충전 패턴 클러스터링을 통한 전기차 충전기 분류

이영웅(전남대학교)
165723@jnu.ac.kr

요 약

본문은 전기차 충전기의 각 시각당 평균 충전량을 구해 하루동안의 충전패턴을 구한다. 구한 충전 패턴을 클러스터링 기법을 통해 분류한다.

Ⅰ. 서 론

세계적으로 친환경 에너지에 관심을 가지고 속도가 전기자동차 사용량이 늘고 있다.[1] 이는 대한민국에서도 마찬가지이고 그에 따라 전기차 충전소의 개수도 증가하고 있다. 23.12.15기준 한전에서 설치한 충전소가 4668개소, 충전기가 10674개이다. 이 충전소는 위치에 따라 공용, 아파트용, 업무용으로 나뉘고 충전 속도에 따라 급속, 완속으로 나뉜다.[2] 충전소 위치에 따라 사용량을 분석한 결과 계절, 지역에 따라서는 큰 차이점이 없다. 시간에 따라서는 차이점을 보였는데 공용, 업무용은 시간대에 따른 사용량이 비슷하고 아파트용은 상이하다.[3]. 공용과 업무용은 낮에 사용량이 많았으며, 아파트용은 밤과 새벽에 사용량이 많았다. 본문에서는 위 조사와는 반대로 시간에 따른 사용량을 통해 각 충전기의 패턴을 분석한 다음 클러스터링 기법으로 충전기를 분류하고자 한다.

Ⅱ. 본론

본문은 경기지역 경제포털의 "전기차 충전이력 - 2023년 1분기" 데이터를 활용하였다. 데이터의 내용은 충전 일시, 충전소 ID, 충전기 ID, 시도 코드, 시 시군구 코드, 시도 명, 시군구 명, 충전소 명, 위도, 경도, 충전 구분, 충전 방식, 시작 일시, 종료 일시, 시작 SOC, 종료 SOC, SOC 충전량, 전력 사용량, 충전 시간, 전력 사용량(경부하), 전력 사용량(중간부하), 전력 사용량(최대부하), 비고, 생산일시이다. 이 중 충전기ID, 시작일시, 종료 일시, 전력 사용량을써서 클러스터링 기법을 활용해 시간별 사용량에 따른 충전기 분류를 한다. 지역과 날짜만 있는 데이터는 사용하지 않는다. 계절, 지역에 따라서는 충전 패턴의 차이가 별로 없기 때문이다.[3] 충전량 데이터 중 전력 사용량 데이터만 활용한다. 충전 속도가 충전 시간동안 항상 일정하다고 가정하고 데이터를 전처리했다.

1시 시간대의 전력사용량을 1시 부터 2시 사이에 전력 사용량이라 하자. 시작 일시, 종료 일시의 날짜 값 중 시간데이터만 사용하면 각 시간대의 전력 사용량을 구할 수 있다. 충전 속도가 일정하다면 각 시간대의 충전량은 다음 식을 만족한다.

$$Charge_{thishour} = Charge \times \left(\frac{t_{thishour}}{t_{total}} \right). \qquad (1)$$

여기서 $Charge_{thishour}$는 각 시간대의 충전량, $Charge$는 전력 사용량, $t_{thishour}$는 이 시간대에서 충전한 시간, t_{total}은 총 충전시간이다.

이 값을 각 충전기 별 시간대 별로 더한뒤 데이터를 입력한 총 날 수로 나누면, 각 충전기의 하루 평균 시간대 별 충전량을 알 수 있다. 0시, 1시, ... 23시 까지의 총 24차원의 데이터가 되며 이를 K-means clustering을 활용하면 비슷한 유형의 충전기를 분류할 수 있다.

클러스터링 결과는 다음과 같다. 기존의 설치 위치에 따른 분류 방법과 같이 3가지로 분류 하였으며 편의상 초록색, 파란색, 빨간색으로 정의하겠다.

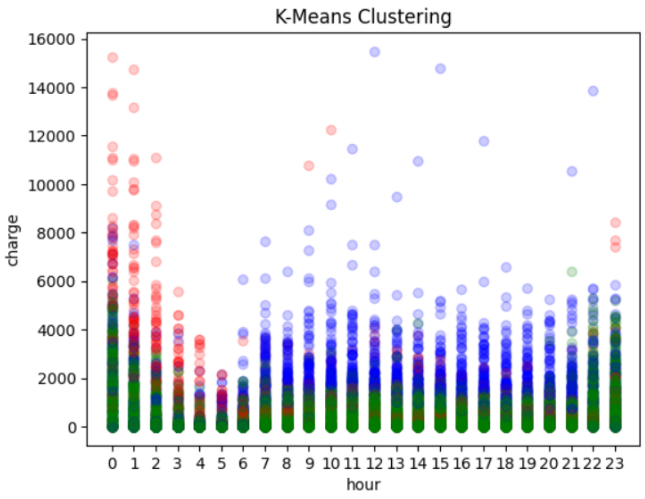

그림 1. 클러스터링 결과 종합

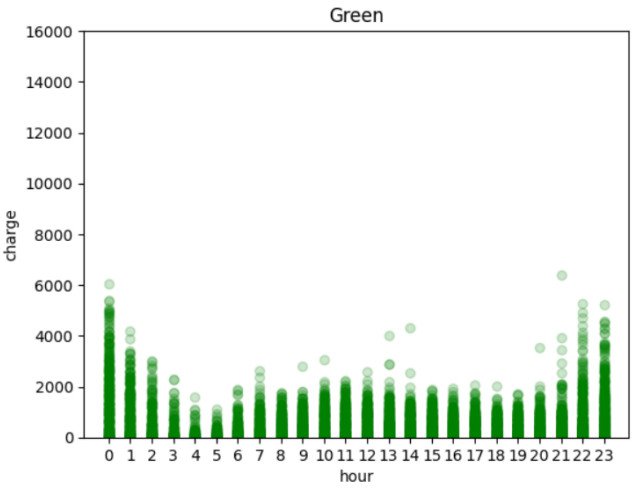

그림 2. 초록색 분류 충전기의 시간대별 충전량

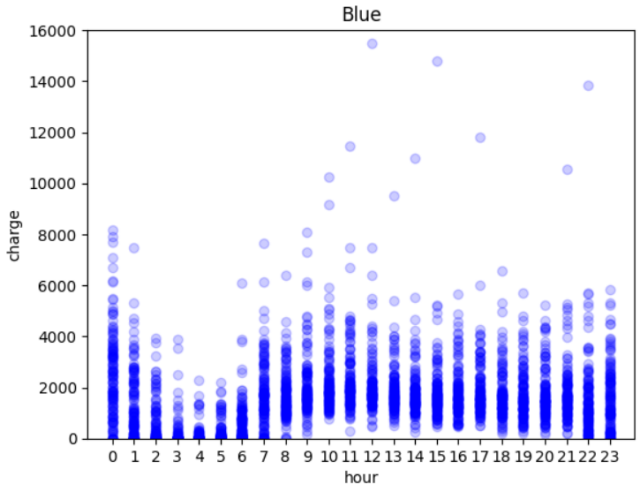

그림 3. 파란색 분류 충전기의 시간대별 충전량

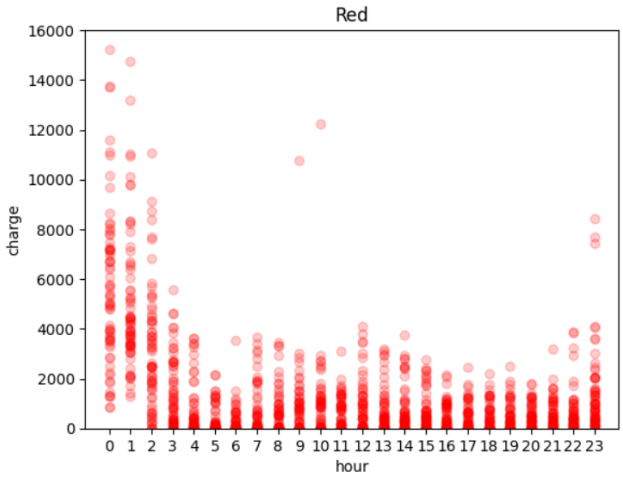

그림 4. 빨간색 분류 충전기의 시간대별 충전량

결과를 살펴보면 초록색은 전체적으로 사용량이 적은 것을 알 수 있고, 파란색은 초록색보다 전체적으로 사용량이 많으면서 새벽을 제외한 시간대에 사용량이 고르게 퍼져있다. 빨간색은 새벽에 사용량이 집중되었다. 평균치를 구해서 그래프를 나타내보겠다.

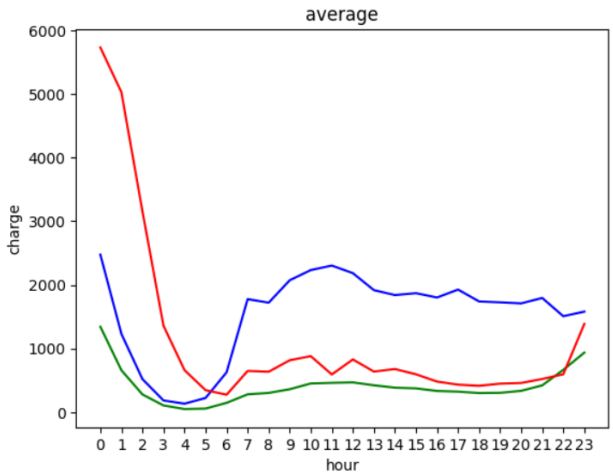

그림 5. 각 분류의 시간대별 충전량 평균치 비교

마지막으로 실제로 비슷한 특성을 가진 값들을 분류했는지 확인하기 위해 차원 축소를 통한 주성분 분석, PCA를 한 뒤 그래프로 나타낸다. 총 24차원의 시간대 별 데이터를 2차원으로 축소하였다.

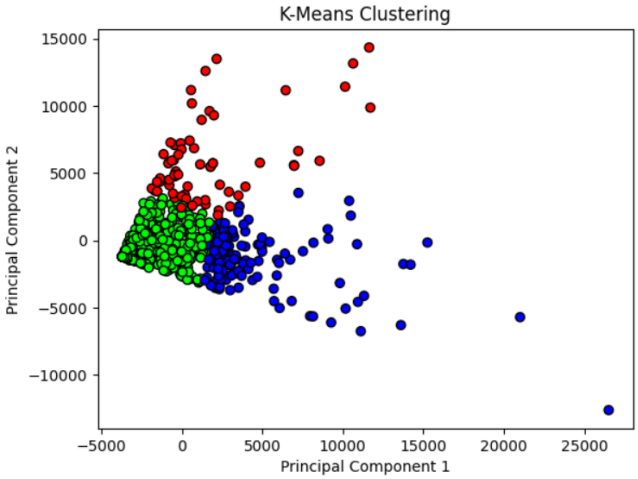

그림 6. 주성분 분석 결과

Principal Component1, Principal Component2가 무엇인지 명확하게 말할 수는 없지만 각 분류가 서로 겹치지 않고 특색이 있는것을 알 수 있다.

파란색은 기존의 공용, 업무용과, 빨간색은 아파트용과 사용패턴이 비슷하다고 할 수 있다. 하지만 완전이 똑같은 분류라고 말 할수는 없다. 파란색은 공용, 아파트용에 비해 사용량이 낮, 밤 시간대의 사용량의 편차가 작다. 빨간색은 아파트용에 비해 낮 시간대의 사용량이 현저히 적다. 초록색은 새로운 패턴으로 파란색, 빨간색에 비해 전체적으로 사용량이 적다.

Ⅲ. 결론

본문에서는 각 충전기의 시간대별 충전량을 분석해 충전기를 분류하였다.
결과를 통해 기존의 위치 기반, 충전 속도가 아닌 충전 패턴으로 충전기를 분류 할 수 있음을 확인하였다. 이 분류 방법으로 각 분류의 충전기의 위치, 충전 속도를 분석해서 앞으로 충전기의 설치 위치, 충전기 설치 개수, 충전기 충전 속도 등을 결정할 수 있다고 생각한다.

참 고 문 헌

[1] SNE 리서치 "2023 글로벌 전기차 인도량 1,377만대, 전년대비 30.6%로 성장 둔화 전망"
https://www.sneresearch.com/kr/insight/release_view/190/page/0
[2] 한국 전력 전기차충전서비스 통계정보 충전기 현황
https://evc.kepco.co.kr:4445/service/service04.do
[3] 김준혁, 문상근, 이병성, 서인진, 김철환 "실데이터 기반의 전기자동차 충전 데이터 분석 및 패턴 도출", 전기학회논문지 67권 11호

APPENDIX

Matlab/Python Source Code를 TEXT 형태로 이곳에 추가해 주세요. 주의: 그림 파일 아님. <글씨크기 9 Point>

<Python 코드> (python의 jupyter notebook 활용)

[1]
```
# 필요한 모듈 불러오기
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from matplotlib.colors import LinearSegmentedColormap
```

[2]
```
# 데이터 스키마
data_schema = pd.read_csv('./TC_ELCTY_ATMBL_ELCTC_RECD_20230609095252_schema.csv', encoding='euc-kr')
data_schema
```

[3]
```
# 데이터 목록 가져오기
path = "./TC_ELCTY_ATMBL_ELCTC_RECD_20230609095252/"
file_list = os.listdir(path)
```

[4]
```
# 데이터 가져오기
data = []
for file_name in file_list:
    file_path = path + file_name
    data.append(pd.read_csv(file_path)[['충전기_ID', '시작_일시', '종료_일시', '전력_사용량']])
data = pd.concat(data, ignore_index=True)
```

[5]
```
# 가져온 데이터 확인
data.info()
```

[6]
```
# 첫 데이터로 데이터 형식 확인
data.head(1)
```

[7]
```
# 날짜 데이터에서 시분초 데이터만 추출해서 추가
data['시작_시간'] = (data['시작_일시'].str[-8:-6] + data['시작_일시'].str[-5:-3] + data['시작_일시'].str[-2:])
data['종료_시간'] = (data['종료_일시'].str[-8:-6] + data['종료_일시'].str[-5:-3] + data['종료_일시'].str[-2:])
```

[8]
```
# 날짜 데이터 삭제
data = data.drop(['시작_일시', '종료_일시'], axis=1)
```

[9]
```
# 바꾼 데이터 확인
data.head(1)
```

[10]
```
# 충전기 id를 사전순으로 정렬한 것
ids = sorted(data['충전기_ID'].unique())
```

[11]
```
# 충전량 데이터
# 가로축 충전소 ID, 세로축 시각
id_time_charge = {}
for i in ids:
    id_time_charge[i] = np.zeros(24)
```

[12]
```
# '030000'을 정수형으로 바꾸면 030000이 아니라 30000이 되므로 뒤에서부터 인덱싱 한다
def get_h(t):
    return int(t[:-4])
def get_m(t):
    return int(t[-4:-2])
def get_s(t):
    return int(t[-2:])
```

[13]
```
# 문자열 시간 데이터에서 시, 분, 초를 정수형으로 반환하는 함수
def get_hms(t):
    # 시, 분, 초
    return get_h(t), get_m(t), get_s(t)
```

[14]
```
# 초단위 시간
def modulo_ds(ds):
    # 하루는 86400초이다
    # 24*60*60 = 86400
    return ds%86400
```

[15]
```
# 총시간을 계산
def cal_total_s(h,m,s):
    return modulo_ds(3600*h+60*m+s)
```

[16]
```
# 두개의 문자열 시간 데이터의 차이를 초 단위로 반환하는 함수
def cal_ds(s,e):
    # 시분초 구하기
    sh, sm, ss = get_hms(s)
    eh, em, es = get_hms(e)
```

```
    # 시분초 각각의 차이 계산
    dh = eh-sh
    dm = em-sm
    ds = es-ss
    return cal_total_s(dh,dm,ds)
```

[17]
```
# 각각의 충전기의 시간별 데이터 입력
for i in range(len(data)):
    d = data.iloc[i] # 1개의 데이터 가져오기

    id = d['충전기_ID'] # ID
    s = d['시작_시간'] # start time
    e = d['종료_시간'] # end time
    c = d['전력_사용량'] # charge
    # 충전량이 0이면 이상치이므로 버린다
    if c == 0:
        continue
    # 문자열 시간데이터 정수형으로 바꾸기
    sh = get_h(s)
    eh = get_h(e)
    # 총 충전시간 구하기
    ds = cal_ds(s,e)
    # 충전시간이 0이하이면 이상치 이므로 버린다
    if ds <= 0:
        continue
    # 정각시간문자열
    # 정각시간을 문자열로 구하기 위해 정의
    o = '0000'
    # 각 시간대의 충전량을 구하기
    # 소수점 아래는 버린다

    # 시작 시간과 다음 정각과의 시간차를 구한 뒤
    # 충전 시간과의 비율을 구해 전력 사용량과 곱해
    # 시간대의 충전량을 구하기
    id_time_charge[id][sh] += cal_ds(s, str(sh+1)+o)*c//ds
    # 시작 시간, 종료 시간이 아니라면 1시간 내내 충전하므로
    # 시간차를 구하지 않고 3600으로 계산한다
    if sh > eh:
        eh+=24
    for j in range(sh+1, eh):
        id_time_charge[id][j%24] += 3600*c//ds
    # 끝 시간과 그 전 정각과의 시간차를 구한 뒤
    # 충전 시간과의 비율을 구해 전력 사용량과 곱해
    # 시간대의 충전량을 구하기
    id_time_charge[id][eh%24] += cal_ds(str(eh)+o, e)*c//ds
```

[18]
```
# 2023년 1월 1일부터 3월 31일 까지 데이터 이므로
# 하루당 데이터를 얻으려면 날수로 나눠야한다
# 1월 : 31일
# 2월 : 28일
# 3월 : 31일
# 총  : 90일
for i in ids:
    id_time_charge[i] //= 90
# 데이터 기록 후 데이터프레임으로 변환
# 충전량 데이터
# 가로축 시각, 세로축 충전소 ID
id_time_charge_df = pd.DataFrame(id_time_charge).T
```

[19]
```
# 데이터 전처리 확인
id_time_charge_df
```

[20]
```
# 전처리는 오래걸리기 때문에
# 결과 데이터프레임을 파일로 저장
id_time_charge_df.to_csv('./pretreatment_data.csv')
# 이후 클러스터링은 여기 아래서부터 전처리한 파일로 하면 된다.
```

[21]
```
# 필요한 모듈 불러오기
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from matplotlib.colors import LinearSegmentedColormap
```

[22]
```
id_time_charge_df = pd.read_csv('./pretreatment_data.csv', index_col=0)
id_time_charge_df
```

[23]
```
# K-Means 클러스터링
n_clusters = 3  # 클러스터 개수 설정
kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
kmeans.fit(id_time_charge_df)
# 클러스터링 결과
labels = kmeans.labels_
```

[24]
```
# 클러스터링 결과 시각화
colors = ['g', 'b', 'r']
for i in range(len(id_time_charge_df)):
    plt.scatter(id_time_charge_df.columns, id_time_charge_df.iloc[i], c=colors[labels[i]], alpha=0.2)
plt.title('K-Means Clustering')
plt.xlabel('hour')
plt.ylabel('charge')
plt.show()
```

[25]

```
# 초록색 분류
plt.ylim(0,16000)
for i in range(len(id_time_charge_df)):
    if labels[i] != 0:
        continue
    plt.scatter(id_time_charge_df.columns, id_time_charge_df.iloc[i], c=colors[labels[i]], alpha=0.2)
plt.title('Green')
plt.xlabel('hour')
plt.ylabel('charge')
plt.show()
```

```
[26]
# 파란색
plt.ylim(0,16000)
for i in range(len(id_time_charge_df)):
    if labels[i] != 1:
        continue
    plt.scatter(id_time_charge_df.columns, id_time_charge_df.iloc[i], c=colors[labels[i]], alpha=0.2)
plt.title('Blue')
plt.xlabel('hour')
plt.ylabel('charge')
plt.show()
```

```
[27]
# 빨간색
plt.ylim(0,16000)
for i in range(len(id_time_charge_df)):
    if labels[i] != 2:
        continue
    plt.scatter(id_time_charge_df.columns, id_time_charge_df.iloc[i], c=colors[labels[i]], alpha=0.2)
plt.title('Red')
plt.xlabel('hour')
plt.ylabel('charge')
plt.show()
```

```
[28]
# 각 색의 평균 시각화
# 각 색의 개수
g, b, r = np.sum(labels == 0), np.sum(labels == 1), np.sum(labels == 2)
# 평균값을 계산할 빈 배열 생성
green = [0 for i in range(24)]
blue = [0 for i in range(24)]
red = [0 for i in range(24)]
for i in range(len(id_time_charge_df)):
    # 초록색 합
    if labels[i] == 0:
        green += id_time_charge_df.iloc[i]
    # 파란색 합
    elif labels[i] == 1:
        blue += id_time_charge_df.iloc[i]
    # 빨간색 합
    elif labels[i] == 2:
        red += id_time_charge_df.iloc[i]
# 평균
green /= g
blue /= b
red /= r
# 꺾은선 그래프
plt.plot(id_time_charge_df.columns, green, c='g')
plt.plot(id_time_charge_df.columns, blue, c='b')
plt.plot(id_time_charge_df.columns, red, c='r')
plt.title('average')
plt.xlabel('hour')
plt.ylabel('charge')
plt.show()
```

```
[29]
# 커스텀 cmap. 같은 색을 내는 cmap이 없어서 만들었다.
colors = ['#00FF00', '#0000FF', '#FF0000']
custom_cmap = LinearSegmentedColormap.from_list('my_cmap', colors)
```

```
[30]
# 2차원으로 시각화하기 위해 PCA를 사용하여 2차원으로 축소
pca = PCA(n_components=2)
id_time_charge_df_pca = pca.fit_transform(id_time_charge_df)
# 클러스터링 결과 시각화
plt.scatter(id_time_charge_df_pca[:, 0], id_time_charge_df_pca[:, 1], c=labels, cmap=custom_cmap, edgecolor='k') # , cmap='Spectral'
plt.title('K-Means Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```