoch 10 | time: 110.27s | valid accuracy 60.837
end of epoch 20 | time: 108.42s | valid accuracy 64.590
end of epoch 30 | time: 108.95s | valid accuracy 66.180
```

```
[108] test_file_path = '/content/drive/MyDrive/nlp/ratings_test.txt'

      test_data_pipe = dp.iter.IterableWrapper([test_file_path])
      test_data_pipe = dp.iter.FileOpener(test_data_pipe, mode='rb')
      test_data_pipe = test_data_pipe.parse_csv(skip_lines=1, delimiter='\t', as_tuple=True)
      test_data_pipe = test_data_pipe.map(removeAttribution)

      test_dataloader = DataLoader(list(test_data_pipe), batch_size=16,
                                   shuffle=True, collate_fn=custom_collate_fn)
      print('test accuracy {:8.3f} '.format(evaluate(test_dataloader)))

      test accuracy   65.920
```

<실험 결과>

사전 훈련된 임베딩을 사용한 LSTM 및 CNN 모델이 높은 정확도를 보이며, LSTM 모델이 CNN 모델보다 더 높은 정확도를 가집니다. 또한, 사전 훈련된 임베딩을 사용하지 않는 경우 모델의 정확도가 현저히 낮아질 것이라 예상했는데, CNN에서는 그러한 결과가 분명하게 드러났지만, LSTM에서는 둘 사이의 차이가 크지 않았다는 점이 이상했다.

1. Lstm with pretrained embedding
    - Epoch: 10 | Epoch Time: 1m 8s
    - Train Loss: 0.382 | Train Acc: 82.05%
    - Val. Loss: 0.373 | Val. Acc: 83.21%
    - Test Loss: 0.380 | Test Acc: 82.96%
    - Inference : 9/10

2. Lstm without pretrained embedding
    - Epoch: 10 | Epoch Time: 2m 21s
    - Train Loss: 0.267 | Train Acc: 88.99%
    - Val. Loss: 0.345 | Val. Acc: 86.62%
    - Test Loss: 0.351 | Test Acc: 86.64%
    - Inference : 8/10

3. Cnn with pretrained embedding
    - Epoch: 10 | Epoch Time: 2m 11s
    - Train Loss: 0.407 | Train Acc: 81.55%
    - Val. Loss: 0.427 | Val. Acc: 80.91%
    - Test Loss: 0.427 | Test Acc: 80.86%
    - Inference : 9/10

4. Cnn without pretrained embedding
    - Epoch: 10 | Epoch Time: 2m 5s
    - Train Loss: 0.690 | Train Acc: 54.61%
    - Val. Loss: 0.674 | Val. Acc: 61.34%
    - Test Loss: 0.674 | Test Acc: 61.50%
    - Inference : 7/10