머신 러닝
레포트

김호권

목 차

1. 크롤링

1-1 크롤링이란

웹 페이지에서 데이터를 수집하는 프로세스를 말한다.
크롤링의 순서로는 요청, 분석, 추출, 저장 순으로 진행되며 여러 페이지를 수집해야할 경우에는 이 순서를 반복한다.

밑에서는 두 가지의 방법으로 네이버 API와 공공데이터를 이용한 크롤링 작업을 진행한다.

1-2 네이버 API를 이용한 크롤링

```python
#[CODE 1]
def getRequestUrl(url):
    req = urllib.request.Request(url)
    req.add_header("X-Naver-Client-Id", client_id)
    req.add_header("X-Naver-Client-Secret", client_secret)

    try:
        response = urllib.request.urlopen(req)
        if response.getcode() == 200:
            print ("[%s] Url Request Success" % datetime.datetime.now())
            return response.read().decode('utf-8')
    except Exception as e:
        print(e)
        print("[%s] Error for URL : %s" % (datetime.datetime.now(), url))
        return None
```

code 1

1. request 모듈을 임포트 한 뒤에
헤더에 naver api에 필요한 클라이언트 id, secret를 추가.

2. http 요청을 수행하고 응답 받아오라고 한 뒤에
만약에 응답코드가 200인 경우에는 성공했다는 'Url Request Success'라는 메시지를 현재 시간을 합해서 출력.
그 뒤 응답 내용을 utf-8로 디코딩한 뒤에 반환.

3. 만약에 예외가 발생하였을 경우 에러 메시지인 'Error for URL'이라는 메시지를 현재 오류 발생한 시간과 합해서 출력하고 None을 반환.

4. 위의 try에서 예외가 발생했으면 예외 메시지를 출력.

```python
#[CODE 2]
def getNaverSearch(node, srcText, start, display):
    base = "https://openapi.naver.com/v1/search"
    node = "/%s.json" % node
    parameters = "?query=%s&start=%s&display=%s" % (urllib.parse.quote(srcText), start, display)

    url = base + node + parameters
    responseDecode = getRequestUrl(url)   #[CODE 1]

    if (responseDecode == None):
        return None
    else:
        return json.loads(responseDecode)
```

code 2

1. base로 네이버 open api 검색 주소를 주고 검색할 노드와 검색어를 가져옴.

2. base와 node, parameters의 정보를 합해서 url을 생성하고 getRequestUrl을 이용해서 네이버 API에 요청 수행.

3. 만약에 응답이 없으면 None,
응답을 할 경우 json형식으로 변환한 뒤에 반환.

```python
#[CODE 3]
def getPostData(post, jsonResult, cnt):
    title = post['title']
    description = post['description']
    org_link = post['originallink']
    link = post['link']

    pDate = datetime.datetime.strptime(post['pubDate'],  '%a, %d %b %Y %H:%M:%S +0900')
    pDate = pDate.strftime('%Y-%m-%d %H:%M:%S')

    jsonResult.append({'cnt':cnt, 'title':title, 'description': description,
'org_link':org_link,    'link': link,    'pDate':pDate})
    return
```

code 3

1. post 데이터에서 title, description, org_link, link에 해당하는 값을 추출 한 뒤 변수에 저장.

2. pubDate의 값을 추출한 뒤에 ('%a, %d %b %Y %H:%M:%S +0900')의 형식으로 변환하여 pDate에 저장.

3. 앞에서 추출했던 정보들을 기반으로 jsonResult에 포함.

```python
#[CODE 0]
def main():
    node = 'news'   # 크롤링 할 대상
    srcText = input('검색어를 입력하세요: ')
    cnt = 0
    jsonResult = []

    jsonResponse = getNaverSearch(node, srcText, 1, 100)  #[CODE 2]
    total = jsonResponse['total']

    while ((jsonResponse != None) and (jsonResponse['display'] != 0)):
        for post in jsonResponse['items']:
            cnt += 1
            getPostData(post, jsonResult, cnt)  #[CODE 3]

        start = jsonResponse['start'] + jsonResponse['display']
        jsonResponse = getNaverSearch(node, srcText, start, 100)  #[CODE 2]

    print('전체 검색 : %d 건' %total)

    with open('%s_naver_%s.json' % (srcText, node), 'w', encoding='utf8') as outfile:
        jsonFile = json.dumps(jsonResult,  indent=4, sort_keys=True,  ensure_ascii=False)

        outfile.write(jsonFile)

    print("가져온 데이터 : %d 건" %(cnt))
    print ('%s_naver_%s.json SAVED' % (srcText, node))

if __name__ == '__main__':
    main()
```

code 0

1. node를 news로 설정한 뒤 사용자로부터 검색어를 받고 이 검색어를 네이버 검색 API를 통해 검색 정보를 가져옴.

2. 만약에 검색 결과가 있고 출력할 데이터가 있으면 반복할 수 있도록 while문을 사용하여 검색 결과를 getPostData 함수를 이용해서 정보를 추출하고 다음 검색 결과를 가져올 수 있도록 다시 검색을 시작.

3. 검색이 끝났으면 검색 결과 건수를 출력하고 json 파일로 저장한 뒤 결과를 출력.

- 견해

이 과정이 정상적으로 끝나면 검색결과가 지정된 폴더에 json파일로 저장이 되고 결과가 출력이 될 것이다.
수업의 일환으로 만인들이 사용하는 홈페이지인 네이버의 검색 결과를 크롤링 하는 방법을 사용하였다.
이 크롤링을 이용하면 다른 홈페이지에서도 네이버의 검색 결과를 가져오거나 할 수 있으나 그렇다면 네이버 홈페이지에 직접 가서 검색하는것과 무엇이 다른지 이해하지 못하였기에 새로 공공데이터 API를 기반으로 하는 크롤링을 한 번 더 하였다.

1-3 공공데이터 API를 이용한 크롤링

```
#[CODE 1]
def getRequestUrl(url):
    req = urllib.request.Request(url)
    try:
        response = urllib.request.urlopen(req)
        if response.getcode() == 200:
            print ("[%s] Url Request Success" % datetime.datetime.now())
            return response.read().decode('utf-8')
    except Exception as e:
        print(e)
        print("[%s] Error for URL : %s" % (datetime.datetime.now(), url))
        return None
```

code 1

1. request 모듈을 임포트 한 뒤에 url에 대한 정보를 추가.

2. http 요청을 수행하고 응답 받아오라고 한 뒤
만약에 응답코드가 200인 경우에는 성공했다는 'Url Request Success'라는 메시지를 현재 시간을 합해서 출력하고 그 뒤 응답 내용을 utf-8로 디코딩한 뒤에 반환.

3. 만약 예외가 발생하였을 경우 에러 메시지인 'Error for URL'이라는 메시지를 현재 오류 발생한 시간과 합해서 출력하고 None을 반환.

4. 위의 try에서 예외가 발생했으면 예외 메시지를 출력.

```
#[CODE 2]
def getTourismStatsItem(yyyymm, national_code, ed_cd):
    service_url = "http://openapi.tour.go.kr/openapi/service/EdrcntTourismStatsService/getEdrcntTourismStatsList"
    parameters = "?_type=json&serviceKey=" + ServiceKey  #인증키
    parameters += "&YM=" + yyyymm
    parameters += "&NAT_CD=" + national_code
    parameters += "&ED_CD=" + ed_cd
    url = service_url + parameters

    retData = getRequestUrl(url)  #[CODE 1]

    if (retData == None):
        return None
    else:
        return json.loads(retData)
```

code 2

1. 기본 url을 설정 한 뒤에 parameters를 설정.
'인증키'
'년도/월'
'국가정보'
'지역정보'
앞에서 설정한 정보들과 url을 합해서 전체 url을 생성.

2. 만약 응답이 없을 경우 None을 반환 응답이 있을 경우 json형식으로 변환한 후에 출력.

```python
#[CODE 3]
def getTourismStatsService(nat_cd, ed_cd, nStartYear, nEndYear):
    jsonResult = []
    result = []
    natName=''
    dataEND = "{0}{1:0>2}".format(str(nEndYear), str(12)) #데이터 끝 초기화
    isDataEnd = 0 #데이터 끝 확인용 flag 초기화

    for year in range(nStartYear, nEndYear+1):
        for month in range(1, 13):
            if(isDataEnd == 1): break #데이터 끝 flag 설정되어있으면 작업 중지.
            yyyymm = "{0}{1:0>2}".format(str(year), str(month))
            jsonData = getTourismStatsItem(yyyymm, nat_cd, ed_cd) #[CODE 2]

            if (jsonData['response']['header']['resultMsg'] == 'OK'):
                # 입력된 범위까지 수집하지 않았지만, 더이상 제공되는 데이터가 없는 마지막 항목인 경우 ----------
                if jsonData['response']['body']['items'] == '':
                    isDataEnd = 1 #데이터 끝 flag 설정
                    dataEND = "{0}{1:0>2}".format(str(year), str(month-1))
                    print("데이터 없음.... \n 제공되는 통계 데이터는 %s년 %s월까지입니다."
                          %(str(year), str(month-1)))
                    break
                #jsonData를 출력하여 확인.....................................................
                print (json.dumps(jsonData, indent=4,sort_keys=True, ensure_ascii=False))
                natName = jsonData['response']['body']['items']['item']['natKorNm']
                natName = natName.replace(' ', '')
                num = jsonData['response']['body']['items']['item']['num']
                ed = jsonData['response']['body']['items']['item']['ed']
                print('[ %s_%s : %s ]' %(natName, yyyymm, num))
                print('--------------------------------------------------------------------')
                jsonResult.append({'nat_name': natName, 'nat_cd': nat_cd,
                                   'yyyymm': yyyymm, 'visit_cnt': num})
                result.append([natName, nat_cd, yyyymm, num])

    return (jsonResult, result, natName, ed, dataEND)
```

code 3

1. 변수들을 저장할 리스트 함수를 만듦.

2. 입력한 연도 범위 안에서 검색을 반복하여 진행.
연도와 월을 이용해서 yyyy mm형식의 날짜 문자열을 만듦
{0} = 연도
{1:0>2} = 월을 2자리 수로 표현하고 오른쪽 정렬로 빈자리에는 0을 입력.

그 다음에 getTourismStatsItem 함수를 이용해서 통계 데이터를 가져옴.

3. 만약에 입력한 범위를 다 수집하지 않았어도 더 이상 나오는 데이터가 없을 경우 검색을 종료하며 (데이터 없음..) 이라는 메세지와 함께 검색한 범위의 데이터를 연 월을 표기하여 출력.

4. 가져온 데이터에서 필요한 정보들을 추출한 뒤에 결과를 출력하고 그 결과를 리스트에 추가.

```python
#[CODE 0]
def main():
    jsonResult = []
    result = []
    natName=''
    print("<< 국내 입국한 외국인의 통계 데이터를 수집합니다. >>")
    nat_cd = input('국가 코드를 입력하세요(중국: 112 / 일본: 130 / 미국: 275) : ')
    nStartYear =int(input('데이터를 몇 년부터 수집할까요? : '))
    nEndYear = int(input('데이터를 몇 년까지 수집할까요? : '))
    ed_cd = "E"     #E : 방한외래관광객, D : 해외 출국

    jsonResult, result, natName, ed, dataEND =getTourismStatsService(nat_cd,
                                    ed_cd, nStartYear, nEndYear) #[CODE 3]

    if (natName=='') : #URL 요청은 성공하였지만, 데이터 제공이 안된 경우
        print('데이터가 전달되지 않았습니다. 공공데이터포털의 서비스 상태를 확인하기 바랍니다.')
    else:
        #파일저장 1 : json 파일
        with open('./%s_%s_%d_%s.json' % (natName, ed, nStartYear, dataEND), 'w',
                  encoding='utf8') as outfile:
            jsonFile  = json.dumps(jsonResult, indent=4, sort_keys=True, ensure_ascii=False)
            outfile.write(jsonFile)
        #파일저장 2 : csv 파일
        columns = ["입국자국가", "국가코드", "입국연월", "입국자 수"]
        result_df = pd.DataFrame(result, columns = columns)
        result_df.to_csv('./%s_%s_%d_%s.csv' % (natName, ed, nStartYear, dataEND),
                  index=False, encoding='cp949')

if __name__ == '__main__':
    main()
```

code 0

1. 사용자에게서 국가코드, 연도를 입력받고 입력받은 정보를 사용하여 getTourismStatsService 함수를 이용하여 데이터를 만듦.

2. 만약 데이터 제공이 되지 않았을 경우에는 "데이터가 전달되지 않았습니다."라는 메시지를 출력.

3. 데이터 제공이 원할히 되었을 경우 json파일과 csv 파일로 저장.

- 견해

위 과정이 끝나면 지정된 폴더에 json과 csv파일이 저장되어 있을 것이다.
수업의 일환으로 공공데이터 포털에서 제공하는 데이터를 이용하여 크롤링 하는 방법을 사용 해보았다.
이걸 이용하면 옛 데이터를 엑셀 프로그램에서 한땀한땀 찾을 필요없이 정리 되어 있는 파일을 제공 받을 수 있기 때문에 효용성은 있다고 판단되나 엑셀 프로그램에서 검색하는 것은 검색어를 누르면 바로 볼 수 있는 것과는 달리 모든 조건을 다 입력 한 뒤에 결과 파일을 받아야 확인 할 수 있으므로 일장일단이 있다고 생각한다.

2. 스크래핑

2-1 스크래핑이란

스크래핑이란 특정 웹페이지에서 원하는 정보를 탐색하기 위해 웹퍼이지의 HTML을 분석하고 원하는 데이터를 추출하여 사용하는 프로세스를 말한다.

밑에서는 두 가지의 방법으로 Hollys커피의 데이터, WebDriver를 이용하며 진행한다.

2-2 Hollys.co.kr을 이용한 스크래핑

```
def hollys_store(result):
    for page in range(1,52):
        Hollys_url = 'https://www.hollys.co.kr/store/korea/korStore2.do?pageNo=%d&sido=&gugun=&store='
        print(Hollys_url)
        html = urllib.request.urlopen(Hollys_url)
        soupHollys = BeautifulSoup(html, 'html.parser')
        tag_tbody = soupHollys.find('tbody')
        for store in tag_tbody.find_all('tr'):
            if len(store) <=3:
                break
            store_td = store.find_all('td')
            store_name = store_td[1].string
            store_sido = store_td[0].string
            store_address = store_td[3].string
            store_phone = store_td[5].string
            result.append([store_name]+[store_sido]+[store_address]+[store_phone])
    return
```

code 1

1. hollys 커피 url에 접속하여 html을 가져옴.

2. 페이지는 1~51까지 순회.

3. html에서 tbody 찾고 tr 태그 안에 있는 매장 정보를 추출.

4. 추출한 매장 정보에서
이름(store_name)
시/도(store_sido)
주소(store_ address)
전화번호(store_phone)
4가지를 추출하여 result리스트에 추가.

```
def main():
    result = []
    print('Hollys store crawling >>>>>>>>>>>>>>>>>>>>>>')
    hollys_store(result)
    hollys_tbl = pd.DataFrame(result, columns=('store', 'sido_gu', 'address', 'phone'))
    hollys_tbl.to_csv('D:/Workspaces/python/hollys.csv', encoding='cp949', mode='w', index=True)
    del result[:]

if __name__ == '__main__':
    main()
```

	store	sido_gu	address	phone
0	부산사상광장점	부산 사상구	부산광역시 사상구 광장로 22 (괘법동) 2층 사상구 괘법동 565-2	051-322-4117
1	하남덕풍점	경기 하남시	경기도 하남시 덕풍북로 2 (덕풍동) .	070-4791-4731
2	대전오류점	대전 중구	대전광역시 중구 계룡로 876 메종드메디컬빌딩 104호, 202호 .	042-522-3141
3	계룡대점	충남 계룡시	충청남도 계룡시 신도안면 신도안1길 43 (계룡스포즈센터) 계룡스포즈센터 1층	042-551-5508
4	화정DI점	경기 고양시 덕양구	경기도 고양시 덕양구 화신로 254 (화정동)	031-971-4097
5	부산광복점	부산 중구	부산광역시 중구 광복로 70 (광복동2가) .	051-253-8033
6	양평북한강DI점	경기도 양평군	경기도 양평군 양서면 양수로118번길 29 칼리스	031-8079-7090
7	완주삼봉점	전북 완주군	전라북도 완주군 삼례읍 신기길 5-3 수계리 1212-4	063-261-2003
8	잠실새내역점	서울 송파구	서울특별시 송파구 올림픽로12길 8 (잠실동, 제일빌딩) .	02-416-2085
9	화성남양점	경기 화성시	경기도 화성시 남양읍2279-2 .	031-8067-7760
10	부산사상광장점	부산 사상구	부산광역시 사상구 광장로 22 (괘법동) 2층 사상구 괘법동 565-2	051-322-4117
11	하남덕풍점	경기 하남시	경기도 하남시 덕풍북로 2 (덕풍동) .	070-4791-4731
12	대전오류점	대전 중구	대전광역시 중구 계룡로 876 메종드메디컬빌딩 104호, 202호 .	042-522-3141
13	계룡대점	충남 계룡시	충청남도 계룡시 신도안면 신도안1길 43 (계룡스포즈센터) 계룡스포즈센터 1층	042-551-5508
14	화정DI점	경기 고양시 덕양구	경기도 고양시 덕양구 화신로 254 (화정동)	031-971-4097
15	부산광복점	부산 중구	부산광역시 중구 광복로 70 (광복동2가) .	051-253-8033
16	양평북한강DI점	경기 양평군	경기도 양평군 양서면 양수로118번길 29 칼리스	031-8079-7090
17	완주삼봉점	전북 완주군	전라북도 완주군 삼례읍 신기길 5-3 수계리 1212-4	063-261-2003
18	잠실새내역점	서울 송파구	서울특별시 송파구 올림픽로12길 8 (잠실동, 제일빌딩) .	02-416-2085
19	화성남양점	경기 화성시	경기도 화성시 남양읍2279-2 .	031-8067-7760
20	부산사상광장점	부산 사상구	부산광역시 사상구 광장로 22 (괘법동) 2층 사상구 괘법동 565-2	051-322-4117
21	하남덕풍점	경기 하남시	경기도 하남시 덕풍북로 2 (덕풍동) .	070-4791-4731
22	대전오류점	대전 중구	대전광역시 중구 계룡로 876 메종드메디컬빌딩 104호, 202호 .	042-522-3141
23	계룡대점	충남 계룡시	충청남도 계룡시 신도안면 신도안1길 43 (계룡스포즈센터) 계룡스포즈센터 1층	042-551-5508
24	화정DI점	경기 고양시 덕양구	경기도 고양시 덕양구 화신로 254 (화정동)	031-971-4097
25	부산광복점	부산 중구	부산광역시 중구 광복로 70 (광복동2가) .	051-253-8033
26	양평북한강DI점	경기 양평군	경기도 양평군 양서면 양수로118번길 29 칼리스	031-8079-7090
27	완주삼봉점	전북 완주군	전라북도 완주군 삼례읍 신기길 5-3 수계리 1212-4	063-261-2003
28	잠실새내역점	서울 송파구	서울특별시 송파구 올림픽로12길 8 (잠실동, 제일빌딩) .	02-416-2085

code 2

1. hollys_store(result)함수를 호출하여 매장 정보 수집.

2. 수집한 정보를 DataFrame으로 변환.

3. 변환한 DateFrame을 CSV파일로 CP949로 한글 인코딩 한 뒤에 hollys이란 파일명으로 저장.

4. CSV 파일을 열면 추출한 매장 정보인 store, sido_gu, address, phone를 열 이름으로 받아서 보여주는 것을 확인.

- 견해

수업의 일환으로 Hollys 커피에 있는 매장 정보 홈페이지를 스크래핑 하여 파일로 저장하였는데 스크랩이란 단어에 맞게 정보를 모조리 저장하는 그런 것으로 보였으나 내가 파일의 파라미터를 지정할 때 추출에서 제외할 수도 있을 수 있다는 생각과 결국은 이것도 크롤링이랑 다른바가 무엇인가 라는 생각이 듦.

2-3 Webdriver을 이용한 스크래핑

```python
#[CODE 1]
def MemodeCafe_store(result):
    MemodeCafe_URL = "https://www.mmthcoffee.com/sub/store.html"
    wd = webdriver.Chrome('./chromedriver.exe')

    wd.get(MemodeCafe_URL)
    time.sleep(1)  #웹페이지 연결할 동안 1초 대기

    wd.execute_script("goSearch()")
    time.sleep(1) #스크립트 실행 할 동안 1초 대기
    html = wd.page_source
    soupCB = BeautifulSoup(html, 'html.parser')
    store_info = soupCB.select("div.right > ul > li")

    for store_li in store_info:
        store_name = store_li.select("a div.txt_w div.tit strong")[0].string
        store_address = store_li.select("a div.txt_w div.txt p")[0].string.replace("주소 : ","")

        print(store_name,store_address)

        result.append([store_name]+[store_address])
    return
```

code 1

1. memode 커피 url에 접속하여 html을 가져와서 해당 url로 웹 브라우저를 엶.

2. 웹 페이지가 로딩되기를 1초간 기다림.

3. 페이지 내에서 'goSearch()를 이용.

4. HTML 소스코드를 가져와서 div.right > ul > li 순으로 매장 정보가 담긴 요소를 선택하고 정보를 순회.

5. 매장의 이름과 주소를 가져온 뒤 result 리스트에 추가.

```python
#[CODE 0]
def main():
    result = []
    print('MemodCafe store crawling >>>>>>>>>>>>>>>>>>>>>>>>>')
    MemodeCafe_store(result)  #[CODE 1]

    CB_tbl = pd.DataFrame(result, columns=('store', 'address'))
    CB_tbl.to_csv('./MemodCafe.csv', encoding='cp949', mode='w', index=True)

if __name__ == '__main__':
    main()
```

code 2

1.Memod Cafe store crawling >>>>>>>>> 을 출력.

2. MemodeCafe_store(result)를 호출한 뒤 매장 정보 수집하고 리스트에 매장 정보 추가.

3. 수집한 매장 정보를 DataFrame으로 변환.

3. 변환한 DateFrame을 CSV파일로 CP949로 한글 인코딩 한 뒤에 MemodCafe 란 파일명으로 저장.

4. CSV 파일을 열면 추출한 매장 정보인 store, address를 열 이름으로 받아서 보여주는 것을 확인.

	A	B	C
1		store	address
2	0	둔산제일학원점	대전 서구 둔산서로 29, 1층 일부
3	1	대전시청점	대전광역시 서구 둔산중로 50, 101호(둔산동, 파이낸스빌딩)
4	2	대전을지한화점	대전 서구 둔산로51번길 84, 21세기빌딩 1층 103호
5	3	대전오류동점	대전 중구 계룡로816번길 11, 1층
6	4	대전도룡점	대전광역시 유성구 엑스포로 151번길 19, 지상1층 E102호(도룡동, 도룡하우스디)
7	5	대전대흥점	대전 중구 중교로 29, 1층
8	6	세종시청점	세종특별자치시 한누리대로 2144, 1층 110호(보람동, 스마트허브II)
9	7	세종대평벨리온점	세종특별자치시 집현중앙7로 6, 1동 1층 A130호(집현동, 지식산업센터)
10	8	세종중앙타운점	세종특별자치시 절재로 194, 1층 103호(어진동, 중앙타운)
11	9	천안아산역점	충청남도 아산시 배방읍 고속철대로 147, 107호(우성메디피아)
12	10	천안상명대점	충청남도 천안시 동남구 상명대길 28, 1층 101호(안서동)
13	11	천안미래에이스점	충청남도 천안시 서북구 백석공단1로 10, 지원140호(백석동 천안 미래에이스하이테크시티)
14	12	온양온천점	충청남도 아산시 중무로 7-1, 1층 104호(온천동)
15	13	천안삼성점	충청남도 천안시 서북구 3공단5로 58, 1층 104호(성성동)
16	14	전주아중점	전라북도 전주시 덕진구 무삼지로 55, 1층 101호(인후동1가)
17	15	전주서신점	전라북도 전주시 완산구 당산로 39, 상가108동 1층 113호(서신동, 새마을금고)
18	16	전주효자점	전라북도 전주시 완산구 거마평로 191-2, 1층(효자동1가)
19	17	전주평화점	전라북도 전주시 완산구 모악로 4750, 1층 (약국우측) (평화동1가)
20	18	전주효천지구점	전라북도 전주시 완산구 효천중앙로 22, A동 지하1층 A-B112호(효자동2가)
21	19	상주봇꽃점	경상북도 상주시 냉림4길 91(냉림동)
22	20	문경점촌점	경상북도 문경시 중앙6길 16, 1층 102호(점촌동)

- 견해

수업의 일환으로 webdriver를 이용하여 메머드 커피 홈페이지에 있는 매장 정보를 스크래핑 하여 파일로 저장했다. url을 이용한 방식과 다르게 코드가 간결해지는 느낌이 있으나, 페이지에서 goSearch를 이용하고 정보가 있는 곳을 찾는 과정이 번거로웠고 이러한 점으로 보아 이것도 일장일단이 있다고 생각된다.

3. 회귀분석(선형회귀)

3-1 회귀분석이란

회귀분석이란 가지고 있는 여러 데이터들을 기반으로 변수들 간의 관계를 모델링하고 예측하는 통계적 기법 중 하나로 종속 변수와, 독립 변수간의 관계를 설명하는데 사용된다. 대표적인 회귀분석에는 선형회귀분석이 있다.

3-2 선형회귀란

선형회귀분석은 종속 변수와 하나 이상의 독립 변수 간의 선형 관계를 모델링 하는 기법이다.

밑에서는 자동차 연비 관련 데이터를 가지고 jupyter notebook를 이용하여 선형회귀분석을 진행할 것이다.

3-3 자동차연비 데이터를 이용한 선형회귀분석

```
data_df = pd.read_csv('./auto-mpg.csv', header=0, engine='python')

print(' 데이터셋 크기 : ', data_df.shape)

data_df.head()
```

```
 데이터셋 크기 :  (392, 9)
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

```
data_df = data_df.drop(['car_name','origin'], axis=1, inplace=False)

data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 392 entries, 0 to 391
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int64
 2   displacement  392 non-null    float64
 3   horsepower    392 non-null    int64
 4   weight        392 non-null    int64
 5   acceleration  392 non-null    float64
 6   model_year    392 non-null    int64
dtypes: float64(3), int64(4)
memory usage: 21.6 KB
```

```
Y = data_df['mpg']
X = data_df.drop(['mpg'], axis=1, inplace=False)
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=0)
```

```
lr = LinearRegression()
```

```
lr.fit(X_train, Y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
print(lr.score(X_train, Y_train))
print(lr.score(X_test, Y_test))
```

```
0.811683812593389
0.7983345128838719
```

```
fig, axs = plt.subplots(figsize=(16, 16), ncols=3, nrows=2)

x_features = ['model_year', 'acceleration', 'displacement', 'weight', 'cylinders','horsepower']
plot_color = ['r', 'b', 'y', 'g', 'r','b']


for i, feature in enumerate(x_features):
    row = int(i/3)
    col = i%3
    sns.regplot(x=feature, y='mpg', data=data_df, ax=axs[row][col], color=plot_color[i])
```

1. Pandas 모듈을 이용하여 'auto-mpg.csv'파일에 저장되어 있는 데이터를 인코딩 오류 방지를 위하여 engine을 python으로 설정하여 호출한 뒤 head()로 확인.

2. 데이터 중 분석에 필요하지 않는 'origin', 'car_name'의 컬럼이 있으므로 모듈을 학습시켰을 때 신뢰도를 올리기 위하여 두 컬럼을 삭제.

3. info()로 'origin', 'car_name'의 컬럼이 삭제된 것을 확인.

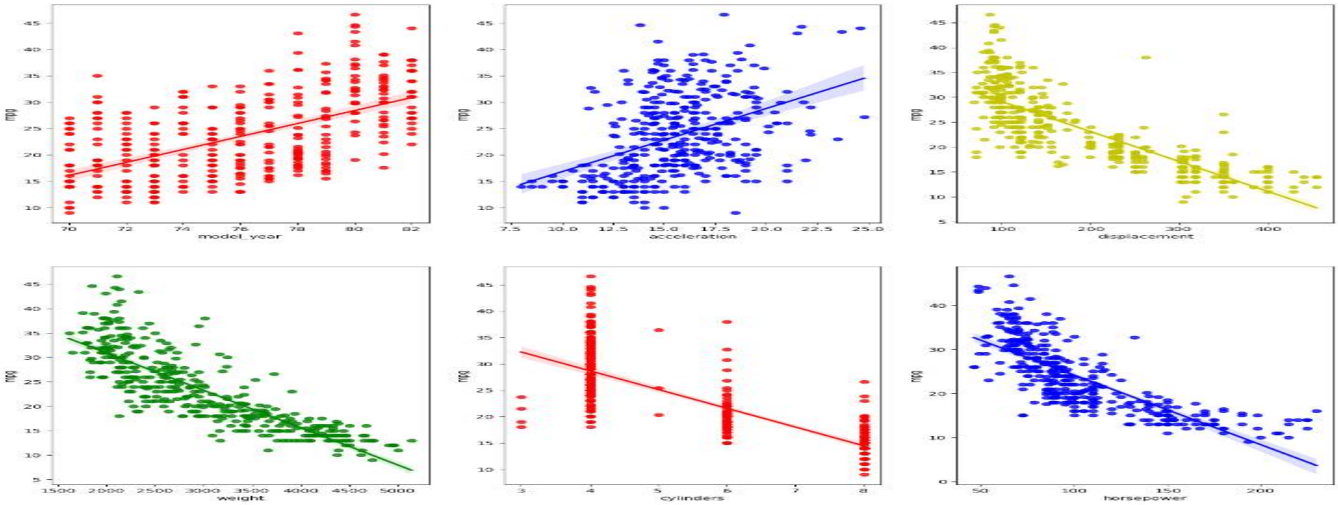
4. 종속 변수로 데이터프레임의 mpg열의 데이터를 할당, 독립 변수로 데이터프레임의 mpg열을 제외한 나머지 열의 데이터를 할당.

5. 학습 데이터와 테스트 데이터로 분할하되 테스트 데이터의 비율은 전체 데이터의 30%를 사용.

6. 선형 회귀 모델을 구현하고 구현한 모델에 학습데이터를 학습.

7. 학습한 모델의 결과 분석을 위하여 학습 데이터와 테스트 데이터의 신뢰도 확인.

8. Matplotlib, Seaborn을 사용하여 설정한 독립 변수와 종속 변수 간의 관계를 한 번에 보기 위해 각 독립 변수마다 색을 다르게 지정하여 구분할 수 있는 산점도와 회귀선을 시각화하는 그래프 생성.



<8번 행동으로 만들어진 그래프>

```
poly = PolynomialFeatures(include_bias=False)

poly.fit(X_train)
X_train_poly = poly.transform(X_train)
X_test_poly = poly.transform(X_test)
print(X_train_poly.shape)
```

```
(274, 27)
```

```
poly.get_feature_names_out()
```

```
array(['cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model_year', 'cylinders^2',
       'cylinders displacement', 'cylinders horsepower',
       'cylinders weight', 'cylinders acceleration',
       'cylinders model_year', 'displacement^2',
       'displacement horsepower', 'displacement weight',
       'displacement acceleration', 'displacement model_year',
       'horsepower^2', 'horsepower weight', 'horsepower acceleration',
       'horsepower model_year', 'weight^2', 'weight acceleration',
       'weight model_year', 'acceleration^2', 'acceleration model_year',
       'model_year^2'], dtype=object)
```

9. 다항식 조합을 생성하여 데이터를 확장하는 방법을 이용하기 위해 단항 데이터를 다항 데이터로 변환.

10. 9번의 행동으로 생성된 다항식 특성들의 이름 확인
이 특성들은 각각 새로운 독립 변수들로 활용 가능

```
lr.fit(X_train_poly, Y_train)
print(lr.score(X_train_poly, Y_train))
print(lr.score(X_test_poly, Y_test))
```

```
0.8847009937100931
0.8824080331334024
```

11. 다항식으로 변환한 데이터로 다시 학습시킨 뒤 신뢰도 확인.

7번 행동의 신뢰도와 비교하여 신뢰도가 올라간 것을 확인
0.8116~ -> 0.8847~
0.7983~ -> 0.8824~

```
print("연비를 예측하고 싶은 차의 정보를 입력해주세요.")
input_data = [[]]
input_data[0].append(int(input("cylinders : ")))
input_data[0].append(int(input("displacement : ")))
input_data[0].append(int(input("horsepower")))
input_data[0].append(int(input("weight : ")))
input_data[0].append(int(input("acceleration : ")))
input_data[0].append(int(input("model_year : ")))
```

```
연비를 예측하고 싶은 차의 정보를 입력해주세요.
cylinders :
```

```
input_data_poly = poly.transform(input_data)

mpg_predict = lr.predict(input_data_poly)
print("이 자동차의 예상 연비(mpg)는 %.2f 입니다." %mpg_predict)
```

12. 지금까지 입력된 정보를 기반으로 예측 수행 가능
차의 정보를 순서대로 입력하면 연비를 예측하여 예상 연비를 출력

- 견해

수업의 일환으로 회귀분석 중 선형회귀분석 이용하여 자동차 연비를 예측하는 프로그램을 만들었다.

하나의 종속 변수를 축으로 하고 나머지 독립 변수들을 점으로 표현한 그래프가 간단하고 이해하기 쉽게 표현된 부분이 마음에 든다.

일상생활에서 이를 이용한 것들은 광고 지출이 증가되면 판매에 미치는 영향을 예측하거나 제품 수요 예측하는 방식 등 이용할 수 있는 사례가 많다.

그러나 선으로만 표현하는 것은 선과 아주 멀리 떨어진 데이터를 어떻게 처리해야 하는지, 그리고 선형회귀분석을 사용하더라도 데이터가 급격하게 변할 경우 예측도 급격하게 변할 수 있으므로 적절한 상황에 맞는 분석을 선택해야 한다는 결론을 내릴 수 있다.

4. 로지스틱회귀

4-1 로지스틱회귀란

로지스틱 회귀는 통계학적 기법으로 변수들의 선형 결합을 통해 얻은 값을 로지스틱 함수에 넣어서 0과 1 사이의 확률 값을 얻어내고 이 값을 통해 분류함으로써 주로 분류 문제에 많이 활용된다.

밑에서는 유방암 진단 데이터를 가지고 jupyter notebook를 이용하여 로지스틱 회귀 분석을 진행할 것이다.

4-2 유방암 진단 데이터를 이용한 로지스틱 회귀 분석

```
[2]: b_cancer = load_breast_cancer()

[4]: b_cancer_df = pd.DataFrame(b_cancer.data, columns = b_cancer.feature_names)

[5]: b_cancer_df['diagnosis']= b_cancer.target

[8]: b_cancer_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   mean radius             569 non-null    float64
 1   mean texture            569 non-null    float64
 2   mean perimeter          569 non-null    float64
 3   mean area               569 non-null    float64
 4   mean smoothness         569 non-null    float64
 5   mean compactness        569 non-null    float64
 6   mean concavity          569 non-null    float64
 7   mean concave points     569 non-null    float64
 8   mean symmetry           569 non-null    float64
 9   mean fractal dimension  569 non-null    float64
 10  radius error            569 non-null    float64
 11  texture error           569 non-null    float64
 12  perimeter error         569 non-null    float64
 13  area error              569 non-null    float64
```

```
b_cancer_scaled = scaler.fit_transform(b_cancer.data)
```

```
print(b_cancer.data[0])
```

```
[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
 1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
 6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
 1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
 4.601e-01 1.189e-01]
```

```
print(b_cancer_scaled[0])
```

```
[ 1.09706398 -2.07333501  1.26993369  0.9843749   1.56846633  3.28351467
  2.65287398  2.53247522  2.21751501  2.25574689  2.48973393 -0.56526506
  2.83303087  2.48757756 -0.21400165  1.31686157  0.72402616  0.66081994
  1.14875667  0.90708308  1.88668963 -1.35929347  2.30360062  2.00123749
  1.30768627  2.61666502  2.10952635  2.29607613  2.75062224  1.93701461]
```

```
Y = b_cancer_df['diagnosis']
X = b_cancer_scaled
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=0)
```

```
lr_b_cancer = LogisticRegression()
```

```
lr_b_cancer.fit(X_train, Y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
print(lr_b_cancer.score(X_train,Y_train))
print(lr_b_cancer.score(X_test,Y_test))
```

```
0.9899497487437185
0.9766081871345029
```

```
Y_predict = lr_b_cancer.predict(X_test)
```

1. load_breast_cancer 함수를 사용하여 유방암 진단 데이터 셋 로드.

2. 데이터를 데이터프레임으로 변환하고, 각 열에 특성의 이름을 부여.

3. 데이터프레임에 유방암의 진단 결과('diagnosis')를 나타내는 열을 추가.

4. info()를 사용하여 데이터프레임 정보를 출력하여 기본정보 확인

- 3번 활동으로 추가된 'diagnosis'열은 맨 마지막에 추가되어 있는 것을 확인

5. scaler를 사용하여 데이터의 특성을 평균 0, 표준편차 1로 조정하여 전처리 과정을 진행.

6. print()를 이용하여 원본 데이터와 스케일링 된 데이터를 출력.

- 6번 활동을 이용하여 데이터 변환 전, 후를 비교 가능

7. X(독립 변수)로 스케일링 된 데이터를 가져오고 Y 변수로 진단결과 레이블을 가져옴

8. 학습 데이터와, 평가 데이터를 나눔

9. 로지스틱 모델을 초기화 한 뒤에 학습 데이터를 모델에게 학습.

10. 학습 데이터와 평가 데이터에 대한 정확도를 출력.

11. 평가 데이터에 대한 예측을 수행하여 예측 결과(Y_predict)를 구함.

```
accuracy = accuracy_score(Y_test, Y_predict)
precision = precision_score(Y_test, Y_predict)
recall = recall_score(Y_test, Y_predict)
f1 = f1_score(Y_test, Y_predict)
roc_auc = roc_auc_score(Y_test, Y_predict)
```

```
print('정확도: {0:.3f}, 정밀도: {1:.3f}, 재현율: {2:.3f},  F1: {3:.3f}'.format(accuracy,precision,recall,f1))
```

```
정확도: 0.977, 정밀도: 0.973, 재현율: 0.991,  F1: 0.982
```

12. 실제값과 예측값을 이용하여 정확도, 정밀도, 재현율, F1 스코어, ROC-AUC 스코어를 계산.

13. print()를 이용하여 계산된 지표들을 포맷을 활용하여 출력.

- 견해

수업의 일환으로 로지스틱 회귀분석 이용하여 유방암 진단을 예측하는 프로그램을 만들었다.

로지스틱 회귀분석은 간단하고 이해하기 쉬우며, 선형 결합을 통해 얻는 값을 0과 1 사이의 확률로 표현하는 방법으로 이진 분류 문제에 적합하다. 일상생활에서 이를 활용한 사례로는 이번 수업으로 진행한 어떠한 질병에 걸렸는지 예측하는 사례나 금융 기관에서 대출을 진행할 때 고객의 신용점수나 소득 수준 등을 통해 대출을 승인해도 되는지 예측하는 사례 등이 있다.

그러나 로지스틱 회귀분석은 선형 결합을 사용하기 때문에 비선형 데이터에는 적합하지 않으며 만약 데이터에 오타나 잘못된 입력 등 이상이 생기면 그 데이터를 가지고 계산하기에 그 이상치를 해결하기 위한 전처리를 하지 않는다면 잘못된 값이 나오는 단점이 있다.

따라서 이상치를 처리하지 않으면 결과에 영향을 미칠 수 있으며 또한, 특성이 많거나 클래스 수가 불균형하게 분포되어 있으면 로지스틱 회귀분석을 사용하기 어려울 수 있으므로 데이터의 특성을 파악한 뒤 적절한 모델을 선택해야 한다는 결론이 나온다.

5. 결정 트리

5-1 결정 트리란

데이터를 기반으로 여러 결정 규칙을 학습하여 의사 결정 프로세스를 모델링한 뒤 이를 통해 특정한 클래스 또는 값으로 분류하거나 예측할 수 있는 지도 학습 알고리즘 중 하나로 분류와 회귀 문제에 모두 사용이 가능하다.

밑에서는 행동 분석 데이터를 분류하기 위해 결정 트리를 이용하여 jupyter notebook으로 결정 트리를 만들고 과적합을 방지하기 위한 다른 방법들을 이용하여 행동분석을 해보려고 한다.

5-2 결정 트리를 이용한 행동분석 하기

```
feature_name_df = pd.read_csv('./UCI HAR Dataset/features.txt', sep='\s+',
                              header = None, names = ['index','feature_name'], engine = 'python')
feature_name_df
```

```
feature_name = feature_name_df.iloc[:, 1].values.tolist()
```

```
len(feature_name)
```

```
len(feature_name)-len(set(feature_name))
```

```
feature_name = [i +'_'+str(idx) for idx, i in enumerate(feature_name)]
```

```
feature_name[:5]
```

```
X_train = pd.read_csv('./UCI HAR Dataset/train/X_train.txt', sep='\s+', names=feature_name, engine='python')
X_train.head()
```

```
X_test = pd.read_csv('./UCI HAR Dataset/test/X_test.txt', sep='\s+', names=feature_name, header=None, engine='python')
```

```
Y_train = pd.read_csv('./UCI HAR Dataset/train/y_train.txt', sep='\s+', names=['action'], header =None, engine='python')
```

```
Y_test = pd.read_csv('./UCI HAR Dataset/test/y_test.txt', sep='\s+', names=['action'], header =None, engine='python')
```

```
X_train.shape, Y_train.shape, X_test.shape, Y_test.shape
```

1. features.txt 파일을 읽어와서 데이터프레임으로 저장하되 파일 내용은 1열은 index, 2열은 feature_name으로 설정 .

2. feature_name의 데이터를 리스트로 저장하고 중복된 특징이 있는지 확인하고 중복을 제외한 특징의 개수를 계산.

3. 각 특징에 인덱스를 추가하여 특징의 이름을 업데이트 하고 특징 이름 중 처음 5개를 확인.

4. X_train.txt, X_test.txt, Y_train.txt, Y_test.txt 파일을 읽어와서 데이터프레임으로 저장 한 뒤 데이터의 형태 확인.

```
label_name_df = pd.read_csv('./UCI HAR Dataset/activity_labels.txt', sep = '\s+',
                            header = None, names = ['index', 'label'], engine='python')
label_name_df
```

5. activity_labels.txt 파일을 읽어와서 데이터프레임으로 저장하되 파일 내용은 1열은 index, 2열은 label으로 설정.

```
label_name = label_name_df.iloc[:,1].values.tolist()
```

6. label_name_df에 2열 데이터를 리스트로 변환한 뒤 확인.

```
label_name
```

```
dt_HAR = DecisionTreeClassifier(random_state=156)
```

7. 난수를 156으로 설정한 결정 트리 모델을 생성하고 X_train, Y_train을 결정 트리 모델에 학습.

```
dt_HAR.fit(X_train, Y_train)
```

8. 테스트 데이터(X_test)를 사용하여 예측을 수행하고 Y_predict에 저장.

```
Y_predict = dt_HAR.predict(X_test)
```

9. 실제 레이블(Y_test)과 예측 결과를 비교하여 정확도 계산한 뒤 출력.

```
accuracy = accuracy_score(Y_test, Y_predict)
print('결정 트리 예측 정확도 {0:4f}'.format(accuracy))
```

10. 사용된 결정 트리 모델의 하이퍼파라미터를 출력.

```
print('결정 트리의 현재 하이퍼 매개변수: \n', dt_HAR.get_params())
```

11. 하이퍼파라미터 범위를 설정 한 후 훈련 데이터를 사용하여 그리드 서치를 수행후 최적의 하이퍼파라마터 탐색.

```
params = {'max_depth' : [6,8,10,12,16,20,24]}

grid_cv = GridSearchCV(dt_HAR, param_grid = params, scoring='accuracy', cv = 5, return_train_score = True)
grid_cv.fit(X_train, Y_train)
```

12. 그리드 서치 결과를 데이터프레임으로 저장한 후에 최고 평균 정확도와 최적 하이퍼파라미터 출력.

```
cv_results_df = pd.DataFrame(grid_cv.cv_results_)

cv_results_df[['param_max_depth', 'mean_test_score', 'mean_train_score']]

print('최고 평균 정확도: {0:.4f}, 최적 하이퍼 매개변수: {1}'.format(grid_cv.best_score_, grid_cv.best_params_))
```

13. 탐색할 다른 하이퍼파라미터를 설정한 후에 11~12번 행동 반복.

```
params = {'max_depth':[8,16,20], 'min_samples_split':[8,16,24]}
grid_cv = GridSearchCV(dt_HAR, param_grid = params, scoring = 'accuracy', cv =5, return_train_score = True)
grid_cv.fit(X_train, Y_train)

cv_results_df = pd.DataFrame(grid_cv.cv_results_)
cv_results_df[['param_max_depth', 'param_min_samples_split','mean_test_score', 'mean_train_score']]

print('최고 평균 정확도: {0:.4f}, 최적 하이퍼 매개변수: {1}'.format(grid_cv.best_score_, grid_cv.best_params_))
```

14. 최적의 하이퍼파라미터를 가진 모델을 선택 후 테스트 데이터를 사용하여 예측을 수행. 그 뒤에 예측 결과와 실제 레이블을 비교하여 최적 모델의 정확도 계산 및 출력.

```
best_dt_HAR = grid_cv.best_estimator_
best_Y_predict = best_dt_HAR.predict(X_test)
best_accuracy = accuracy_score(Y_test, best_Y_predict)

print('best 결정 트리 예측 정확도: {0:4f}'.format(best_accuracy))
```

15. 최적의 결정 트리 모델에서 특성 중요도 추출 후 각 특성에 해당하는 열 이름을 인덱스로 설정 한 뒤 시리즈로 변환.

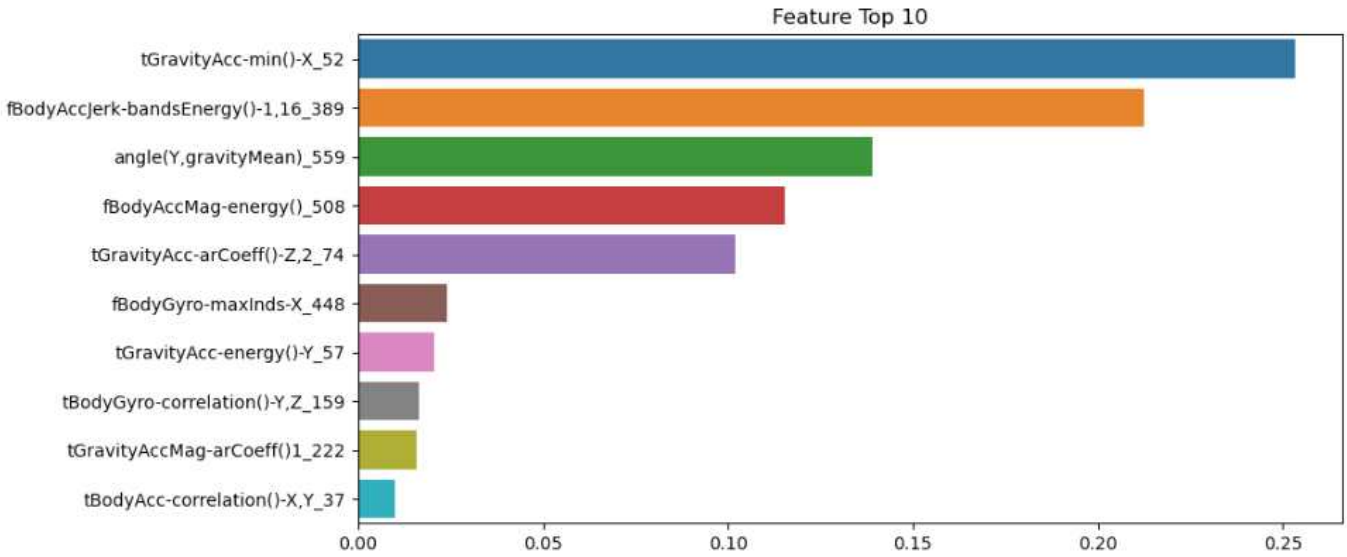
```
feature_importance_values = best_dt_HAR.feature_importances_
feature_importance_values_s = pd.Series(feature_importance_values,index = X_train.columns)

feature_top10 = feature_importance_values_s.sort_values(ascending = False)[:10]
```

16. 중요도를 내림차순 정렬 후 상위 10개 특성을 선택한 후에 그래프의 크기와 제목을 설정한 후에 막대 그래프로 시각화 및 출력.

```
plt.figure(figsize=(10,5))
plt.title('Feature Top 10')
sns.barplot(x=feature_top10, y = feature_top10.index)
plt.show()
```

17. 최적의 결정 트리 모델을 'tree.dot' 파일로 보낸 뒤 파일 내용을 기반으로 그래프 생성한 뒤 생성한 그래프를 '행동분석결과'라는 이름으로 jpg 형식으로 저장.

```
from sklearn.tree import export_graphviz

export_graphviz(best_dt_HAR, out_file ="tree.dot", class_names=label_name, feature_names = feature_name,
                impurity = True, filled = True)

import graphviz

with open("tree.dot") as f:
    dot_graph = f.read()
graph= graphviz.Source(dot_graph, format='png')
graph.render(filename='행동분석결과',directory='./',format='jpg')
```



<17번 행동으로 만들어진 그래프>

- 견해

수업의 일환으로 행동 분석 데이터를 이용하여 결정 트리를 만들고 행동분석결과라는 파일을 만들었다.

결정 트리는 만들어진 그래프를 보면 시각화 할 때 적절한 조건을 설정한다면 직관적으로 이해하기 쉬우며 어떤 특성에 더 중요성을 두고 예측을 수행했는지를 알려주는 등의 장점이 있다. 일상생활에서 이를 활용한 사례로는 고객의 특성을 기반으로 마케팅 전략을 수립 및 특정 제품에 대한 반응 예측에 이용하거나 일반적으로 사람들이 언급하는 알고리즘 중 하나로 결정트리를 사용하는 경우가 많다.

그러나 데이터의 작은 변화에 모델 구조가 변할 수 있어서 안정성이 떨어지며 클래스가 불균형하게 분포되어 있는 경우에는 큰 클래스에 더 많은 가지가 생성되기 때문에 작은 클래스의 예측이 어려워지는 등의 단점이 있다.

그러니 우선 데이터의 클래스가 균형적인지의 확인이나 아니면 이러한 단점을 극복하기 위해 다양한 방법들을 적용한다면 만족할 만한 결과를 얻을 수 있을 것으로 기대한다.

6. K-알고리즘

6-1 K-알고리즘이란

입력한 데이터를 특정값으로 분류하는데 현재 데이터와 가장 가까운 k개의 데이터를 찾아 가장 많은 분류 값으로 데이터를 분류하는 알고리즘이다.

밑에서는 쇼핑몰 사이트 이용정보를 이용하여 K-알고리즘으로 분류를 하려고 한다.

```
retail_df = pd.read_excel('./Online Retail.xlsx')
retail_df.head()
```

```
retail_df = retail_df[retail_df['Quantity']>0]
retail_df = retail_df[retail_df['UnitPrice']>0]
retail_df = retail_df[retail_df['CustomerID'].notnull()]
retail_df.info()
```

```
retail_df['CustomerID'] = retail_df['CustomerID'].astype(int)
```

```
print(retail_df.isnull().sum())
print(retail_df.shape)
```

```
retail_df.drop_duplicates(inplace = True)
print(retail_df.shape)
```

1. 쇼핑몰 사이트 이용 데이터 파일을 데이터프레임으로 불러온 후에 출력하여 데이터의 구조나 정보를 확인한다.

2. 'Quantity' 열이 양수인 행, 'UnitPrice' 열이 양수인 행, 'CustomerID' 열이 누락되지 않은 행만 선택하여 데이터프레임을 갱신하고 'CustomerID' 열의 데이터 타입을 정수형으로 변환.

3. 각 열의 결측치 개수, 데이터프레임의 행과 열의 개수를 출력하여 확인하고 중복된 행을 제거하고 데이터프레임의 반영한 후 중복 제거 후의 행과 열의 개수를 확인한다.

```
pd.DataFrame([{'Product':len(retail_df['StockCode'].value_counts()),
               'Transaction':len(retail_df['InvoiceNo'].value_counts()),
               'Customer':len(retail_df['CustomerID'].value_counts())}],
               columns = ['Product', 'Transaction','Customer'],
               index = ['counts'])
```

```
retail_df['Country'].value_counts()
```

```
retail_df['SaleAmount'] = retail_df['UnitPrice']*retail_df['Quantity']
```

```
aggregations = {
 'InvoiceNo':'count',
 'SaleAmount':'sum',
 'InvoiceDate':'max'
}

customer_df = retail_df.groupby('CustomerID').agg(aggregations)
customer_df = customer_df.reset_index()
```

```
customer_df = customer_df.rename(columns = {'InvoiceNo':'Freq', 'InvoiceDate':'ElapsedDays'})
```

```
import datetime

customer_df['ElapsedDays'] = datetime.datetime(2011,12,10) - customer_df['ElapsedDays']
```

```
customer_df['ElapsedDays'] = customer_df['ElapsedDays'].apply(lambda x: x.days+1)
```

```
customer_df.info()
```

4. 각 행은 'Product', 'Transaction', 'Customer' 의 순서로, 저 3개의 칼럼에 대한 값으로 구성되며 제품 수, 거래 수, 고객 수를 구하여 데이터를 입력한 데이터프레임을 생성.

5. 각 국가별 거래 횟수를 확인하고 'SaleAmount' 열을 새로 생성한 후 각 거래별 판매 금액을 계산한다.

6. 'CustomerID' 열을 기준으로 데이터를 그룹화한 뒤 그룹별로 거래 횟수, 총 판매 금액, 최근 거래 일을 집계.

7. 'ElapsedDays' 열에 저장된 최근 거래 일로부터 경과한 날 수를 계산하고 info()를 통해 데이터프레임의 정보 확인.

```
fig, ax = plt.subplots()
ax.boxplot([customer_df['Freq'], customer_df['SaleAmount'], customer_df['ElapsedDays']], sym = 'bo')
plt.xticks([1, 2, 3], ['Freq', 'SaleAmount','ElapsedDays'])
```

8. 'Freq', 'SaleAmount', 'ElapsedDays' 열에 대한 박스 플롯을 생성하고 x축의 눈금을 각각 'Freq', 'SaleAmount', 'ElapsedDays'로 지정.

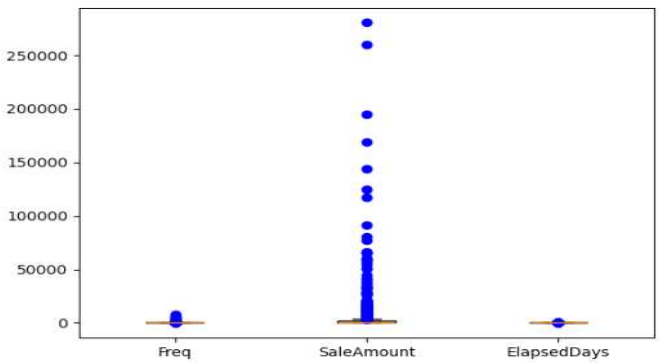
```
customer_df['Freq_log'] = np.log1p(customer_df['Freq'])
customer_df['SaleAmount_log'] = np.log1p(customer_df['SaleAmount'])
customer_df['ElapsedDays_log'] = np.log1p(customer_df['ElapsedDays'])
```

9. 각 요소의 로그 변환 값을 각 요소_log 열에 저장.

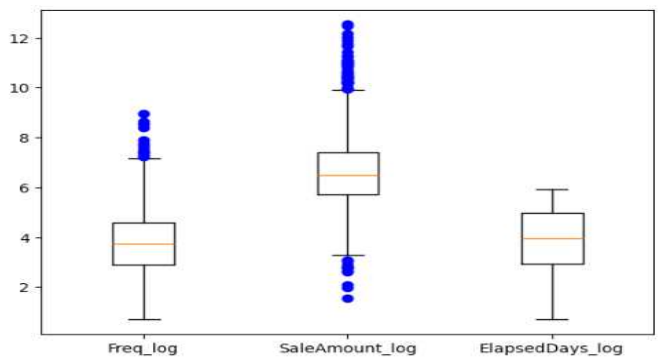
```
fig, ax = plt.subplots()
ax.boxplot([customer_df['Freq_log'], customer_df['SaleAmount_log'], customer_df['ElapsedDays_log']], sym = 'bo')
plt.xticks([1, 2, 3], ['Freq_log', 'SaleAmount_log', 'ElapsedDays_log'])
```

10. 'Freq_log', 'SaleAmount_log', 'ElapsedDays_log' 열에 대한 박스 플롯을 생성하고 x축 눈금을 각각 'Freq_log', 'SaleAmount_log', 'ElapsedDays_log'로 지정.

- 로그 변환을 한 뒤 만든 박스 플롯에 데이터가 제대로 표현되어 있는 것이 확인.



<8번 행동으로 만들어진 박스플롯>



<10번 행동으로 만들어진 박스플롯>

```
X_features = customer_df[['Freq_log', 'SaleAmount_log', 'ElapsedDays_log']].values
```

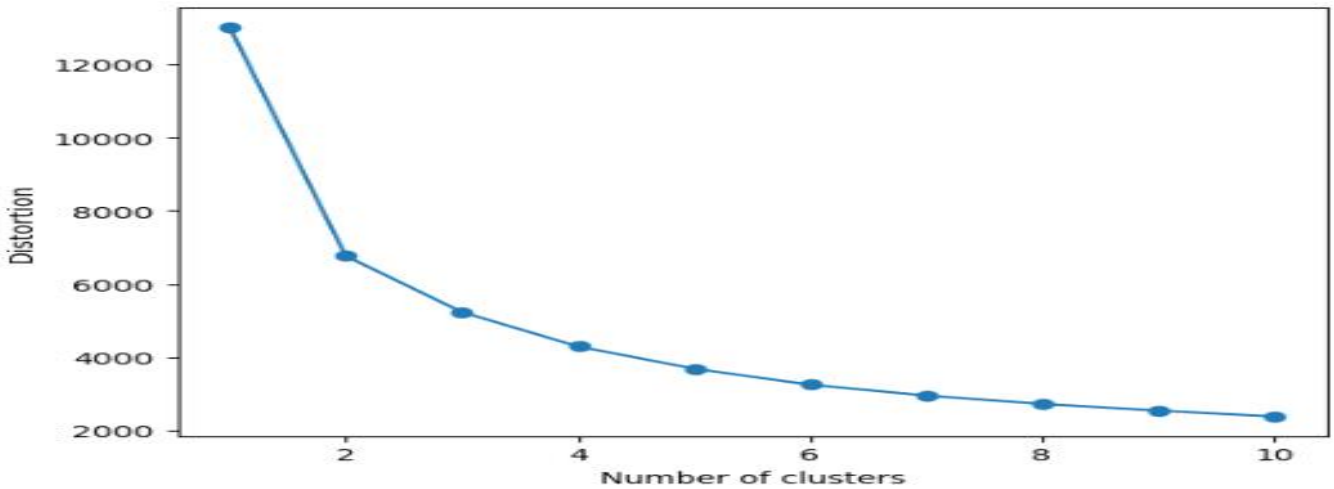
```
from sklearn.preprocessing import StandardScaler
X_features_scaled = StandardScaler().fit_transform(X_features)
```

11. 고객 데이터프레임에서 'Freq_log', 'SaleAmount_log', 'ElapsedDays_log' 열을 추출한 뒤 배열로 변환하고 추출된 특성들에 대해 스케일 조정.

```
distortions = []

for i in range(1, 11):
    kmeans_i = KMeans(n_clusters = i, random_state = 0)
    kmeans_i.fit(X_features_scaled)
    distortions.append(kmeans_i.inertia_)

plt.plot(range(1,11), distortions, marker = 'o')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.show()
```

12. 클러스터의 개수를 1부터 10까지 변화시키면서 반복하여 클러스터의 개수에 해당하는 K-평균 객체를 생성한 뒤 데이터에 적용시키고 적용시킨 K-평균 객체의 왜곡 값을 리스트에 추가.

13. 12번 활동으로 만들어진 왜곡 값을 x축은 클러스터의 개수를, y축은 왜곡 값으로 라벨을 지정하고 그래프를 출력



<13번 행동으로 만들어진 그래프>

```
kmeans = KMeans(n_clusters=2, random_state=0)
```

14. K-평균 알고리즘을 사용하여 클러스터링을 수행하고, 'fit_predict' 메서드를 사용하여 클러스터링을 수행한 뒤 반환된 클러스터 레이블은 변수에 저장.

```
customer_df['ClusterLabel'] = Y_labels
customer_df.head()
```

15. 클러스터링 결과를 데이터프레임에 새로운 열인 'ClusterLabel'에 저장.

```
def silhouetteViz(n_cluster, X_features):
    kmeans = KMeans(n_clusters = n_cluster, random_state = 0)
    Y_labels = kmeans.fit_predict(X_features)

    silhouette_values = silhouette_samples(X_features, Y_labels, metric = 'euclidean')

    y_ax_lower, y_ax_upper = 0, 0
    y_ticks = []

    for c in range(n_cluster):
        c_silhouettes = silhouette_values[Y_labels == c]
        c_silhouettes.sort()
        y_ax_upper += len(c_silhouettes)
        color = cm.jet(float(c) / n_cluster)
        plt.barh(range(y_ax_lower, y_ax_upper), c_silhouettes, height = 1.0, edgecolor = 'none', color = color)
        y_ticks.append((y_ax_lower + y_ax_upper) / 2.)
        y_ax_lower += len(c_silhouettes)

    silhouette_avg = np.mean(silhouette_values)
    plt.axvline(silhouette_avg, color = 'red', linestyle = '--')
    plt.title('Number of Cluster : '+ str(n_cluster) + '\n'  + 'Silhouette Score : '+ str(round(silhouette_avg,3)))
    plt.yticks(y_ticks, range(n_cluster))
    plt.xticks([0, 0.2, 0.4, 0.6, 0.8, 1])
    plt.ylabel('Cluster')
    plt.xlabel('Silhouette coefficient')
    plt.tight_layout()
    plt.show()
```

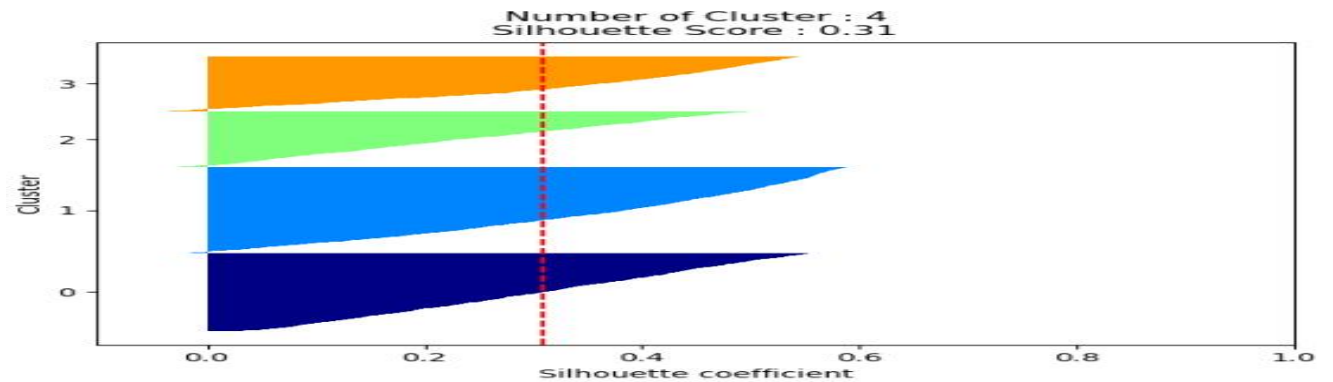
16. 클러스터의 개수와 특성 데이터를 받아서 사용하는 함수를 정의.

17. 클러스터 개수를 이용하여 KMeans 클러스터링 객체를 알고리즘의 랜덤 시드를 설정하고 생성한 뒤 'fit_predict' 메서드를 사용하여 클러스터링을 수행한 뒤 반환된 클러스터 레이블은 변수에 저장.

18. 특성 데이터(X_features), 클러스터 레이블(Y_labels), 거리 측정 방법(metric) 사용하여 'Silhouette' 계수를 계산.

19. 18번 행동을 클러스터의 개수만큼 반복하고 각 클러스터에 대한 'Silhouette' 계수를 계산하고 시각화.

20. 19번 행동으로 계산된 계수를 수평 막대그래프로 출력하되 막대그래프의 높이는 클러스터에 속하는 데이터의 수와 동일하게 설정하고 전체 데이터 셋에 대한 평균 'Silhouette' 계수를 빨간색 점선으로 그래프에 표시.



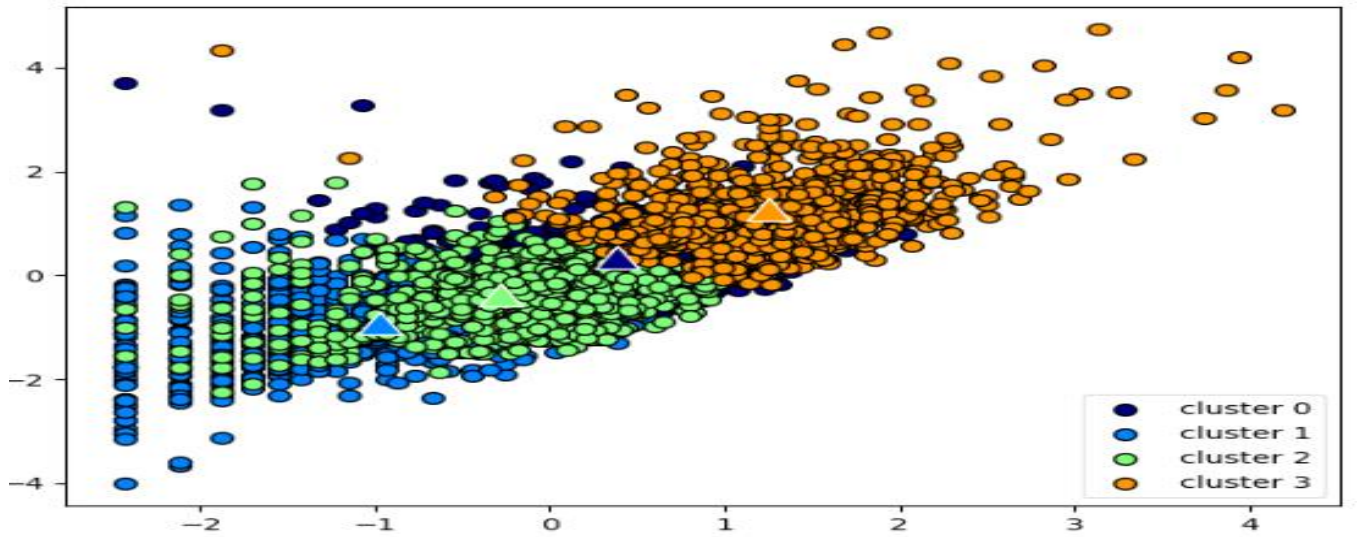
<20번 행동으로 만들어진 그래프>

```
def clusterScatter(n_cluster, X_features):
    c_colors = []
    kmeans = KMeans(n_clusters = n_cluster, random_state = 0)
    Y_labels = kmeans.fit_predict(X_features)

    for i in range(n_cluster):
        c_color = cm.jet(float(i) / n_cluster)
        c_colors.append(c_color)
        plt.scatter(X_features[Y_labels == i,0], X_features[Y_labels == i,1], marker = 'o', color =  c_color,
                    edgecolor = 'black', s = 50, label = 'cluster '+ str(i))

    for i in range(n_cluster):
        plt.scatter(kmeans.cluster_centers_[i,0], kmeans.cluster_centers_[i,1], marker = '^', color = c_colors[i], edgecolor = 'w', s = 200)
        plt.legend()
        plt.grid()
        plt.tight_layout()
    plt.show()
```

21. 다수의 클러스터를 만들고 특성 데이터(X_features)를 사용하여 클러스터링을 수행.

22. K-평균 알고리즘을 사용하여 특성 데이터에 대한 클러스터 할당을 예측.

23. 각 클러스터 마다 반복하고 다른 색상을 부여하여 산점도로 표시.



<23번 행동으로 만들어진 그래프>

```
best_cluster = 4
kmeans = KMeans(n_clusters = best_cluster, random_state = 0)
Y_labels = kmeans.fit_predict(X_features_scaled)
```

24. 베스트 클라스터를 4로 설정 후 스케일링된 데이터에 대한 클러스터 할당을 예측.

```
customer_df['ClusterLabel'] = Y_labels
customer_df.head()
```

25. 클러스터링 결과를 원본 데이터 프레임의 'ClusterLabel' 열에 각 고객의 클러스터 레이블을 저장.

```
customer_df.to_csv('./Online_Retail_Customer_Cluster.csv')
```

26. 클러스터링 결과가 포함된 데이터 프레임을 CSV파일로 저장.

```
customer_df.groupby('ClusterLabel')['CustomerID'].count()
```

견해

수업의 일환으로 K-알고리즘을 이용하여 쇼핑몰 사이트 이용정보를 이용하여 분류를 진행하였다.

K-알고리즘은 직관적이고 구현하기 쉬운 알고리즘이며 클러스터링 결과를 시각화하기에 해석하기도 쉽다는 등의 장점이 있다. 일상생활에서 이를 활용한 사례로는 온라인 커뮤니티에서 사용자를 그룹화하여 그룹에 맞는 맞춤화된 콘텐츠를 제공하거나 고객을 세분화하여 유사한 구매 패턴이나 관심사를 가진 그룹을 형성하여 맞춤화된 마케팅 전략을 수립하는데 사용하는 등이 있다.

그러나 최적의 클러스터 개수를 선택하는 것은 주관적이므로 분석가가 잘못 판단했을 경우 결과가 크게 왜곡될 수가 있으며 초기 중심점의 위치를 임의로 선택하여 시작하기에 초기 중심점이 잘못 선택되었을 경우에도 결과가 제대로 나오지 않을 수 있는 등의 단점도 가지고 있다.

그러니 K-알고리즘의 종류가 K-평균, K-군집, K-이웃 등 종류에 따라 각 알고리즘을 파악하고 알맞게 사용해야 만족할만한 결과가 나올 것이라 생각된다.

7. 머신 러닝에 대한 견해

수업의 일환으로 머신 러닝에 대해 이해하고 그 일환으로 여러 데이터을 이용하여 머신 러닝을 익혔다.

머신 러닝은 컴퓨터가 데이터에서 학습하고 패턴을 발견하여 작업을 자동화하고 예측을 수행할 수 있게 하는 기술로 현대 사회에서 쓰이지 않는 곳이 없을 정도로 많은 산업과 분야에서 사용되고 있다.

하지만 머신 러닝은 데이터에 의존하여 데이터가 많지 않으면 제대로 성능이 나오지 않고 데이터가 과적합되는 문제 등 단점도 가지고 있는 것이 사실이다.

그러니 머신 러닝의 성능을 끌어올리고 제대로 사용하기 위해 과적합을 방지하는 방법을 찾거나 구조를 최적화 하는 방법을 찾는 등 많은 방법을 이용해야 할 것이다. 그러면 자율 주행 자동차 기술이나 인공지능의 발전, 의료 등 많은 분야에서 미래적인 발전이 가능할 것이라 생각한다.