컴퓨터과학과 학사 졸업 논문

공공 자전거 데이터를 활용한 수요 조사 및 예측 모델 프로그래밍

Development of Demand Survey and Forecasting Model Using Public Bicycle Data

2019년 11월

한국방송통신대학교

컴퓨터과학과

정 용 기

목 차

제 1 장 서론

1.1 오픈 데이터의 활용: 수요 예측의 필요성

분산 데이터 처리 기술의 발전과 함께 주목받기 시작한 빅데이터는 데이터베이스의 형태를 띄고 있거나 가공되지 않은 데이터 형태로 현대 사회에서 다양한 분야에서 맞춤 정보를 제공하고 있다.

특히, 온라인에서 그 활용이 두드러지고 있는데 그 중에서도 대형 포털 사이트나 소셜 네트워크 서비스에서 사용자 기반의 온라인 데이터 분석을 통해 쇼핑 목록을 추천해주거나, 친구를 추천하고 비슷한 단체와 모임을 주선해주기까지 온라인 활동에 편리함을 제공하고 있다.

오프라인에서도 누적된 날씨 데이터와 실시간 기상정보를 통해 보다 정확한 날씨를 알 수 있고, 산업에서는 누적된 데이터의 활용으로 생산성의 최적화 효과로 이익의 극대화로 이어지고 있다.

누적된 데이터의 활용은 우리 생활에 밀착되어 있고 현재는 그 데이터들을 좀 더 가까운 곳에서부터 활용할 필요가 있다.

오픈 데이터는 누구나 접근해서 열람하고 가공할 수 있는데 현재까지는 방대한 양은 아니지만, 여러 방면에서 데이터가 공개되고 있다. 그 중에서도 정부 주도로 공개되는 공공 데이터들은 공개된 정보에 한해 접근이 가능하기 때문에 응용 가능한 분야에서의 사용도 늘어나고 있다.

공공 자전거는 보급된 지 오래되지 않은 대중교통 수단 중 하나로, 국가나 지방자치단체에서 주로 운영하며 교통이 복잡한 대도시 주변으로 대중교통의 수요를 분산시키거나 대중교통으로 접근하기 어려운 지역까지 시민들의 편의를 위해 제공되고 있다.

지리적인 위치나 주변 교통 사정에 따라 공공 자전거의 수요는 매번 달라지는데, 출발점에서 이용가능한 자전거가 부족하거나 넘치는 현상이 계속되기도 한다. 이는 한정된 지역에서 공공 자전거에 수요에 대한 예측이 전혀 없거나 즉각적으로 반영하기 어려운 상황에 있을 수 있기 때문에 더욱 수요 예측이 필요하다.

수요 예측은 사용자의 공공자전거 이용으로 발생한 데이터로 이루어지며, 데이터를 가공하거나 있는 그대로 활용해서 최적의 조건을 찾아내는 게 가장 큰 핵심이라고 볼 수 있다. 공공자전거 도입 초창기부터 수요가 꾸준히 늘어나고 있고 이변이 없지 않는 이상 계속해서 수요는 증가할 것으로 예상된다.

예측에 필요한 데이터의 양이 너무 적으면 표본에서 예측을 하는 데 어려움이 있을 수 있다. 특히, 공공자전거 데이터 특성상 저장되는 데이터의 한계가 있을 수 있다. 예를 들면, 이용 특성상 날씨와 온도와 시간에 따라 수요가 비교적 크게 증가하거나 감소할 수 있기 때문에, 누적된 데이터 중에서 일반적인 데이터 외에 특징적인 정보가 존재하지 않는다면 날씨와 같은 조건에 따른 수요 파악이 어렵다.

공공 자전거 수요 예측을 통해 누적된 정보만으로도 어떤 시간과 어떤 날, 어떤 달에 많이 이용하는지 여부를 파악할 수 있으며 그에 따른 자전거에 유지보수 및 자전거를 정거장에 다시 위치할 수 있는 시간을 보다 효율적으로 구성할 수 있다.

이것을 위해서는 일정 기준으로 가장 많이 이용하는 정거장에 대한 수요 조사가 필요하다. 이용을 시작하는 출발점과 이용을 마치는 도착점의 수요 조사는 자전거의 평균 이동 수량을 확인할 수 있어서 주요 시작점과 도착점에 얼마나 자전거를 배치해야 할지 구체적으로 구분할 수 있게 된다.

이미 실시간으로 파악할 수 있는 시스템이 존재하지만, 수요예측을 통해 장기적으로 정거장의 거치대를 늘리고 줄이는 데 있어서 중요한 척도가 될 것으로 예상된다.

공공 자전거는 특성상 한 정거장에 정차하게 되면, 사용자가 없는 이상 계속해서 자전거가 한정거장에 누적되기 때문에 운영관리에 있어서 특정 시간마다 재배치해야 하는 어려움이 있다. 이용할 때 가장 많이 이용하는 루트를 파악할 수 있으면 마찬가지로 집중적으로 관리할 수 있는 기준을 만들 수 있다.

이 논문에서는 프로그래밍 언어 파이썬을 이용해서 노르웨이의 트론헤임 공공 자전거 오픈 데이터 CSV 파일을 분석해 가공한 뒤 시각화하고, 텐서플로우 선형회귀 인공지능을 통해 월간 공공 자전거 대여 수에 따른 다음 달의 자전거 수요 예측을 알아보는 기계학습을 한다.

1.2 공공 자전거 데이터 구성 요소

본 절에서는 데이터의 구성요소와 데이터의 형태 및 구현 방향에 대해서 설명한다.

데이터 분석에 필요한 CSV 파일은 텍스트 데이터로 쉼표, 로 데이터를 구분한 파일이다. 해당 파일은 .CSV 확장자 형태로 이용하는 소프트웨어에 따라 형태가 다르게 나타날 수 있기 때문에 파일을 이용하기 전에 형태를 확인하고 이용해야 파일이 손상되지 않는다.

```
"started_at","ended_at","duration","start_station_id","start_station_name","start_station_description","start_station_latitude","start_station_longitude"
"end_station_id","end_station_name","end_station_description","end_station_latitude","end_station_longitude"
"2019-04-01 07:08:44.434000+00:00","2019-04-01 07:11:15.482000+00:00","151","66","Verftsbrua","Ved
Brattørbrua","63.43527713641004","10.405813604593277","105","Trondheim S","Stasjonsplassen","63.43605560561","10.399405807256699"
```

그림 1.1: 트론헤임 공공 자전거 오픈 데이터 CSV 파일 내용[1]

그림 1.1 은 CSV 파일의 형태를 보여준다. 여기서 사용할 데이터는 문자열 형식의 첫번째 줄의 값들을 Column 의 첫번째 값으로 DataFrame 의 index 로도 사용한다. 이는 데이터를 행 기준으로 데이터를 검색하거나 분류할 때 유용하게 사용된다.

두번째 줄의 데이터는 1 회 이용에 대한 데이터로 여기에 있는 내용 중에서는 시작시간과 출발 정거장, 도착 정거장의 데이터를 이용한다. 특히 시작시간은 Datetime[6] 모듈로 시간 별로 분류해서 데이터를 보다 정밀하게 분류할 수 있다.

CSV 파일을 분류하기 위해서 개발 프로그램으로 파이썬 프로그래밍 언어를 이용한다.

다른 프로그래밍 언어에 비해 개발하기 쉬운 단순화된 인터프리터 형태로 프로그래밍에 소요되는 시간을 단축할 수 있고, 다양한 예제들을 통해서 구현의 어려움을 극복할 수 있다.

특히 월간 데이터의 양이 크지 않은 점을 고려했을 때 컴파일언어의 장점도 상쇄할 수 있어서 개발에 큰 이점이 있다.

제 2 장 파이썬 모듈 로드 및 개발

2.1 파이썬 라이브러리

동적 자료형 파이썬은 변수 선언이 필요 없는 특징과, 사용 중 함수의 자료형을 바꾸거나 테스트가 가능하다는 점, 직관적인 코드로 개발 속도가 빠른 점, 파이썬의 간결한 문법 구조 등이 장점이다.

비록 성능이 다른 언어에 비해 비교적 떨어지는 단점이 있지만, 이 부분은 모듈의 계속된 발전을 통해 개선되고 있다.

Pandas	데이터 분석 패키지 1 차원 Series , 2 차원 DataFrame , 3 차원 Panel
Numpy	C 로 구현된 라이브러리로서 고성능 수치계산을 수행, Panda 기반
Matplotlib	데이터를 시각화해 데이터나 차트로 표현
Scipy	엔지니어링을 위한 과학 라이브러리, 함께 import 한 라이브러리들과 함께 동작
Seaborn	차트에 기능을 추가하는 시각화 패키지
Datetime	날짜와 시간을 조정하며 세부적으로 시간을 나눔

```
[12] import pandas as pd
     import numpy as np
     import matplotlib as mpl
     import matplotlib.pyplot as plt
     import matplotlib.patches as mpatches
     import seaborn as sea
     from scipy import stats
     import datetime

     #그래프 그리기
     %matplotlib inline

     #그래프 가독성
     plt.style.use('ggplot')

     #마이너스 폰트 깨지는 것을 수정
     #mpl.rcParams['axes.unicode_minus'] = False

[13] #드라이브 마운트
     from google.colab import drive
     drive.mount('/content/gdrive')

[14] #파일 가져오기
     train = pd.read_csv("/content/gdrive/My Drive/PAPER/DATA/190410.csv", parse_dates=["started_at"])
     train.shape

     #CSV 파일 내부의 행 쌍따옴표 제거
     def dequote(s):
         if (s[0] == s[-1]) and s.startswith(("'", '"')):
             return s[1:-1]
         return s
```

그림 2.1: 사용될 모듈 로드 및 CSV 파일 로드 소스 코드

그림 2.1 에서 2019 년 04 월부터 10 월까지 공공 자전거 데이터의 내용을 병합한 CSV 파일을 train 으로 로드하고 시계열 데이터로 parse_date 를 CSV 파일 내의 started_at 으로 처리했다.

CSV 데이터 내의 요소마다 쌍 따옴표 “” 처리되어 있어 원활하게 분류하기 위해 제거했다.

2.2 총 수요 데이터 분류

CSV 데이터 train 의 시간 처리를 위해서 Datetime 모듈 .dt 를 그림 2.2 와 같이 사용하는데, train 을 모듈들을 사용해 (rental, year, month, day, hour)로 분류해 index 와 Column 을 추가했다.

대여 수인 rental 은 일간 대여 수량을 확인할 수 없기 때문에, 단순 집계를 위해서 추가한 것이고 나머지는 각각의 대여 현황 데이터 처리를 위해서 시간대 별로 나누었다.

입력 결과로 데이터 내의 각각 DataFrame 에 해당 시간대가 입력된 데이터 프레임으로 다시 구성되었다.

```
[19] #started_at 의 데이터를 시간별로 분류, 각 행을 하나로 카운트하는 rental 열 추가
     train["rental"] = 1
     train["year"] = train["started_at"].dt.year
     train["month"] = train["started_at"].dt.month
     train["day"] = train["started_at"].dt.day
     train["hour"] = train["started_at"].dt.hour
     train["aday"] = train["started_at"].dt.weekday

[25] train.tail()
```

	station_name	end_station_description	end_station_latitude	end_station_longitude	rental	year	month	day	hour	aday
	Hesthagen	Ved Klæbuveien	63.415418	10.399565	1	2019	10	30	22	2
	TMV-odden	Solsiden	63.435399	10.409983	1	2019	10	30	22	2
	Strandveikaia	langs Stiklestadveien ved Strandveien	63.442090	10.427939	1	2019	10	30	22	2
	Bakke bru	Ved holdeplass Bakkegata. Operativt betjenings...	63.432252	10.406996	1	2019	10	30	22	2
	Lademoparken	ved kryss Mellomveien / Innherredsveien. Opera...	63.436605	10.425844	1	2019	10	30	22	2

```
[21] #train 데이터에서 started_at을 년,월,일,시간으로 나눈 후 각 단위마다 그룹화한 뒤, 그룹화된 단위의 수량을 체크
     by_year = train.groupby('year').rental.sum()
     by_month = train.groupby('month').rental.sum()
     by_day = train.groupby('day').rental.sum()
     by_hour = train.groupby('hour').rental.sum()

[30] by_month

     month
     4     34793
     5     49873
     6     46624
     7     37980
     8     66119
     9     55424
     10    43746
     Name: rental, dtype: int64
```

그림 2.2: CSV 파일 기반 사용할 데이터 분류 소스 코드

집계함수 group_by[2] 를 이용해서 그룹을 지정할 수 있으며, 여기서 year, month, day, hour 단위로 용도에 맞게 by_ 형태로 그룹화했다.

이 작업은 CSV 데이터 train 의 2 차원 DataFrame 형태에서 1 차원 Series 로 변환되었는데, 각각의 단위마다 Index 값과 Value 값을 가지는 구조가 되었다.

그리고, by_ 마다 Rental 을 sum ()으로 합산해서 공공 자전거의 총 수량을 볼 수 있다.

2.3 총 수요에 따른 그래프

그림 2.3 의 figure 함수는 그래프를 표현할 액자를 만들어주며, import 된 matplotlib 에서 figure 를 생성하고 관리한다. 액자 리스트 ax 를 개별 개체를 설정하고, plt.subplots 으로 프레임의 Column 과 Row 의 개수를 정한다.

seaborn[3][4][5]은 데이터에 시각화를 도와줄 패키지로 이번 차트에서 막대그래프와 점 그래프를 이용했다. Data 에 원본 데이터를 설정하고, 각각 X, Y 값으로 by_(year, month, day, hour)을 입력하고, X 값만 .tolist 를 이용해서 2 차원 리스트로 출력한다. 마지막으로 .set 을 이용해서 X 와 Y 의 label, title 로 라벨과 이름을 설정했다.

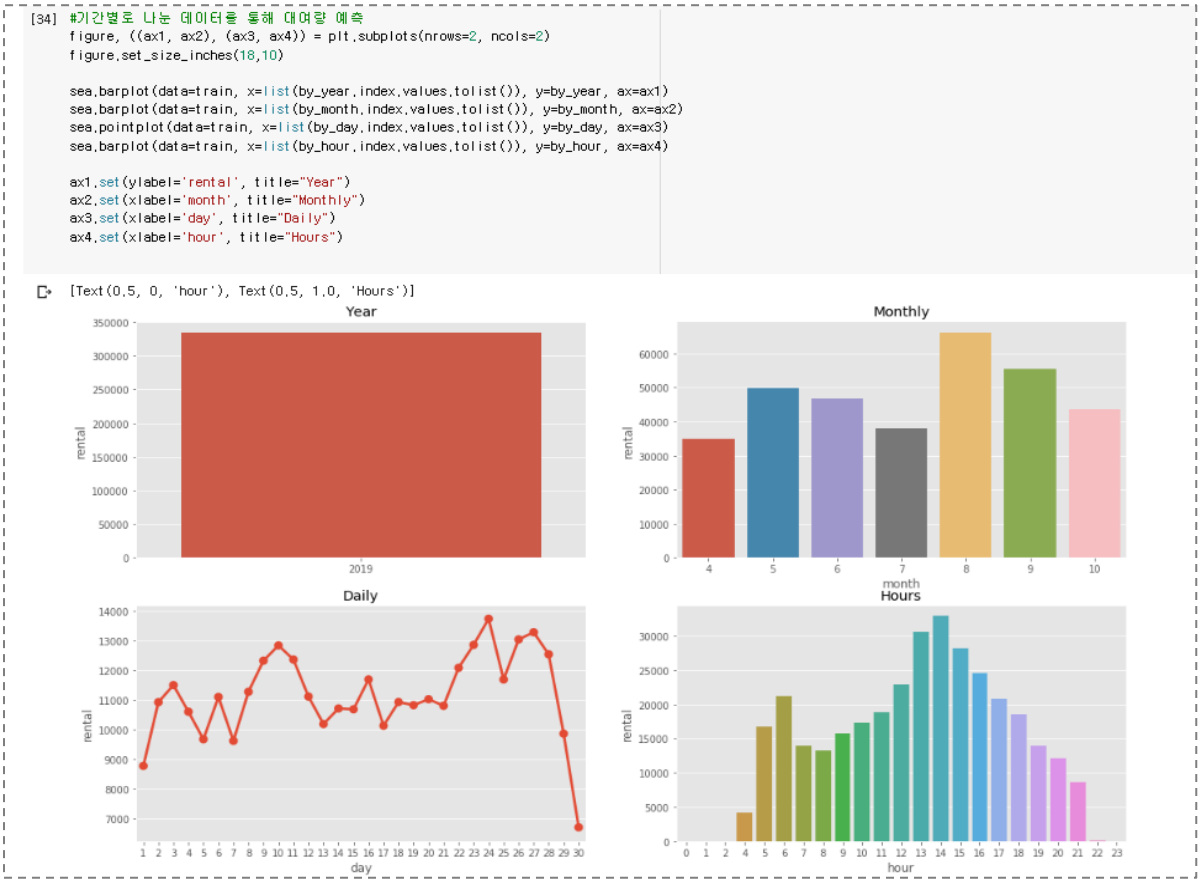


그림 2.3: CSV 파일 기반 분류된 데이터 분석 결과

그림 2.3의 분류 결과를 구역 특성 기반으로, 월간 수요의 변화는 4월의 트론헤임은 초봄의 날씨이며, 7월은 덥고 휴가철이라 낮은 수요 수치를 보여준다. 반면, 8월은 초가을로 활동하기 좋고, 대학교 중심 도시이기 때문에 개강과 활동 증가에 따라 수요가 증가했다. 이후는 안정화되어 감소하는 것을 볼 수 있다.

일간 수요는 노르웨이 사람들은 주말인 일요일은 종교활동이나 여행, 가족들과 시간을 보내기 때문에 도시에서 거의 활동하지 않아서 항상 감소한다. 그럼에도 평균적으로 중심곡선이 유지되는 것은 항구의 관광객이나 일회성 이용자보다 고정적으로 이용자들이 많기 때문이다. 30일은 해당날짜 CSV 데이터가 없어서 하락했다.

시간별 데이터는 아침을 일찍 시작하는 노르웨이 사람들의 생활 패턴을 보여주는데, 아침 6시에 높은 수치를 보여주고 이후 오후 2시를 중심으로 학생들의 하교시간과 직장인들의 퇴근시간이 몰렸다가 줄어들면서 수요가 변한다.

제 3 장　파이썬 제어문 활용 개발

3.1 제어문 활용 데이터 분류

앞에서 간단한 몇 가지 함수만으로 데이터를 처리하고 그래프를 만들었다. 결과는 총 수요량의 변화로 언제 어느 시점에 이용이 증가하고 감소하는지 파악할 수 있었는데, 구체적으로 데이터에 접근이 필요하다.

정렬되는 데이터를 이용해서 가로막대 그래프로 표현하고 그래프의 끝에는 가독성을 위해 대여 수량을 표시한다. 시작점과 도착점 기준으로 가장 많은 순위 5 위까지 표현할 것이다.

```
[ ]  for monthly in range(1, 13):
       exec("si%02d_19 = train[train['month'].isin(['%02d'])]['start_station_name'].value_counts(normalize = False).head(5)" % (monthly, monthly))

[ ]  for monthly in range(1, 13):
       exec("s%02d_19 = train[train['month'].isin(['%02d'])]['start_station_name'].value_counts(normalize = False).head(5)" % (monthly, monthly))
       exec("e%02d_19 = train[train['month'].isin(['%02d'])]['end_station_name'].value_counts(normalize = False).head(5)" % (monthly, monthly))
       exec("slist_%02d_19 = pd.DataFrame({'Station':s%02d_19.index, 'Rental':s%02d_19.values})"% (monthly, monthly, monthly))
       exec("elist_%02d_19 = pd.DataFrame({'Station':e%02d_19.index, 'Rental':e%02d_19.values})"% (monthly, monthly, monthly))
```

그림 2.1: CSV 파일 기반 월별 데이터로 분류 소스 코드

for 는 변수 monthly 와 리스트 range (n, m):를 통해 문장을 반복적으로 수행하는 매우 직관적이며 한눈에 들어오는 문장 구조로 파이썬 만의 특징을 잘 보여준다.

공식적으로 파이썬 2 에서 사용하는 for 의 포맷방식은 권장하지 않지만, 여기서는 Pandas 의 DataFrame 을 선언할 때 {}을 사용해서, 최신 포맷방식인 f-string 이나 .format 은 변수를 입력할 때 마찬가지로 {}을 이용해야 하기 때문에 사용할 수 없었다. 하지만 이 방식도 전처리를 해주면 사용 가능하다.

그림 3.1 에서 monthly 는 range ( ):에서 1 에서 13 까지 숫자가 차례대로 반복되는 구조로 : 콜론 다음 아래의 문장을 반복한다.

exec ( ) 내장 함수는 코드를 동적 실행을 지원하며, 문장 자체를 입력 받는데 = 을 중심으로 왼쪽은 si%02d_19 에 오른쪽을 담는 구조로, %02d 가 변수로 반복된다. 여기서 %02d 형식은 숫자가 반복될 때, 1, 2, 3 형식이 01, 02, 03 형식으로 반복된다. 문장 맨 끝에 %monthly 를 입력한 개수만큼, % 가 입력된 곳을 타겟으로 설정하고 반복된다.

우선, 시작점과 도착점을 분류하는데 s_ 와 e_ 형식으로 Series 와 DataFrame 을 분류한다.

첫번째 Series 데이터는 train 에서 month 을 기준으로 .isin 명령어로 %02d 에서 monthly 하나를 받아 분리되고 각각 start_station_name 을 많은 순서대로 value_counts(normalize=False)을 이용해 내림차순으로 .head ( )가 5 개를 출력된다.

그림 3.1 의 마지막 두줄은 앞에서 만든 Series 를 이용해 DataFrame 을 생성한다.

pd.dataframe 으로 DataFrame 을 생성하는데, Series 의 index 값을 DataFrame 의 Station 에 담고, value 값을 Rental 에 담았다.

3.2 제어문 활용 월간 정거장 수요 그래프

이 절에서는 3.1 절에서 가공된 데이터를 제어문을 이용해 월간 정거장 수요를 파악하는 그래프를 작성한다.

```
[37] #정리된 CSV 목록 중에서 데이터가 없는 달을 제외, 그래프로 표기 하기 위해 준비
     for monthly in range(1, 13):
       exec("filter1 = slist_%02d_19"% (monthly)) #가장 많이 출발하는 정거장
       if len(filter1) == 0: #len값이 0이면 무시한 뒤 else
         continue
       else: #데이터가 있는 달을 건네받아 그래프화
         exec("graph1 = slist_%02d_19.sort_values(by = ['Rental'], axis = 0)" % (monthly))
         graph1.plot(kind = "barh", legend = False, width = 0.8, color = 'skyblue')
         for i, (p, Rental) in enumerate(zip(graph1.Station, graph1.Rental)):
           plt.title("%02d 2019 Depature" %(monthly), size=20, color='black')
           plt.text(s = p, x = 1, y = i, color = "black", verticalalignment = "center", size = 18)
           plt.text(s = str(Rental), x = Rental-300, y = i, color = "black", verticalalignment = "center", horizontalalignment = "left", size = 18)
         #plt.axis("off")
         #plt.xticks([])
         #plt.yticks([])
         plt.tight_layout()
         plt.savefig("data1.png")


     for monthly in range(1, 13):
       exec("filter2 = elist_%02d_19"% (monthly)) #가장 많이 도착하는 정거장
       if len(filter2) == 0: #len값이 0이면 무시한 뒤 else
         continue
       else: #데이터가 있는 달을 건네받아 그래프화
         exec("graph2 = elist_%02d_19.sort_values(by = ['Rental'], axis = 0)" % (monthly))
         graph2.plot(kind = "barh", legend = False, width = 0.8, color = 'skyblue')
         for i, (p, Rental) in enumerate(zip(graph2.Station, graph2.Rental)):
           plt.title("%02d 2019 Arrived" %(monthly), size=20, color='black')
           plt.text(s = p, x = 1, y = i, color = "black", verticalalignment = "center", size = 18)
           plt.text(s = str(Rental), x = Rental-300, y = i, color = "black", verticalalignment = "center", horizontalalignment = "left", size = 18)
         #plt.axis("off")
         #plt.xticks([])
         #plt.yticks([])
         plt.tight_layout()
         plt.savefig("data2.png")
```

그림 3.2: 가공된 데이터로 월간 시작점과 도착점 기준 수요 순위 분류 소스코드

그림 3.2 는 반복하는 변수 monthly 와 리스트 range ( )를 통해 아래의 exec, if, continue, else 들을 처리하는데, 특히 if 의 len 함수를 이용해서 가공된 데이터 중 빈 값 0 인지 확인한다. 빈 값을 분류하지 않을 경우 강제로 데이터를 처리하다가 빈 값을 감지하고 오류가 발생한다.

그래서 값이 0 이면, continue 를 이용해서 해당 순서를 무시하고 반복을 계속하게 해준다.

반복문에 값이 있어서 else 처리된 가공된 데이터는 그래프를 작성하기 위해 exec 의 graph1 로 입력된다.

3.1 절 에서 만들어진, slist_ 의 Rental 대여 수량을 기준으로 정렬한다.

그래프는 바 형태로 나머지 범례와 폭, 색상을 설정한다. 여기서 for 는 enumerate 속성으로 List 나 String , Tuple 을 건네 받아 index 에 포함시키는데, 그래프의 Station 의 이름과 Rental 이용 수량이 zip 으로 병렬 처리되어 index 에 포함되어 그래프에 나타난다.

마지막으로 plt 로 그래프의 제목과 각종 속성을 정하고, 이미지 파일을 저장했다.

시작점과 마찬가지로 도착점 역시 같은 방식으로 처리한다.

그림 3.3 에서는 2019 년 각 월별 시작점과 도착점 기준 이용 수요에 따른 순위권 그래프로 나타냈다.

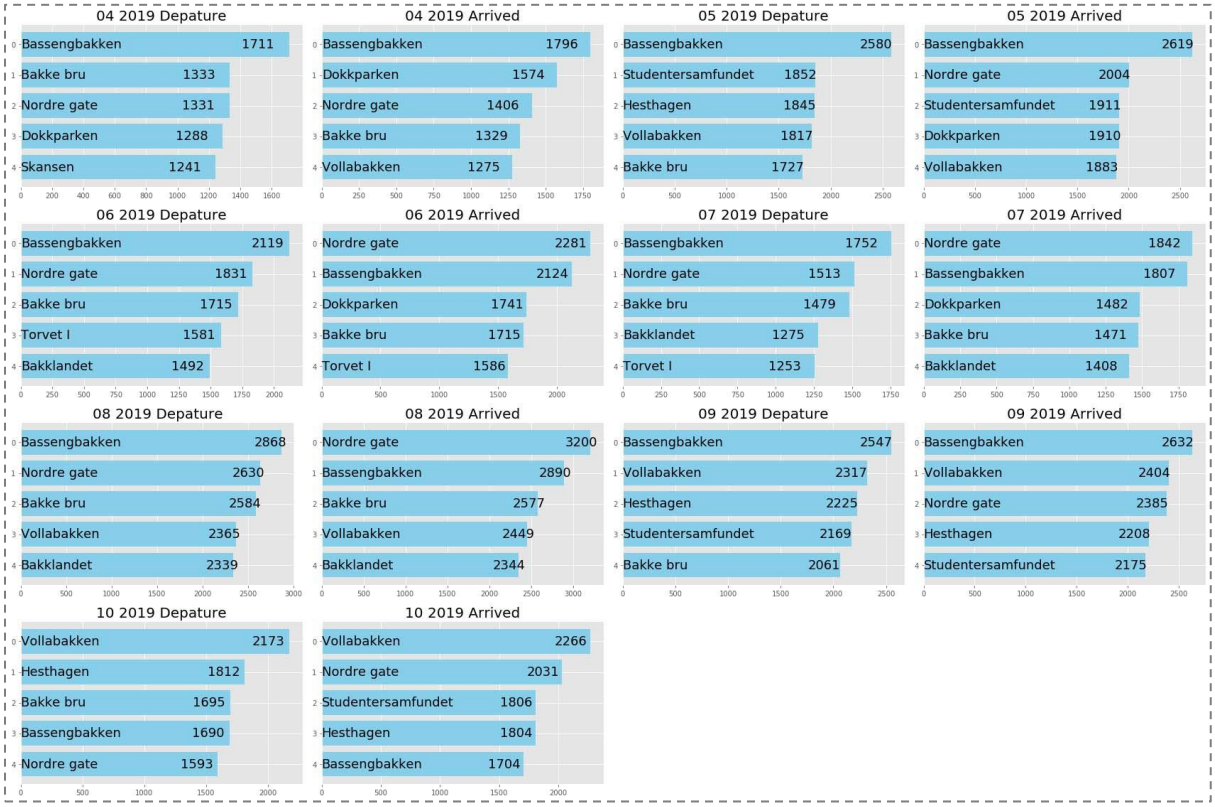


그림 3.3: 가공된 데이터로 월간 시작점과 도착점 기준 수요 순위 분류 결과

그림 3.3 의 분류 결과를 구역 특성 기반으로, 시작점 기준으로 가장 수요가 많은 정거장 중에서 Bassengbakken 은 트론헤임의 Solsiden 으로 상업지구이자 부둣가 앞이다. 이 지역은 쇼핑몰과 식당이 상당히 몰려 있어서 공공 자전거의 수요 역시 비례하게 많은 것을 보여준다.

10 월에 수요가 급격하게 바뀌었는데, 8 월부터 순위권에 들어온 Vollabkken 은 트론헤임의 대학교 앞에 위치하는 정거장으로 개강과 함께 학생들의 이용 수요가 증가한 것으로 보인다.

도착점 기준으로 가장 수요가 많은 것은 역시 Bassengbakken 이다.

5 월부터 Nordre gate 의 수요가 증가했는데, 트론헤임의 상업지구의 중심지로 상점들과 쇼핑몰이 즐비한 곳이며, 다른 지역으로 가는 교통이 마주치는 곳이다. 날씨가 괜찮은 6 월 7 월 8 월을 기준으로 이용객이 증가폭을 보인다.

특징이 있다면, 기상 조건인 비가 많이 오거나 눈이 와도 자전거를 타기 때문에 이 때문에 전체적인 수요량의 대폭 감소는 보이지 않는다.

전체적인 자전거의 수요는 월간 총 수요량 대비 증가하는 추세이다.

3.3 제어문을 활용 월간 최대 이용 경로 분류

공공 자전거를 이용하는 사람들의 시작점과 도착점 데이터가 비슷하게 나타나는 반면, 월간 가장 많이 이용하는 루트는 어떤 루트인지 분류했다.

```
[120] for monthly in range(1, 13):
        exec("si%02d_19 = train[train['month'].isin(['%02d'])]"%(monthly, monthly)) # 전체 CSV에서 month 기준 각 달 분류

[121] for monthly in range(1, 13):
        if len("si%02d_19"% (monthly)) == 0: #len으로 값이 비었는지 검사 비었으면 무시 정거장 분류 전
          continue
        else:
          for ssid in range(1, 294): #해당 월에서 정거장 1 ~ 293까지 각각 분류하여 이름 저장 (비어있는 값에 대해선 카운트
            exec("ss%03d_%02d_19 = si%02d_19[si%02d_19['start_station_id'].isin(['%03d'])]" %(ssid, monthly, monthly, mo
            if len("ss%03d_%02d_19"% (ssid, monthly)) == 0: #해당 월의 각 시작지점이 묶이 값을 하나씩 비어있는지 검사
              continue #비어있으면 무시 #시작 정거장을 각각으로 분류함
            else:
                exec("es%03d_%02d_19 = ss%03d_%02d_19['end_station_name'].value_counts(normalize=False).head(5)"% (ssid,
                if len("es%03d_%02d_19"% (ssid, monthly)) == 0:
                  continue # 분리한 시작 정거장에서 도착 정거장을 그룹화해서 순위대로 표기, 상위 5개로 분류함
                else:
                  exec("r%03d_%02d_19 = pd.DataFrame({'month':ss%03d_%02d_19['month'].head(), 'Depature': ss%03d_%02d_19
                  if len("r%03d_%02d_19"% (ssid, monthly)) == 0:
                    continue #분리한 시작 정거장에서 가장 많이 가는 도착 정거장 5가지로 데이터프레임을 생성
                  else:
                    pass
    #그래프 구역(이전에 탭행킹 뽑아야함

[ ]  #달 범위 생성
     for omg in range(1, 13):
         exec("rl_%02d_19 = pd.concat([r001_%02d_19, r002_%02d_19, r003_%02d_19, r004_%02d_19, r005_%02d_19, r006_%02d_

[264] rl_04_19.sort_values('Rental1',ascending=False).head(5)
      rl_05_19.sort_values('Rental1',ascending=False).head(5)
      rl_06_19.sort_values('Rental1',ascending=False).head(5)
      rl_07_19.sort_values('Rental1',ascending=False).head(5)
      rl_08_19.sort_values('Rental1',ascending=False).head(5)
      rl_09_19.sort_values('Rental1',ascending=False).head(5)
      rl_10_19.sort_values('Rental1',ascending=False).head(5)
```

그림 3.4: 가공된 데이터로 월간 수요가 가장 많은 루트 순위 분류 소스 코드

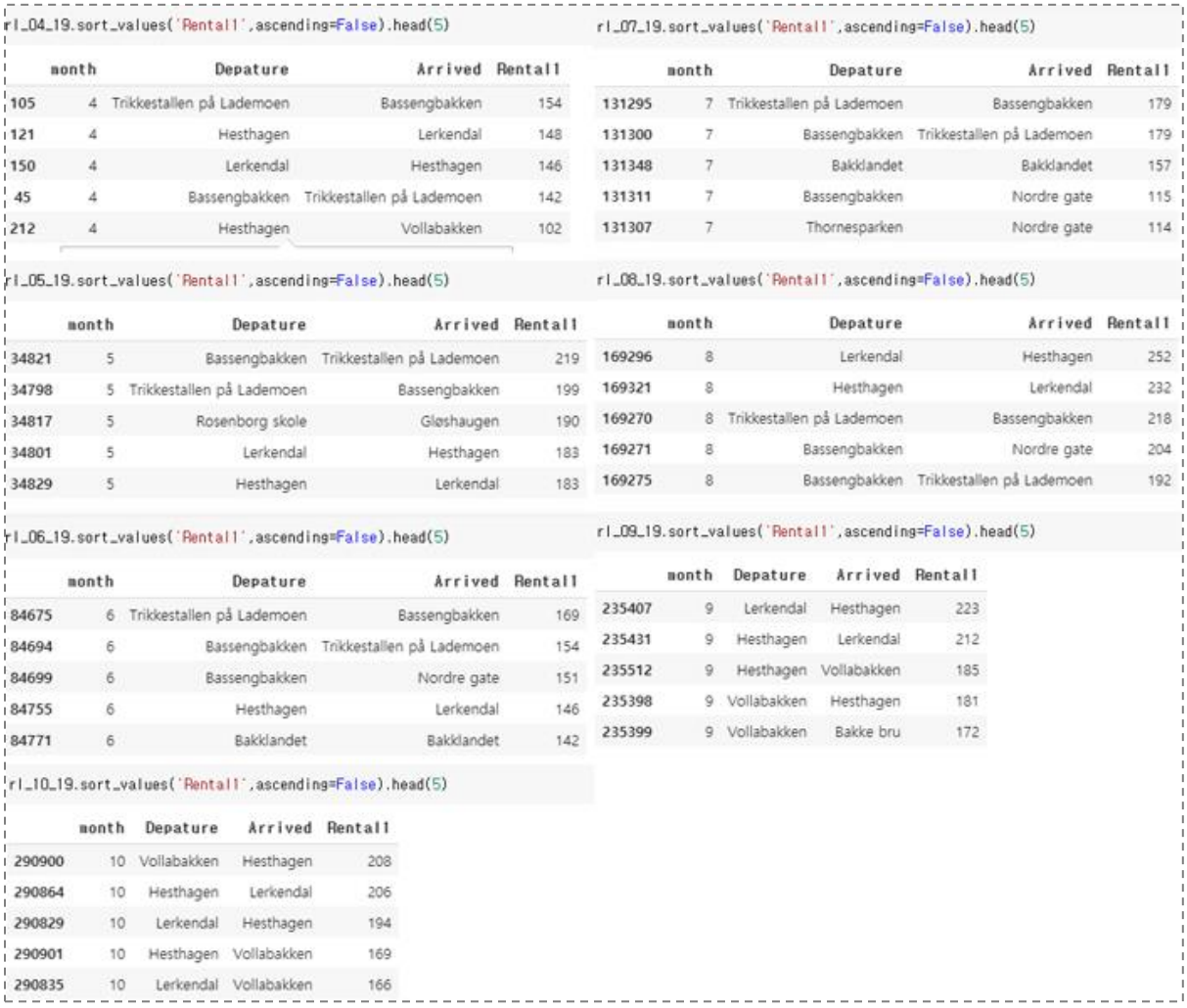
그림 3.4 의 CSV 데이터 train 에서 for 를 이용해서 month 의 값을 .isin 로 추출한 뒤, 각각 달로 분류되는 si%02d_19 로 저장한다. for 로 len 검사 후 다음 단계 else 로 넘긴다. else 에서 for 를 이용해 ss%03d_%02d_19 에 담을 데이터인 앞에서 분류된 si%02d_19 의 start_station_id 시작점 기준으로 month 를 분류한다. 시작 정거장은 기준으로 도착점을 구하는 고정 값이기 때문에, head 를 지정하지 않는다. 지정 시 배열이 맞지 않아 오류가 발생한다.

도착점 데이터는 end_station_name 을 분류하는 과정 까지만 같고, 도착점의 총 합을 구하는 value_count(normalize=False)로 상위 5 개의 값만 출력한다.

ss%03d_%02d_19 시작점과 es%03d_%02d_19 도착점의 Series 데이터를 r%03d_%02d_19 에 담게 되는데, pd.dataframe 을 이용해서 2 차원 DataFrame 화 한다. 시작점에서 month 의 index 값과 Depature 의 column 값을 가져오고 도착점에서는 arrived 의 index 를 가져오고 Rental1 의 value 값을 가져와서 DataFrame 을 만든다.

r%03d_%02d_19 는 month 별로 각각 정류장마다 출발점과 도착점의 순위가 5 개씩 저장된다. rl_%02d_19 를 데이터화 하는데 pd.concat 은 for 를 받지 않아 매크로로 수기 입력했다. 구현하지 못한 for 를 제외하고 rl_%02d_19 는 각각 다음과 같이 그래프를 출력한다.

월 마다 시작점과 도착점 수요가 가장 많은 정거장을 month 별 총 293 개의 정거장 데이터에서, 시작점 개별마 다 가지는 294 개의 도착점의 순위 중 1 위만을 선별한 뒤, 선별한 데이터를 모아서 DataFrame 화 한 뒤, 내림차순 5 순위로 구성했다.



```
rl_04_19.sort_values('Rental1',ascending=False).head(5)
```

	month	Depature	Arrived	Rental1
105	4	Trikkestallen på Lademoen	Bassengbakken	154
121	4	Hesthagen	Lerkendal	148
150	4	Lerkendal	Hesthagen	146
45	4	Bassengbakken	Trikkestallen på Lademoen	142
212	4	Hesthagen	Vollabakken	102

```
rl_07_19.sort_values('Rental1',ascending=False).head(5)
```

	month	Depature	Arrived	Rental1
131295	7	Trikkestallen på Lademoen	Bassengbakken	179
131300	7	Bassengbakken	Trikkestallen på Lademoen	179
131348	7	Bakklandet	Bakklandet	157
131311	7	Bassengbakken	Nordre gate	115
131307	7	Thornesparken	Nordre gate	114

```
rl_05_19.sort_values('Rental1',ascending=False).head(5)
```

	month	Depature	Arrived	Rental1
34821	5	Bassengbakken	Trikkestallen på Lademoen	219
34798	5	Trikkestallen på Lademoen	Bassengbakken	199
34817	5	Rosenborg skole	Gløshaugen	190
34801	5	Lerkendal	Hesthagen	183
34829	5	Hesthagen	Lerkendal	183

```
rl_08_19.sort_values('Rental1',ascending=False).head(5)
```

	month	Depature	Arrived	Rental1
169296	8	Lerkendal	Hesthagen	252
169321	8	Hesthagen	Lerkendal	232
169270	8	Trikkestallen på Lademoen	Bassengbakken	218
169271	8	Bassengbakken	Nordre gate	204
169275	8	Bassengbakken	Trikkestallen på Lademoen	192

```
rl_06_19.sort_values('Rental1',ascending=False).head(5)
```

	month	Depature	Arrived	Rental1
84675	6	Trikkestallen på Lademoen	Bassengbakken	169
84694	6	Bassengbakken	Trikkestallen på Lademoen	154
84699	6	Bassengbakken	Nordre gate	151
84755	6	Hesthagen	Lerkendal	146
84771	6	Bakklandet	Bakklandet	142

```
rl_09_19.sort_values('Rental1',ascending=False).head(5)
```

	month	Depature	Arrived	Rental1
235407	9	Lerkendal	Hesthagen	223
235431	9	Hesthagen	Lerkendal	212
235512	9	Hesthagen	Vollabakken	185
235398	9	Vollabakken	Hesthagen	181
235399	9	Vollabakken	Bakke bru	172

```
rl_10_19.sort_values('Rental1',ascending=False).head(5)
```

	month	Depature	Arrived	Rental1
290900	10	Vollabakken	Hesthagen	208
290864	10	Hesthagen	Lerkendal	206
290829	10	Lerkendal	Hesthagen	194
290901	10	Hesthagen	Vollabakken	169
290835	10	Lerkendal	Vollabakken	166

그림 3.5: 가공된 데이터로 월간 수요가가장 많은 루트 순위분류 결과

그림 3.5 의 분류 결과를 구역 특성을 기반으로, Trikkestallen pa Lademoen 시작점은 아파트 밀집 지역으로 Bassengbakken 을 향하는 사용자가 최대 수요로 계속해서 나타난다. 이 정거장은 많은 수요로 인해 정거장에 공공 자전거가 없는 경우가 많다.

8 월에 나타나는 Lerkendal 은 트론헤임의 주요 교통 정체구역 중 한 곳으로 다른 지역으로 나가는 버스노선이 교차하는 지역이다. Lerkendal 시작점에서 향하는 Hesthagen 도착점은 대학교가 가까운 위치로 주로 Lerkendal 교차로에서 버스를 기다리지 않고 학교로 가는 학생들의 고정 수요가 있고, 최근 변경된 Hesthagen 주변 버스 정거장의 폐쇄로 8 월부터 꾸준히 순위권에 위치하므로 변수 사용자가 늘어났다.

이로서 대학 중심의 도시 주변의 공공 자전거 수요는 방학과 개강에 따라 급격하게 증가하거나 감소하는 것을 볼 수 있다.

제 4 장 텐서플로우 선형회귀를 통한 수요 예측

4.1 텐서플로우 선형회귀(Linear Regression)

직선에 가까운 선형적인 구조를 가지는 변수 데이터들의 형태를 모델링 한 것을 선형 회귀라고 한다. 단순히 점 두개의 데이터는 선을 긋게 되면 직선이 되겠지만, 두개 이상의 점이 존재할 경우에 직선에 가깝게 그리고자 할 때 합리적인 방법을 찾게 될 것이다.

선형회귀[7][8]는 학습을 통해 합리적인 식을 찾아내는데 목적을 두지만, 근사값 정도로 예측할 수 있다는 것을 염두해야 한다. 수행 과정에는 표본이 많으면 많을수록 좋은데 여기서 사용할 수 있는 공공 자전거 데이터는 해당 월간 총 자전거 수요에 따른 다음달 자전거 수요를 예측해보려 한다.

$$H(x) = Wx + b \tag{1}$$

수학에서 1차 방정식으로 공식을 둔다. 수식 1.1의 H는 가설을 뜻하는 Hypothesis , W는 Weight , b는 basis를 의미한다.

텐서플로우는 수식 1.2의 변수 W와 b를 tf.Variable 클래스를 사용하며, 이는 값과 이름을 지정하고 random_normal 을 이용해서 랜덤 값으로 [1]차원 배열에 삽입한다.

$$\begin{aligned} W &= tf.Variable(tf.random\_normal([1]), name = 'weight') \\ b &= tf.Variable(tf.random\_normal([1]), name = 'basis') \end{aligned} \tag{2}$$

가설을 세우는데 있어 가장 중요한 부분은 비용 cost로 가설의 정확도를 판단하는 기준이다.

비용은 예측되는 값과 실제 값의 차이, 데이터와 직선과의 거리를 뜻하는데 비용이 낮을수록 모델의 완성도가 높아진다. 비용을 낮추는 방법으로 텐서플로우에서 제공하는 경사 하강 라이브러리를 이용해 프로그램을 작성할 수 있다.

4.2 비용 (Cost)

텐서플로우에서 비용은 임의로 세운 가설의 정확도를 판단하는 기준이 된다. 그림 4.1을 참고하면, X축은 월을 나타내고, Y축은 월간 수요량, 파란점이 각각 실제 수요를 나타낸다.

파란점의 분포의 형태에 따라서 1차 함수 직선(붉은선)으로 가설을 세우면 점과 직선 사이의 거리를 비용 함수로 계산할 수 있다.

점과 직선 사이의 거리를 구할 때 예측 값에서 실제 값을 뺀 뒤, 거리 값이기 때문에 음수를 없애기 위해 뺀 값에 제곱을 하면 평균을 구할 수 있다.

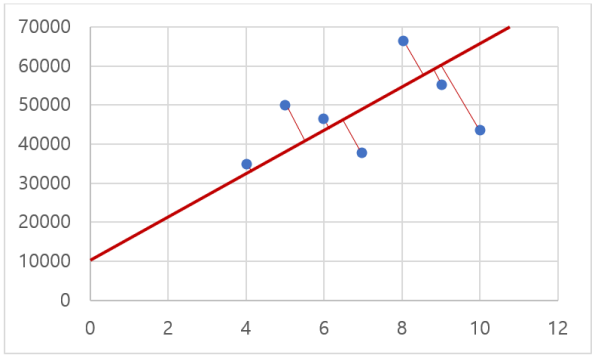


그림 4.1: (x축 월,y축) 점과 직선사이의 거리로 비용함수 계산

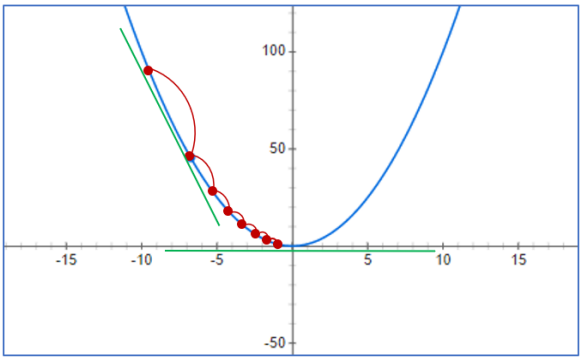


그림 4.2: 경사 하강 알고리즘

1 차 함수 기울기 값 W, Y 절편 값 b 로 비용 함수를 구하며, 비용이 적을수록 좋은 가설이다.

$$cost(W,b) = \frac{1}{m}\sum_{i=0}^{m}(H(x^{(i)}) - y^{(i)})^2 \qquad (3)$$

이제 경사 하강 알고리즘을 참고해서 합리적인 식을 도출할 수 있다. H(x) = Wx 로 식을 간단히 하고 비용 함수는 $(Wx - y)^2$ 를 따르게 되는데, 이를 그래프로 표현하면 그림 4.2 로, X 축을 W 값으로 두고 임의의 W 값을 주면 경사를 따라 특정한 간격으로 하강하게 된다. 여기서 미분을 통해 기울기가 최대한 0 인 수평이 되도록 합리적인 식을 찾는 것이 경사 하강 알고리즘이다.

또, 그림 4.2 의 점이 곡선 구조에 따라 경사를 타고 내려가게 되는데, 처음 이동하는 폭이 크고 점점 줄어드는 것을 볼 수 있다. 이는 이동의 폭이 낮으면 학습시간이 늘어나고, 너무 크면 부정확한 학습이 될 것이다.

이런 원리를 가지는 텐서플로우의 경사 하강 알고리즘을 통해서 예측 값을 구현할 수 있다.

4.3 텐서플로우 선형회귀 공공 자전거 수요 예측

그림 4.3 은 선형회귀 수요 예측 모델이 될 학습 데이터 X 범위를 2.2 절에서 선언한 by_month.index 를 불러와서 월 데이터를 monthly 에 저장하고, Y 범위도 마찬가지로 by_month.index 를 불러와서 월간 총 대여 수량을 rental 에 저장한다.

```
[110] import tensorflow as tf

      monthly = [by_month.index]# 자전거 대여 달
      rental = [by_month.values]# 달마다 총 대여량
      W = tf.Variable(tf.random_uniform([1], -100, 100))# 가중치 설정 100 단위
      b = tf.Variable(tf.random_uniform([1], -100, 100))# 렌덤 값으로 가중치를 조절 1차원 배열에 삽입
      X = tf.placeholder(tf.float32)
      Y = tf.placeholder(tf.float32)
      H = W * X + b # 가설식
      cost = tf.reduce_mean(tf.square(H - Y))# 비용 함수
      a = tf.Variable(0.01)# 경사하강 설정
      optimizer = tf.train.GradientDescentOptimizer(a)# 텐서플로우 경사하강 라이브러리
      train = optimizer.minimize(cost)
      init = tf.global_variables_initializer()
      sess = tf.Session()
      sess.run(init)


      for i in range(10001): #
          sess.run(train, feed_dict={X: monthly, Y: rental})
          if i% 500 == 0: # 500개 마다 모니터링
              print (i, sess.run(cost, feed_dict={X: monthly, Y: rental}), sess.run(W), sess.run(b))

      print (sess.run(H, feed_dict={X: [8]}))# 다음달 수요 예측
```

그림 4.3: 가공된 데이터와 텐서플로우를 활용한 수요 예측 소스 코드

텐서플로우에서 제공하는 변수 W, b 를 4.1 절에서 소개했는데, np.random.uniform 균등분포로부터 무작위 표본 추출 방식을 이용한다. 이를 1 차원 배열에 삽입하고 -100 부터 100 사이의 변수가 입력된다. X 와 Y 에 각각 데이터를 저장하는 placeholder 를 정의한다.

H = W * X + b 로 가설식을 선언한다.

비용 함수 cost 의 square 는 제곱을 의미하며, 예측 값 H 에 실제 값 Y 를 뺀 뒤, 남은 값을 제곱한 것으로 reduce_mean 을 이용해서 평균 값을 구할 수 있다.

비용 함수를 백분율 값으로 0.01 로 경사 하강 값을 주었다.

텐서플로우 학습과 관련된 경사 하강 라이브러리 tf.train.GaradientDescentOptimizer 을 로드 한다.

비용 함수를 적게 만들도록 train 에 minimize(cost)를 선언하고 .init 으로 tf.globa_variables_initializer 로 변수 초기화 한 뒤, 만든 텐서플로우 객체에서 세션을 sess 로드하고, 변수 초기화한다.

이제 반복문을 통해 10001 을 반복하게 하고 X 와 Y 에 데이터를 매칭시킨 뒤, if 문으로 500 번에 한번씩, 학습의 횟수와 함께 가중치 값(기울기 값)을 보여주도록 출력을 설정했다.

마지막으로, X 의 값에 따라 예측 값을 보여줄 수 있도록 설정했다.

4.4 수요 예측 결과

그림 4.4 에 텐서플로우 학습결과 결과적으로 예측된 데이터가 출력되었다.

앞서 본문 4.2 절에서 텐서플로우 선형회귀 경사 하강 알고리즘에 따라, 학습을 처음 시작할 때는 데이터의 변동 폭이 큰 반면, 그 폭이 줄어들다가 값이 중반에 들어서 고정되었다.

그리고 결과 값은 학습 중 합리적인 답에 가까워졌기 때문에 더 이상 크게 변화하지 않고 고정된 값 그대로 10000 번의 학습 끝에 가장 나은 데이터라고 출력했다.

표본 데이터의 양은 두개 이상일 때 예측이 가능한 점을 고려해본다면 사용한 표본 데이터는 그리 크지 않지만 예측 값은 어느정도 합리적으로 보인다.

```
0 173905400.0 [6853.149] [915.45184]
500 98657990.0 [4103.149] [17921.75]
1000 84618880.0 [3028.3728] [26047.953]
1500 81437810.0 [2516.7654] [29916.129]
2000 80717010.0 [2273.2358] [31757.414]
2500 80553690.0 [2157.3123] [32633.896]
3000 80516690.0 [2102.1316] [33051.105]
3500 80508290.0 [2075.8643] [33249.71]
4000 80506410.0 [2063.3613] [33344.242]
4500 80505960.0 [2057.41] [33389.242]
5000 80505864.0 [2054.5767] [33410.66]
5500 80505850.0 [2053.223] [33420.9]
6000 80505840.0 [2052.5874] [33425.703]
6500 80505840.0 [2052.296] [33427.906]
7000 80505860.0 [2052.1694] [33428.86]
7500 80505860.0 [2052.1694] [33428.86]
8000 80505860.0 [2052.1694] [33428.86]
8500 80505860.0 [2052.1694] [33428.86]
9000 80505860.0 [2052.1694] [33428.86]
9500 80505860.0 [2052.1694] [33428.86]
10000 80505860.0 [2052.1694] [33428.86]
[49846.215]
```

그림 4.4: 가공된 데이터와 텐서플로우를 활용한 수요 예측 결과 (출력 결과 순서: [학습횟수] [Cost] [W] [b])

결과 데이터에 따르면 수요는 다음달 11 월은 49846.215 로 예측되며 그림 4.1 에서 상반기 수요 분포 증가한 것을 볼 수 있다. 또한, 선형 회귀 식의 기울기(W)와 Y 절편(b)는 각각 2052.1694 와 33428.86 으로 계산되었다.

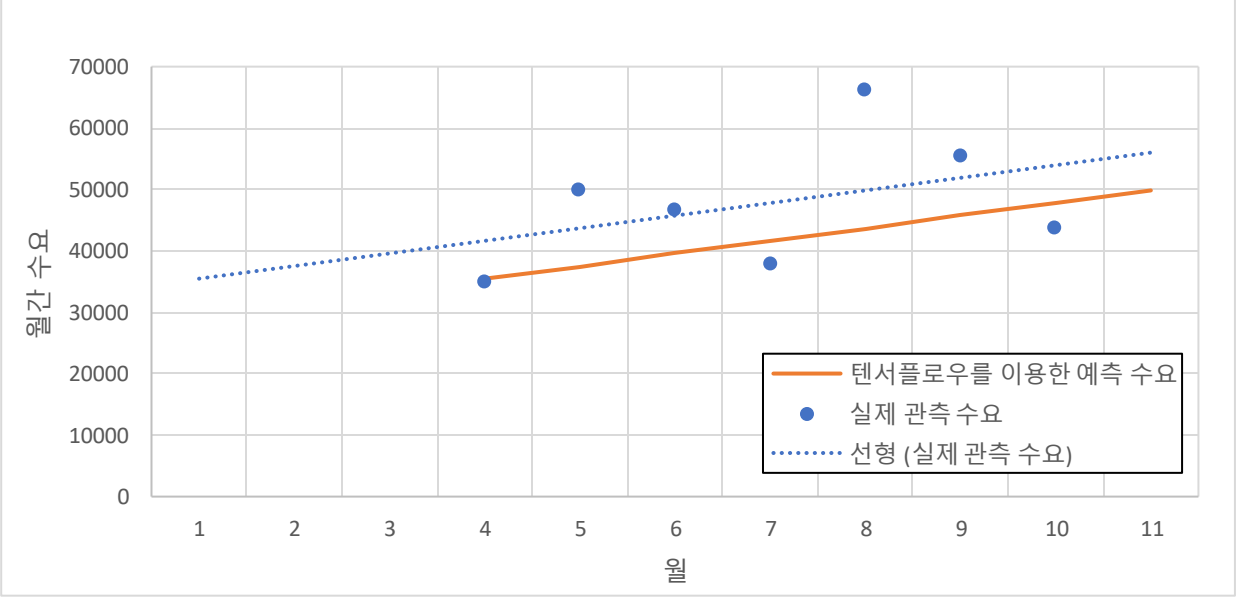


그림 4.5: 실제 관측데이터와 텐서플로우 선형회귀 경사 하강 알고리즘을 이용한 수요 예측 비교

그림 4.5는 그림 4.4에서 계산된 선형 회귀 식을 시각화 한 것이다. 파란 점은 실제 수요 분포를 나타내고 주황색 실선은 수요 예측을 나타낸 것이며, 파란색 점선은 마이크로소프트 엑셀의 추세선 기능을 활용하여 계산한 수요예측이다.

마이크로소프트 엑셀의 추세선과 텐서플로우를 이용한 결과를 비교하면, 기울기는 비슷하지만 Y 절편에 차이가 있는 것으로 결과가 출력되었다. Y 절편이 다른 것은 경사 하강 알고리즘이 추가되었기 때문인 것으로 생각되며, 경사 하강 백분율 값을 변화하면 예측 수요 그래프도 바뀔 것으로 생각된다.

그러나, 기계학습을 통한 수요 예측은 학습 데이터인 월별 수요량의 데이터 질에 따라 얼마든지 달라질 수 있다는 점을 고려해야 한다. 예를 들면, 날씨, 신규 가입자 수 및 현지 사정이 데이터에 반영 될 수 있다면 좀 더 정확한 수요 그래프를 얻을 수 있을 것이다.

이번 보유한 데이터에 한해서 이루어진 월간 수요 예측 결과는 다음달의 수요를 예측하는 데 있어서 매달 증가하는 추세에 따라 합리적인 증가폭을 보여주기 때문에, 이 결과는 신뢰할 수 있는 근사 및 예측 값으로 볼 수 있다.

제 5 장 결론

2.2 절에서 CSV 데이터 train 을 로드해서 Datetime 을 이용한 기존 데이터의 시간과 수요를 구체적으로 분류해서 DataFrame 으로 표현했고, year, month, day, hour 로 분류된 데이터를 Group 한 뒤, 만들어진 Series 의 value 와 index 를 중심으로 seaborn 의 barplot, pointplot 을 이용해 데이터를 year, month, day, hour 의 총 이용 수량으로 시각화 했다.

전체적인 총 이용 수량은 트론헤임 지역의 거주하는 공공 자전거 사용자들의 생활 패턴을 파악할 수 있었고, 시각화 된 그래프는 전체적으로 수요 양을 확인하는데 지장은 없지만 구체적인 데이터가 부족해 3.2 절에서는 다른 그래프 형식을 사용했다.

3.2 절에서는 기존 데이터를 조금 더 구체적으로 많은 양의 데이터를 처리하기 위해 제어문인 조건문과 반복문을 활용해서 데이터를 분류했다.

train 데이터에서 index 값의 column 을 선택해서 .isin 으로 해당 month 을 선택해 분류하거나, 다른 column 을 선택해서 필터하고 정렬하는 작업을 했다.

결과적으로 해당 month 마다 데이터를 분류해서 표기할 수 있었고, 이 데이터를 반복문과 제어문을 이용해서 2.2 절과 다른 그래프로 표현했다. 매달 마다 수요가 가장 많은 정거장을 순서대로 정렬하고 그래프 바에 시작점과 도착점, 이용 수량을 표시해서 가독성 있게 표현했다.

출력 결과는 만족하지만, 데이터를 하나하나 출력하는 방식 보다는 input 으로 값을 입력 받아서 출력하는 프로그래밍이 필요하다.

3.3 절에서 다루었던 가장 수요가 많은 시작, 도착 지점은 데이터를 분류하는 과정에서 하나의 데이터 프레임에 각 정거장과 월 별 분류를 합치는 과정에 데이터 프레임을 합병하는 pd.concat 을 이용하려 했지만, 반복문을 지원하지 않는 형식과, DataFrame 내에서 파이썬의 새로운 반복문 .format 이나 f - string 방식도 적용되지 않는 문제로 가장 고전적인 방법인 직접 입력하는 방식 데이터를 분류했다. 범위는 매크로를 이용해 만들었지만, 구현하지 못한 부분에 대한 개선이 필요하다.

4.2 절의 텐서플로우 라이브러리를 활용한 수요 예측은 사용한 공공 자전거 월간 총 이용 수량 데이터를 이용하는 데 있어서 예제 모델과 다소 다른 형태로 데이터 분석에 알맞지 않다고 생각되는데 학습 범위와 경사 하강 값을 조절해가면서 큰 폭에서 점차 줄어드는 이동 거리를 확인할 수 있었고, 안정화되면서 납득가능한 값을 출력했다.

이를 통해 데이터의 양과 질에 따라서 보다 정확하게 예측할 수 있다는 것을 확인했다.

참고 문헌

[1] Trondheim City bike[Website]. (2019.11.01). URL: https://trondheimbysykkel.no/en/open-data/historical

[2] R, Python 분석과 프로그래밍의 친구 by R Friend[Website]. (2019.10.05). URL: https://rfriend.tistory.com/384/

[3] 데이터 사이언스 스쿨[Website]. (2019.10.06). URL: https://datascienceschool.net/view-notebook/4c2d5ff1caab4b21a708cc662137bc65/

[4] Seaborn: statistical data visualization[Website]. (2019.10.04). URL: http://seaborn.pydata.org/generated/seaborn.barplot.html?highlight=barplot#seaborn.barplot

[5] Kaggle: Your Home for Data Science[Website]. (2019.10.12). URL: https://www.kaggle.com/kwonyoung234/for-beginner/

[6] Basic date and time types : The Python Standard Library[Website]. (2019.10.15). URL: https://docs.python.org/3/library/datetime.html

[7] Google Developers[Website]. (2019.11.02). URL: https://developers.google.com/machine-learning/crash-course/descending-into-ml/linear-regression

[8] 텐서플로우(TensorFlow)에서 선형 회귀의 비용(Cost)[Website]. (2019.11.03). URL: http://blog.naver.com/PostView.nhn?blogId=ndb796&logNo=221277745954&parentCategoryNo=&categoryNo=83&viewDate=&isShowPopularPosts=false&from=postView