GCN 을 활용한 음악 차트 분석

2015104194 이규호

요 약

구글을 만들어낸 기술인 PageRank 는 광범위한 웹사이트를 분석하여 검색 엔진의 새 시대를 열었다. 이는 단순히 사이트 검색에만 활용되는 것이 아니라 Text, 스포츠, 이미지 등 다양한 분야에서 활용될 수 있다. 본 연구에서는 이를 음악에 접목시켜 차트를 분석하고, 검증해보며 새로운 활용방안을 제시해본다.

1. 서론

1.1. 연구 배경

모바일 기기의 보편화로 콘텐츠 소비가 간편해지면서 콘텐츠의 영향력은 더욱 커졌고, 콘텐츠 산업 또한 지속적으로 성장하고 있는 추세이다. 데이터의 중요성 역시 강조됨에 따라 개인이 소비하는 콘텐츠 역시 데이터를 남기며 방대한 양의 데이터를 이용해 시장을 분석할 수 있는 가능성이 생겼다.

특히 음악 분야는 남녀노소 불문하고 소비하는 접근성이 높은 분야이다. 소비자들의 소비 패턴을 예측할 수 있는 음원 차트를 분석한다면 특정 음원에 대한 소비자의 관심을 예측할 수 있다. 각 음원들의 관계에 그래프 구조를 적용시켜, 새로운 음원의 대한 예측을 진행해본다.

1.2. 연구목표

본 연구에서는 이전 음원들의 정보와 순위를 기반으로 새로운 음원의 흥행 정도를 예측한다. 구체적으로 가수의 성별 및 인지도, 그룹 여부, 노래의 장르, 가사, 음원 사이트의 좋아요 수, 앨범의 종류, 발매일로부터의 기간, 년/월 등을 활용해서 음원의 데이터를 분석한다.

2011 년부터 2020 년까지의 음원을 사용하되, 순위가 10 의 배수인 음원은 검증 데이터로 활용한다. 이는 각 음원 간의 관계에 해당 월의 근처 음원이 밀접하다고 판단할 것이기 때문이다. 그 이외의 약 11,000 음원을 통해 GCN(Graph Convolutional Networks)를 설계한다.

2. 관련 연구

2.1. Selenium

셀레니움(Selenium)은 웹 애플리케이션 테스트를 위한 포터블 프레임워크로, 크롬 드라이버를 통해 브라우저 동작을 자동화할 수 있다. 셀레니움을 이용하는 웹크롤링 방식은 프로그래밍으로 동작을 제어해서 마치 사람이 이용하는 것 같이 웹페이지를 요청하고 응답을 받아올 수 있다.

2.2. SciPy

Python 을 기반으로 하여 과학, 분석, 그리고 엔지니어링을 위한 과학적 컴퓨팅 영역의 여러 기본적인 작업을 위한 라이브러리이다. 기본적으로 Numpy 와 함께 동작한다. 수치적분 루틴과 미분방정식 해석기, 방정식의 근, 표준 연속/이산 확률분포와 다양한 통계관련 도구들을 제공한다. Numpy 와 Scipy 를 함께 사용하면 확장 애드온을 포함한 MATLAB 을 완벽하게 대체할 수 있다.

2.3. Scikit-learn

Scikit-learn 은 2007 년 구글 썸머 코드에서 처음 구현됐으며, Python 으로 구현된 기계 학습 오픈 소스 라이브러리다. 이는 크게 지도/비지도 학습, 모델 선택 및 평가, 데이터 변환으로 나눌 수 있다. 지도 학습에서는 서포트 벡터 머신, Naïve Bayes, 결정 트리 등을 제공하며, 비지도 학습에서는 군집화, 이상치 검출 등이 있다. 모델 선택 및 평가에는 교차 검증, 파이프라인 등이 있으며, 마지막으로 데이터 변환에서는 속성 추출, 전처리 등이 있다. Python 에서 기계 학습에 대한 여러가지 기법을 쉽게 사용할 수 있다.

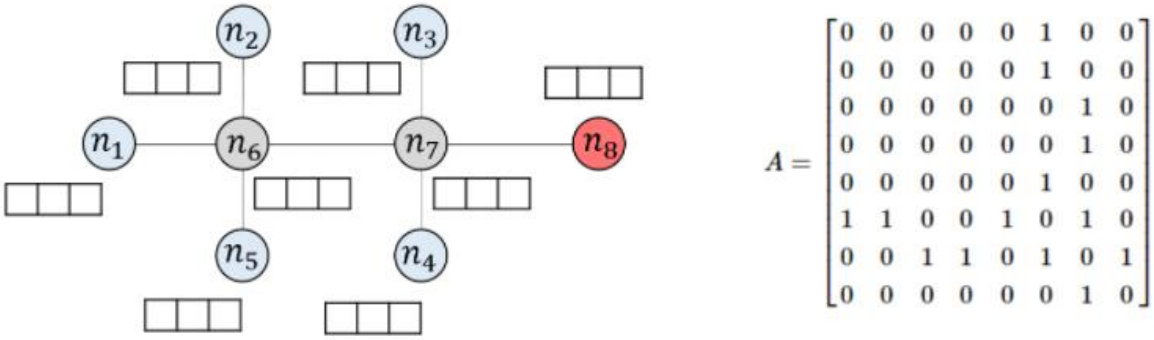
2.4. 딥러닝

딥러닝은 여러 비선형 변환기법의 조합을 통해 높은 수준의 추상화를 시도하는 기계 학습 알고리즘의 집합으로 정의되며, 큰 틀에서 사람의 사고 방식을 컴퓨터에게 가르치는 기계학습의 분야라고 이야기할 수 있다. 다양한 딥 러닝 기법들이 컴퓨터 비전, 음성인식, 자연어 처리, 음성/신호처리 등의 분야에 적용되어 최첨단의 결과를 보여주고 있다.

Python 에서는 PyTorch 를 통해서 딥러닝 개발을 진행할 수 있다. 이는 코어 데이터 구조인 Tensor 를 활용하여 수학적 연산을 가속화하며, 유연하게 딥러닝 모델을 구현할 수 있다. 또한 내부적으로 CUDA 라는 API 를 통해 GPU 를 연산에 사용할 수 있어 연산 속도를 굉장히 높일 수 있다.

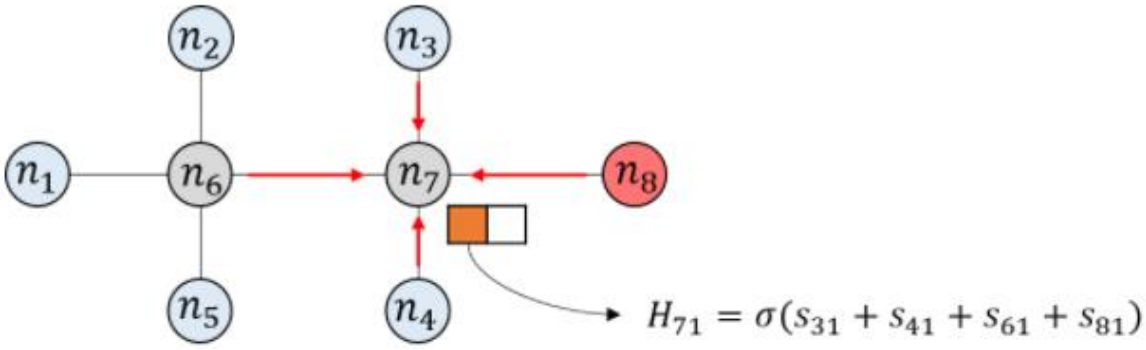
2.5. Graph Convolution

Graph Convolution 은 Graph network 에 CNN 에서의 Convolution 개념을 적용한 것이다. 여기서는 graph convolution 을 이용하여 그래프에 포함된 노드나 그래프 자체를 벡터 형태의 데이터로 변환한다. 그래프에는 노드의 연결을 나타내는 인접행렬과 노드의 feature 를 데이터로 갖는다.



[그림 1] 예시 그래프와 인접행렬

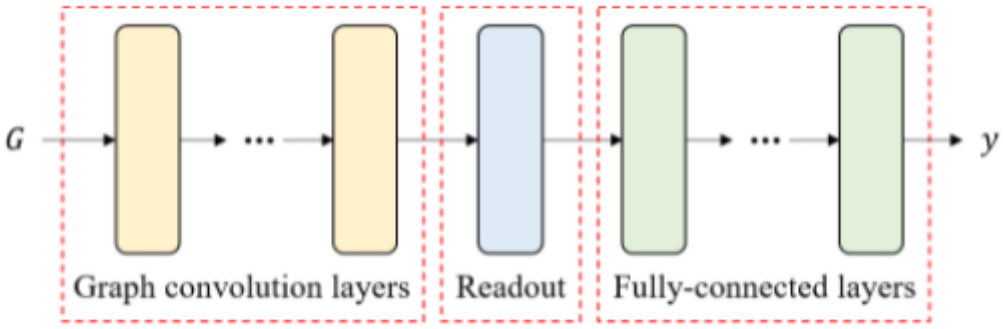
그리고 node feature 와 weight matrix 를 곱함으로써 다른 차원의 새로운 latent node feature matrix 를 생성한다. 이후 인접 행렬과 곱하면 아래 그림과 같이 각 노드의 latent feature vector 는 해당 노드의 이웃 노드를 기반으로 계산된다.



[그림 2] Graph Convolution 을 통한 노드 7 의 첫번째 latent feature 생성

2.6. GCN(Graph Convolutional Networks)

GCN 은 일반적으로 Graph convolution 으로 정의되는 graph convolution layer 와 fully-connected layer 로 구성된다. 본 연구에서 진행될 GCN 모델의 구조는 아래와 같다.



[그림 3] GCN 의 구조

여기서 readout 은 graph convolution layer 를 통해 생성된 latent feature matrix 를 그래프 전체를 표현하는 하나의 벡터로 변환하는 함수이다. 일반적으로 전체 노드의 latent feature vector 를 평균 내어 그래프 전체를 표현하는 하나의 벡터를 생성한다. Graph regression 에서는 필수적이다.

3. 프로젝트

3.1. 데이터 수집

옛날 음원의 Top 100 차트를 지원하는 멜론 사이트 월간 차트 파인더의 크롤링을 통해서 raw data 를 가져온다. 이는 동적 크롤링이 필요하므로, Python 의 Selenium 을 활용하여 데이터 수집을 진행한다.

[그림 4] 멜론의 차트파인더

그리고 각 곡들의 정보를 담을 객체를 저장한다. 멜론 사이트 자체에서 트래픽 제한을 걸어 놓았기 때문에, 시간을 두고 수집하는 절차가 필요하다. 따라서, 피클 파일을 생성해 덮어쓰기 형식으로 객체를 저장한다.

```
class Song:
    def __init__(self):
        self.year = 0
        self.month = 0
        self.rank = 0
        self.title = ''
        self.singer = ''
        self.album = ''
        self.genre = ''
        self.date = ''
        self.likes = 0
        self.lyrics = ''
```

[그림 5] Song Class

이후 필요한 데이터들을 크롬 드라이버를 통해 가져와서 저장한다.

```
for item in range(0, len(objs)):
    song = Song()
    song.year = int(year.text[:4])
    song.month = int(month.text[:-1])
    song.rank = item + 1
    href = driver.find_elements_by_css_selector('#lst50 > td:nth-child(4) > div > a')[item].get_attribute('href')
    number = re.findall('\d+', href)[0]
    driver.execute_script("window.open('https://www.melon.com/song/detail.htm?songId="+number+"');")
    driver.switch_to.window(driver.window_handles[1])
    driver.implicitly_wait(WAIT_TIME)
    song.title = driver.find_elements_by_css_selector('#downloadfrm > div > div > div.entry > div.info > div.song_name')[0].text
    song.singer = driver.find_elements_by_css_selector('#downloadfrm > div > div > div.entry > div.info > div.artist > a > span:nth-child(1)')[0].text
    song.album = driver.find_elements_by_css_selector('#downloadfrm > div > div > div.entry > div.meta > dl > dd:nth-child(2) > a')[0].text
    song.genre = driver.find_elements_by_css_selector('#downloadfrm > div > div > div.entry > div.meta > dl > dd:nth-child(6)')[0].text
    song.date = driver.find_elements_by_css_selector('#downloadfrm > div > div > div.entry > div.meta > dl > dd:nth-child(4)')[0].text
    song.likes = int(driver.find_elements_by_css_selector('#d_like_count')[0].text.replace(',', ''))
    lyrics = driver.find_elements_by_css_selector('#d_video_summary')
    if len(lyrics) > 0:
        song.lyrics = lyrics[0].text
    image_url = driver.find_elements_by_css_selector('#downloadfrm > div > div > div.thumb > a > img')[0].get_attribute('src')
    delete_idx = image_url.find('?')
    song.setImage(image_url[:delete_idx])
    driver.close()
    driver.switch_to.window(driver.window_handles[0])
    driver.implicitly_wait(WAIT_TIME)
    songs.append(song)
    print(song.rank, song.title)
```

[그림 6] Selenium 으로 음원 데이터 수집

그 이후에는 가수의 정보를 구하기 위해, 피클 파일에 저장된 객체에서 가수의 이름만을 뽑아 중복을 제거하기 위해 집합으로 만든다. 그리고 또 다시 Selenium 을 사용하여 멜론에서 가수들의 정보를 가져온다.

```
for name in singer_name[A:B]:
    singer = Singer()
    singer.name = name
    name = name.replace('#', '%23')
    name = name.replace('&', '%26')
    url = 'https://www.melon.com/search/total/index.htm?q='+ name + '&section=&searchGnbYn=Y&kkoSpl=Y&kkoDpType=&linkOrText=T&ipath=srch_form'
    driver.get(url)
    driver.implicitly_wait(WAIT_TIME)
    tmp = driver.find_elements_by_css_selector('#conts > div.section_atist > div > div.atist_dtl_info > dl > dd:nth-child(4)')[0].text
    if len(tmp) > 3:
        singer.sex, singer.group = tmp.split(',')
    else:
        singer.sex, singer.group = '.', '.'
    singer.group.strip()
    singer.fan = int(driver.find_elements_by_css_selector('#conts > div.section_atist > div > div.atist_dtl_info > div > span > span')[0].text.replace(',', ''))
    singers.append(singer)

return singers
```

[그림 7] Selenium 으로 가수 데이터 수집

이후 수집한 음원과 가수의 데이터를 csv 파일로 옮겨, 전처리에 사용할 수 있도록 한다.

3.2. 데이터 전처리

초기에는 모든 Column 을 데이터로 활용했지만, 모델을 돌리면서 성능을 오히려 저하시키는 가사 데이터는 제거했다. 결과적으로 연, 월, 장르, 발매일, 좋아요 수, 그룹 여부, 성별, 가수 좋아요 수를 feature 로 채택했다. label 값으로는 해당 노래의 순위를 주어서 결과적으로 입력 받은 노래의 순위를 예측할 수 있게끔 하였다. 각 feature 는 MinMaxScaler 를 통해 열 정규화를 해주었고, label 은 1~100 값을 가지기에, 나누기 100 연산을 해주어 정규화를 시켰다.

여기서 장르, 그룹 여부, 성별은 숫자 형태로 나타내기 위해서 key – value 형태로 사용하여 변환해주었다. 그리고 기존 발매일만 수집했기에, 현 차트의 연/월을 기준으로 연산을 하여 발매일로부터의 기간을 계산해주었다. 마지막으로 기존 음원 데이터와 가수 데이터의 join 을 통해 각 row 에 알맞은 data 를 합쳐주었다.

```
for data in songs:
    tmp = [data[0], data[1], data[8]]
    date = data[7].split('.')
    tmp.append((int(data[0]) - int(date[0])) * 12 + int(data[1]) - int(date[1]))
    g = data[6].split(',')[0]
    if genre.get(g, 0) != 0:
        tmp.append(genre[g])
    else:
        genre[g] = genre_idx
        tmp.append(genre_idx)
        genre_idx += 1
    tmp.extend(singers_dict[data[4]])

    wr_x.writerow(tmp)
    wr_y.writerow([data[2]])
```

[그림 8] feature preprocessing code

그래프의 edge 는 해당 곡의 월간 차트 ±5 곡과 해당 가수의 다른 곡들을 연결해주었다.

```
for i in range(len(songs)):
    for j in range(i + 1, len(songs)):
        if songs[i][4] == songs[j][4]:
            wr.writerow([i, j])
        elif songs[i][0] == songs[j][0] and songs[i][1] == songs[j][1] and int(songs[j][2]) - int(songs[i][2]) <= 5:
            wr.writerow([i, j])
```

[그림 9] edge preprocessing code

3.3. 모델 구현

우선, 설계한 Graph Convolution 은 다음과 같다. edge 의 인접행렬인 adj 는 sparse matrix 로 구현했기에 spmm 을 사용해주었다.

```
class GraphConvolution(Module):
    """
    Simple GCN layer, similar to https://arxiv.org/abs/1609.02907
    """

    def __init__(self, in_features, out_features, bias=True):
        super(GraphConvolution, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.FloatTensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.FloatTensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input, adj):
        support = torch.mm(input, self.weight)
        output = torch.spmm(adj, support)
        if self.bias is not None:
            return output + self.bias
        else:
            return output
```

[그림 10] Graph convolution code

이후 GCN model 을 설계해주었다. 구조는 [그림 3]과 같다.

```
class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()

        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nhid)
        self.fc1 = nn.Linear(nhid, nhid)
        self.fc2 = nn.Linear(nhid, nclass)
        self.dropout = dropout

    def forward(self, x, adj):
        x = F.relu(self.gc1(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc2(x, adj))
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

[그림 11] GCN code

만들어진 GCN 모델에 전처리한 데이터를 넣고, 학습을 시킨다. 이 때, optimizer 는 Adam 을 사용했고, loss function 은 MSE 를 사용했다. epoch 은 2000 을 사용했다. 그 이상 돌려보아도 값이 수렴하는 결과가 났다.

4. 결론

4.1. 연구 결과

train set 을 기반으로 모델을 학습시키고, test set 을 모델에 넣은 결과 0.0879 의 loss 값을 얻었다. accuracy 는 예측한 순위와 실제 순위의 차이가 x 이하인가를 기준으로 계산했다. 예측 값이 10 위 이내인 경우는 19.42%, 20 위 이내인 경우는 41.25%, 30 위 이내인 경우는 61.17%, 40 위 이내인 경우는 80.08%라는 결과가 나왔다.

```
10 Test set results: loss= 0.0879 accuracy= 19.42%
20 Test set results: loss= 0.0879 accuracy= 41.25%
30 Test set results: loss= 0.0879 accuracy= 61.17%
40 Test set results: loss= 0.0879 accuracy= 80.08%
```

[그림 12] test result

4.2. 한계점

모델의 성능을 높이기 위해 layer, dataset, activation function, optimizer, loss function 등을 바꿔가며 다양한 시도를 해보았지만, 아무래도 음원이 흥행하는 요인에는 곡과 가수에 대한 정보뿐만 아니라 소속사의 파워, 기사 수, 유행, 홍보 등 아주 다양한 요소들이 많이 포함되어 있기 때문에 기대하던 만큼의 높은 정확도를 갖지는 못했다.

4.3. 기대효과

본 연구에서는 특정 음원 사이트의 데이터만을 활용하여 진행하였기에 높은 정확도의 모델을 설계하는 것에 실패하였다. 하지만 앞서 언급한 음원의 흥행에 영향을 끼칠 만한 다양한 데이터들을 추가로 확보하여 노드의 feature 로 놓는다면, 더 높은 정확도를 기대할 수 있을 것이다.

또한 그래프의 edge 에도 더 다양한 시도를 하며 가중치를 부여하는 모델을 구현한다면 더 좋은 성능을 낼 수 있을 것이라고 예상한다.

음원 콘텐츠에만 국한되지 않고 분야를 넓혀서 더 다양한 데이터에 그래프 이론을 적용시킬 수 있을 것이라고 예측한다. 점차 생활속의 다양한 구조가 그래프로 표현되고 있으므로, 여러 상호작용을 효과적으로 분석하고 예측할 수 있을 것이다.

1. 참고 문헌

[1] Google, The PageRank Citation Ranking: Bringing Order to the Web, 1998

[2] 홍진표, 차정원, "TextRank 알고리즘을 이용한 한국어 중요 문장 추출", 2009

[3] 이도연, 장병희, "딥러닝을 이용한 음악흥행 예측모델 개발 연구", 2020

[4] 조은혜, 박재현, 최창환, "PageRank 알고리즘을 활용한 국내 배드민턴 선수들의 랭킹산정", 2018