기계학습 산업응용 project 1

1조

선지민(2018-21380)

이하연(2018-27324)

1. 문제 정의

- 데이터 : 30초짜리 7200곡 (train : 6400, validation : 800)의 장르, track duration등의 정보가 있다. 총 8개의 장르(Hip-Hop, Pop, Folk, Experimental, Rock, International, Electronic, Instrumental)에 각각 900개의 곡이 있다.
- 문제 : training 데이터로 학습시킨 CNN 모델로 validation 셋의 장르를 예측한다.
- 제출 파일 설명
  - 다음은 주어진 템플릿 파일들의 코드에 있었던 함수들을 제외하고, 변경되거나 새로 만들어진 파일 및 함수들의 목록이다.
    - main.py에서 다른 함수들을 호출해 피클을 만들고 학습시키고 테스트하도록 만들었다.
    - preprocess.py에는 raw_to_pickle함수와 load_data함수를 만들었다. 2-B에서 설명하였다.
    - <모델파일>.py에는 <모델함수>가 있다.
    - ensemble.py 에는 ensemble함수가 있다.
    - config.py에서 변수들을 정의하였다.

2. 데이터 전처리

A. Feature Extraction

- Librosa의 mfcc feature를 사용하여 피클 파일을 만들었다. Librosa의 feature 추출 함수들 중 mfcc가 성능이 가장 좋았다.
- librosa.load로 raw 음악을 가져올 때 return 되는 audio time series의 길이가 arguments를 바꿔줄 때마다 변화가 있었는데, 템플릿으로 주어진 compute_mfcc_example을 사용해서 librosa.load의 default sr이 None으로 주어지고 duration=29.0으로 추출하면 audio time series가 약 1278000이 나왔고 sr값을 default인 22050으로 주면 약 66만정도의 길이가 나온다. 그래서 Features.py의 compute_mfcc_example 함수에서 threshold값을 630000이 아닌 1278000으로 바꿔서 시도해보았을 때, 작은 데이터에서는 미미한 성능 향상을 보였으나, 전체 데이터로 모델을 테스트 해 보았을 때 성능이 크게 다르지 않았다. threshold값을 늘려서 전처리를 하게 되면 train set에서는 약 17개의 파일이 버려지게 된다 (원래의 threshold 값으로는 5개가 버려진다).
- 데이터의 feature를 뽑으면서 형성된 label의 dictionary는 genre의 알파벳순서로, 0: Electronic, 1:Experimental, 2:Folk, 3:Hip-Hop, 4:Instrumental, 5:International, 6:Pop, 7:Rock이다.

B. Pickle 파일

- 데이터를 학습시킬 때 마다 매번 raw 음악 파일을 전처리를 하며 불러오는 것이 시간이 오래 걸린다고 판단해 한 번 전 처리한 음악 데이터를 피클 파일로 만들어서 여러 모델을 학습시킬 때 피클 파일을 불러오도록 하였다. 피클 파일을 만드는 함수는 preprocess.py 파일의 raw_to_pickle 함수를 만들어서 메인에서 호출해서 사용했다.

- raw_to_pickle은 metadata의 path와 mode (train인지 validation/test인지) 를 input으로 받고 {'x' : 추출한 데이터의 feature, 'y' : label}의 딕셔너리를 mode명_data.pkl (e.g train_data.pkl, val_data.pkl)의 파일로 만들어주는 함수이다.
- 생성된 피클 파일은 preprocess.py의 load_data함수를 통해 메인에서 불러올 수 있다.
- raw_to_pickle함수의 내부에서는 mode에 따라 dataloader파일의 DataLoader함수를 호출해 feature를 만든다.

3. CNN 모델 구조

- 최종적으로 가장 좋은 성능을 보였던 모델은 다섯 개의 체크포인트 모델을 앙상블 한 모델이었고 우리 모델의 validation set의 성능 정확도는 54.25%였다.
- 모델의 구조는 Input data(mfcc feature vector; height 20, width 1231)에 대해 Convolution - Convolution - Max Pooling을 총 여섯 번 반복한 후 flatten해서 fully connected layer와 softmax를 거쳐 분류를 하게 된다. Conv1_1, conv1_2, pool1, conv2_1, conv2_2, pool2, conv3_1, conv3_2, pool3라고 명명되어 있으며, conv1_1, conv1_2의 filter 수는 64개, conv2_1, conv2_2의 filter수는 160개, conv3_1, conv3_2 filter수는 128개 이다. 모든 filter의 사이즈는 3*3이고, pool size는 2, fully connected layer의 dimension은 64를 주었다.
- Optimizer는 RMSProp을 사용했고, Activation function은 Relu를 사용했다.
- Initialization : Xavier Initialization을 사용했다.
- Regularization : convolution conv2_1, conv2_2 층과 conv4_1, conv4_2층에서만 l2 regularization을 사용했고 weight decay 파라미터 값을 1e-3을 주었다.

4. 결과 및 분석

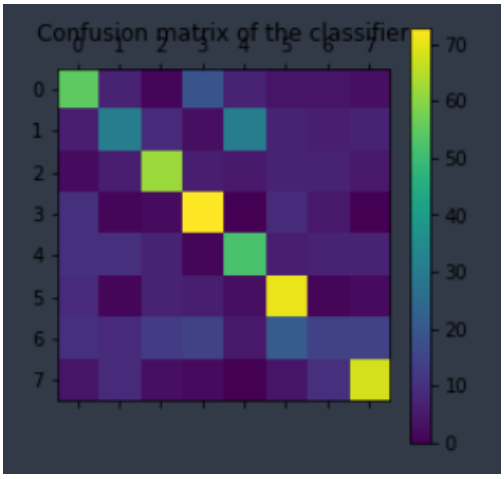
A. 모델 분석 및 결과

- Batch : 처음 template code에 있던 default 값인 32에서 시작해서 16, 10, 8, 5 등을 시도해 보았는데 5로 했을 때 가장 결과가 좋았다.
- Optimizer : SGD와 Adam, RMSProp을 테스트해보았다. SGD를 사용했을 때는 batch 5에 epoch를 300까지 했을 때도 49%에서 크게 증가하지 않았고 비효율적이라고 판단이 되어 Adam과 RMSProp을 추가적으로 테스트 해 보았다. RMSProp을 사용한 모델이 가장 성능이 좋았다.
- Regularization : 처음에는 모든 레이어에서 Regularization을 시도했는데 CNN의 층이 깊지가 않아 regularization을 대부분의 층에서 제거하고 성능을 향상시켰다.
- Threshold value : 모델들을 빠르게 돌리기 위해, 실험할 때에는 1/10 사이즈의 데이터를 사용해 초기 모델들을 테스트 했었는데, 전처리 부분에서 설명했던 features.py의 threshold값을 늘려서
- Ensemble : 48% 이상 나온 Checkpoint 모델 다섯 개를 앙상블 했을 때 성능이 가장 좋았다. 최종 모델이 앙상블 하였던 다섯 개의 모델은 다음과 같다.

CHECKPOINT	단일 성능 (단위 : %)
1029-0518	49.50
1029-0412	50.50
1029-0408	49.38
1029-0553	48.50
1029-1857	48.80
앙상블	54.25

B. 장르별 분류 결과

- Confusion Matrix는 다음과 같다.




```
array([[53,  8,  1, 19,  6,  5,  5,  3],
       [ 5, 36,  9,  3, 27,  7,  6,  7],
       [ 2,  6, 61,  6,  2, 11,  7,  5],
       [ 9,  2,  2, 74,  0,  9,  4,  0],
       [10, 13,  8,  2, 46,  6,  8,  7],
       [10,  1,  6,  4,  2, 75,  1,  1],
       [ 8,  8, 13, 15,  3, 21, 21, 11],
       [ 4,  8,  2,  3,  0,  3, 14, 66]])
```

대체로 거의 모든 장르의 분류 정확도가 비슷했으나 6번 장르인 Pop의 정확도가 많이 떨어지는 것을 볼 수 있었다.

5. Future Works

A. Slicing

- [1]에 따르면 30초의 음악 파일을 3초씩 자르고 학습시킨 후, 테스트 할 때 10개의 3초 파일을 각각 분류한 후 가장 많은 구간에서 vote를 받은 장르로 분류하는 과정을 거치면 성능이 더 좋아진다고 한다. 시간상 모든 아이디어를 구현해 보지는 못했지만, 이 아이디어에 착안해서 우리는 다음과 같은 방법을 시도해 볼 수 있다고 제시한다. 3초씩 단순히 자르는 것이 아닌, 1.5초씩 3초의 조각들이 구간이 겹치도록 파일을 자르고 30초의 음악 데이터를 10개가 아닌 19개의 파일로 만들어서 기법을 사용한다면 더 좋은 성능을 기대해 볼 수 있을 것이다.

B. Deeper CNN

- 본 프로젝트에 GPU는 GTX-1080Ti 11GB 하나를 공용으로 사용하였으나 제한된 시간 안에 더 깊은 CNN을 만들고 실험하기에 무리라고 판단하였다. VGG16 등의 깊은 층의 CNN을 regularizer, batch normalization등과 함께 사용하는 실험을 통해 결과를 더 향상시킬 수 있을 것으로 예상한다.

6. References

[1] Despois, Julien. (Nov 16, 2016). Finding the genre of a song with Deep Learning. Retrieved from https://hackernoon.com/finding-the-genre-of-a-song-with-deep-learning-da8f59a61194