Term Project 결과보고서

프로젝트 기간	2021년 2월 1일~2021년 2월 3일
담임강사	노태상
조명	4조 안밀리조
조원	황재식 신유나 오지예 정원선 최윤근

Ⅰ. 프로젝트 개요

가. 프로젝트 소개

- 한국도로공사에서 제공하는 공공데이터를 기반으로 전국 톨게이트 입·출구 교통량 API 데이터를 수집하고, 이상 탐지 모델 설계를 통해 실시간 교통량 증가량 이상을 감지한다. 이후 이상 값 측정 지역의 원인을 파악하고 해결 방안을 탐구하는 것을 주된 목표로 한다.

1. 톨게이트 일교통량_1일 API 데이터(고속도로 공공데이터 포털 : http://data.ex.co.kr) 수집
2. 3년이내 톨게이트 입·출구 교통량 데이터를 참고하여 각 시간, 요일, 월별로 실시간 API 데이터와 비교하여 영업소(총 365개) 교통량 데이터 증감 확인 후 교통량이 많은 영업소를 이상 값으로 탐지
3. 데이터 분석을 바탕으로 교통량이 안정적인 시간대 제공 및 교통 혼잡을 해소하기 위한 타 경로 안내

나. 개발배경 및 필요성

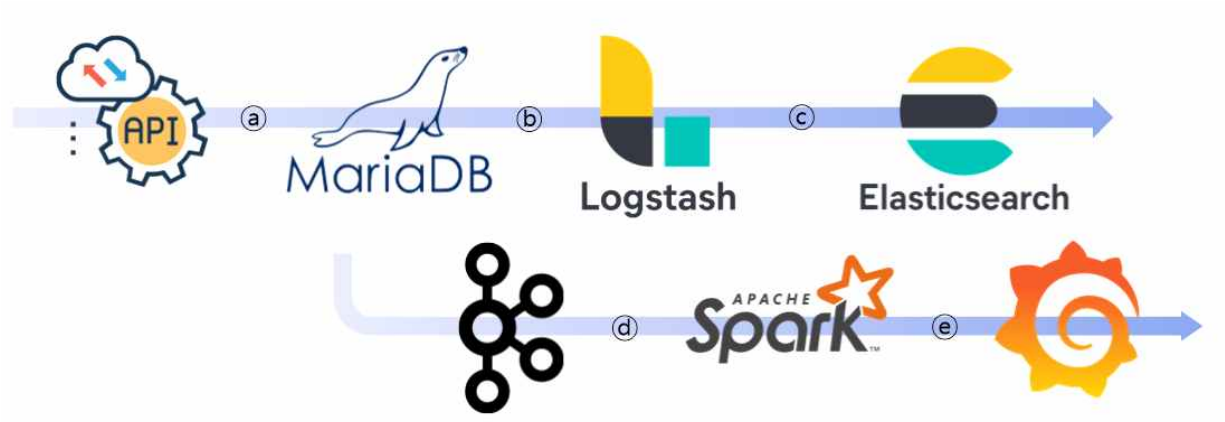
- EDA (탐색적 데이터 분석) 및 데이터 전처리

EDA(탐색적 데이터 분석)의 목적은 데이터를 이해하는 것이다. 데이터를 다양한 각도에서 관찰하고 이해하는 과정에서 데이터의 품질과 데이터에 담긴 정보량이 큰 영향을 미치며, 충분히 가다듬어진 데이터를 확보하는 것이 중요하다. 분석의 소스가 될 데이터는 다양한 분야에서 생성되고 있으며, 이 데이터 중 분석에 사용 가능한 데이터를 확보해야 한다. 이러한 관점에서 많은 양의 데이터 확보가 가능한 한국도로공사 고속도로 공공데이터 포털에서 실시간 API 데이터를 활용하여 톨게이트 교통량 데이터를 수집하여 가공한다.

- 실시간 이상탐지 모델

이상 탐지란 특정 도메인에서 정상으로 규정된 데이터와 다른 특징을 가지는 데이터를 찾아내는 것이다. 데이터만을 가지고 학습할 때 무작위로 분포된 데이터 중 비슷한 특성을 가진 데이터들을 묶는 방식으로 대표적인 비지도 학습인 K-means 알고리즘을 사용한다. 이 과정에서 이전 3개년의 교통량을 활용하여 학습시킨 후, 이상값을 확인한다. 이후 해당 원인을 분석하고 해결 방안을 고찰하여 실생활에서 발생하는 톨게이트 교통량 이상 문제를 해결할 수 있다.

다. 작품 구성도



라. 작품의 특징 및 장점

ㅇ 신뢰성

- 실제 측정 교통량 기반 객관적 기록데이터
- 정확한 지점(영업소 입·출구), 시점(일시), 상황(교통량) 데이터

ㅇ 실시간성

- 실시간으로 발생하는 톨게이트 교통량 이상 감지
- 실시간으로 이상값 발생 원인 분석 및 해결방안 고찰

ㅇ 실용성

- 수업시간에 학습한 이론 적용 및 Tool 활용
- 실생활과 밀접한 관련이 있는 주제 선정

Ⅱ. 프로젝트 수행결과

가. 주요기능

구분	기능	항목	설명
S/W	OS	CentOS Linux	개발 OS
	Environment	Oracle JAVA	개발 도구 구동 환경
	Filter	Logstash	오픈소스 서버측 데이터 처리 파이프라인
	Distributor	Kafka	분산형 스트리밍 플랫폼
		Zookeeper	Kafka의 메타데이터 정보 저장, Kafka의 상태관리
	Streamer	Spark	실시간 데이터 처리, 머신러닝 알고리즘 실행
	Repository	Elasticsearch	Logstash를 통해 수신된 데이터 저장
	Visualization	Grafana	데이터 시각적 탐색 및 시각화
	Shipper	Filebeat	오픈 소스 데이터 발송
	Database	MariaDB	데이터베이스 관리
	WorkPlace	Anaconda	파이썬 컴파일러, 코드 작성 및 확인
	Language	Python	머신러닝 적용 프로그래밍 언어
H/W	Server	OracleVM Virtualbox	컴퓨터 가상화 프로그램
	Server	Samsung Laptop	Host OS

나. 프로젝트 개발환경

구분		항목	적용내역
S/W	OS	CentOS Linux	CentOS 7
	개발환경	Oracle JAVA	1.8.0_281
	개발도구	Logstash	7.10.2
		Kafka	2.7.0
		Zookeeper	3.5.6
		Spark	2.4.7
		Elasticsearch	7.10.2
		Grafana	6.7.3
		Filebeat	7.10.2
		MariaDB	10.4
		Anaconda	Anaconda3 2019.10
	개발언어	Python	3.7.4
H/W 구성 장비	디바이스	Oracle VM Virtualbox	버전 6.1.16 r1400961, SSD 100GB, 총 5대
		Samsung Laptop	i7-6700K, DDR3 16GB, Geforce 920MX, SSD 256GB, 총 5대

다. 장비(기자재/재료) 활용

번호	품명	작품에서의 주요기능
1	Oracle VM Virtualbox	버전 6.1.16 r1400961, SSD 100GB, 총 5대
2	Samsung Laptop	i7-6700K, DDR3 16GB, Geforce 920MX, SSD 256GB, 총 5대

라. 결과물 상세 이미지

마. 달성성과

ㅇ 데이터 전처리

공공데이터 포털에서 Open API를 활용하여 톨게이트 입·출구 교통량을 수집하고, MariaDB, ELK, Kafka를 활용하여 데이터 전처리 과정을 수행하였다. 원본 데이터 중 원하는 항목의 데이터를 추출하였고, 실시간으로 데이터를 확인하였다. 이상탐지 모델 알고리즘 적용을 위한 데이터셋을 도출하였다.

ㅇ 실시간 이상감지 모델

도출된 데이터셋과 Spark를 활용하여 실시간 교통량 이상을 탐지하는 모델을 구현하고자 하였다. K-means 알고리즘을 공부하였고, 시각화를 통해 실시간으로 들어오는 데이터를 확인할 수 있도록 구현하였다. Spark에서 EDA를 하여 가공을 하고, 과거 데이터와 실시간 데이터를 spark에 넣어 학습을 시켜 이상 감지를 시도하려고 했으나 시간 부족으로 어려움을 겪었다.

Ⅲ. 프로젝트 수행방법

가. 업무분장

번호	성명	역할	담당업무
1	노태상	담임강사	프로젝트 수행 관리
2	황재식	조장	Project Leading 프로젝트 전체를 관리하고 실무를 겸하여 프로젝트 목표 달성 촉진
3	신유나	조원	Development Engineering 프로젝트가 다루는 데이터의 파이프라인 구축 및 관련 기술 설계
4	오지예	조원	Reporting 프로젝트 산출물들을 정리하고 이슈와 문제해결 과정 문서화
5	정원선	조원	Planning 프로젝트가 주어진 시간과 리소스에 맞게 진행될 수 있도록 기획
6	최윤근	조원	Quality Controling 프로젝트 목표에 따른 결과물 상태 확인 및 해결방안 모색

나. 프로젝트 수행일정

일자	추진내용
1일차 **(01/26)**	- R&R 정의 및 프로젝트 계획서 작성 - 개발 환경 구체화 - 피드백 확인 후 보완 회의
2일차 **(01/27)**	- 개발 환경 조사 및 분석 - 개발 환경 설계 및 세팅 - Open API 기반 톨게이트 교통량 데이터 수집
3일차 **(01/28)**	- 수집한 톨게이트 데이터 전처리 및 PoC - 개발 환경 추가 세팅 - 데이터 파이프라인 설계
4일차 **(01/29)**	- 전처리 데이터 DB 적재 - DB 적재 후 Logstash로 이동
5일차 **(02/01)**	- 프로젝트 방향 설정 구체화 - 파이프라인 파일럿 테스트 - 이상감지 알고리즘 모델 조사 및 PoC
6일차 **(02/02)**	- 이상감지 알고리즘 모델 설계 - 이상감지 알고리즘, 파이프라인 고도화 및 시각화
7일차 **(02/03)**	- 결과보고서 작성, 최종 자료 정리 - 최종 평가 준비
기타	특이사항 - COVID-19로 인하여 온라인/오프라인 회의 병행

다. 문제점 및 해결방안

구분	문제점	해결방안
Virtual Box	머신 실행 오류	경로 재설정, 머신 백업 및 정리 후 재실행
공공 데이터	공공데이터포털(https://www.data.go.kr)에서 OpenAPI 서비스 이용시 "SERVICE KEY IS NOT REGISTERED ERROR."(인증키 오류) 발생으로 추가적으로 인코딩/디코딩 과정 필요	고속도로 공공데이터 포털 – 한국도로공사(http://data.ex.co.kr)로 변경하여 OpenAPI 서비스 이용
	공공 데이터 원본 복잡성	"unitName", "unitCode", "trafficAmout", "inoutName" 4개 항목만 추출
	원본의 경우 4분기로 분할되어 있음	당해 분기 데이터를 통합하는 작업 수행 (2020년의 경우 4분기 파일이 존재하지 않아 제외)
Python	Colab 이용 시 CentOS7와 연동이 잘 되지 않음	주피터노트북 설치 후 CentOS7과 드라이브가 잘 공유됨을 확인
Kafka	데이터 수신 오류	파이프라인 필터에 codec => json 추가

IV. 참고자료

References	URL
Apache Kafka	https://m.blog.naver.com/sundooedu/221230385470
Data preprocessing	https://velog.io/@kimdukbae/%EB%8D%B0%EC%9D%B4%ED%84%B0-%EC%A0%84%EC%B2%98%EB%A6%AC-%EC%9D%B4%EB%A1%A0-%EB%B0%8F-%EC%8B%A4%EC%8A%B51
EDA	https://statkclee.github.io/ml/ml-eda.html
ELK Stack	https://classicismist.blogspot.com/2020/01/centos7-elk-stack-filebeat-logstash.html
K-means	https://ko.wikipedia.org/wiki/K-%ED%8F%89%EA%B7%A0_%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98
Maria DB	https://m.blog.naver.com/PostView.nhn?blogId=itmaster1&logNo=220243508705&proxyReferer=https:%2F%2Fwww.google.com%2F
Open API	http://data.ex.co.kr/portal/fdwn/view?type=TCS&num=C7&requestfrom=dataset
Spark - Kafka	https://eyeballs.tistory.com/210

[별첨] 산학프로젝트 산출물

산학프로젝트 산출물

1. 데이터 전처리

가. 개요

- 한국도로공사 고속도로 공공데이터 포털에서 제공하는 API 데이터를 수집하여 CSV파일을 DB에 적재한다. 적재 후 Logstash 파이프라인 구축하여 원본은 ElasticSearch에 저장하고 일부 항목은 Kafka로 발행한다.

나. 프로그램

- Python 3.7.4
- Elasticsearch, Logstash, Kibana 7.10.2
- Zookeeper 3.5.6
- Kafka 2.7.0
- MariaDB 10.4

다. 소스코드

○ 4분기 데이터를 받아 통합하여 1년치 데이터로 셋팅

```
# 분기별 데이터를 python으로 취합하여 1년치의 자료 추출
import csv
import glob

path = '/usr/local/yuna/project/'
mergefile = '/usr/local/yuna/project/2019_total_gate.csv'

files = glob.glob(path+'*')

with open(mergefile, 'w') as f:
    for i, file in enumerate (files) :
        if i == 0:
            with open(file, 'r') as f2 :
                while True :
                    line = f2.readline()

                    if not line :
                        break
                    f.write(line)

            file_name = file.split('₩₩')[-1]
            print(file.split('₩₩')[-1]+'파일 읽기 완료')
```

```
        else :
            with open(file, 'r') as f2:
                n = 0
                while True:
                    line = f2.readline()
                    if n != 0:
                        f.write(line)

                    if not line:
                        break
                    n += 1
            file_name = file.split('₩₩')[-1]
            print(file.split('₩₩')[-1] + '파일 읽기 완료')

print('파일 합체')
```

⭘ DB 적재

```
# 파이썬 코딩 후 DB로 전송
create table traffic ( 집계일자 char(50), 영업소코드 int(10), 입출구명 char(10), 총교통량 int(80));
load data infile '/usr/local/yuna/python/2019_total_gate.csv' into table traffic fields terminated by ',' enclosed by '"' lines terminated by '₩n' ignore 1 rows (@집계일자, @영업소코드, @영업소명, @입출구구분코드, @입출구명, @TCS하이패스구분코드, @TCS하이패스명, @고속도로운영기관구분코드 , @고속도로운영기관명, @영업형태구분코드, @영업형태명, @1종교통량. @2종교통량, @3종교통량, @4종교통량, @5종교통량, @6종교통량, @총교통량) set 집계일자=@집계일자, 영업소코드=@영업소코드, 입출구명=@입출구명, 총교통량=@총교통량;
```

⭘ DB 적재 후 -> ElasticSearch와 Kafka로 발행

```
# DB와 Logstash 연동을 위해 jdbc input 플러그인 사용
 input {
      jdbc {
        jdbc_driver_library => "./usr/local/yuna/mysql-connector-java-8.0.18/mysql-connector-java-8.0.18.jar"
        jdbc_driver_class => "com.mysql.jdbc.Driver"
        jdbc_connection_string => "jdbc:mysql://localhost:3306/project?serverTimezone=Asia/Seoul"
        jdbc_paging_enabled => true
        jdbc_user => "root"
        jdbc_password =>"eun8585"
        schedule => "*/60 * * * * *"
        statement => "SELECT * FROM traffic"
      }
   }


 filter {  }

# 정제된 데이터를 Kafka의 'project' 토픽에 발행
 output {
        kafka {
        bootstrap_servers => "localhost:9092"
        topic_id => "project"
        codec => json
         }
    }

      elasticsearch {
      index => "project_data"
      hosts => "localhost:9200"
       }
    }
  }'
```

ㅇ 톨게이트 교통량 오픈 API 호출 후 CSV 파일로 저장. DB에 데이터 적재

```
# Python을 이용하여 json 포맷으로 API 호출
from bs4 import BeautifulSoup
from datetime import datetime
import requests
import time
import pandas as pd
import numpy as np
import csv
import json
import scipy.io
import pymysql

def get_code(gate_code):
    url = url = "http://data.ex.co.kr/openapi/trafficapi/trafficIc?key=0756800066&type=json&tmType=2&unitCode=" + gate_code
    result = requests.get(url)
    bs_obj = BeautifulSoup(result.content, "html.parser")
    jsonobj = json.loads(str(bs_obj))
    return jsonobj

with open('/root/project01/gate.txt', 'r') as text_file:
    gate_code = text_file.readlines()
    gate_code = list(map(lambda s: s.strip(), gate_code))

while True :
    amount=[]
    now = datetime.now()
    print(now)
    for item in gate_code:
        now_amount = get_code(item)
        amount.append(now_amount)
        df = pd.DataFrame(amount)
        df = df['trafficIc']
    x = set(range(365))
    amount2=[]
    for i in x :
        df2 = df[i]
        for e in set(range(len(df2))) :
            df3 = df2[e]
            del df3['pageNo']
            del df3['numOfRows']
            del df3['tmType']
            del df3['exDivCode']
            del df3['exDivName']
            del df3['inoutType']
            del df3['tmName']
            del df3['tcsType']
            del df3['tcsName']
            del df3['carType']
            del df3['sumTm']
            amount2.append(df3)
            df4 = pd.DataFrame(amount2)
    df4 = df4.astype({'trafficAmout' : 'int'})
    df5 = df4.groupby(['unitName','unitCode','inoutName'],as_index).sum()
    df5['timestamp'] = now
    df5.to_csv('/root/project01/traffic.csv',index=False, encoding='utf8')


# MariaDB에 데이터 적재(15분 단위)
    conn = pymysql.connect(host='127.0.0.1', user='root', password='wotlr5835!', db='traffic', charset='utf8')
    curs = conn.cursor()
    with open('/root/project01/traffic.csv','r',encoding='utf8') as f :
      csvReader = csv.reader(f)
      next(csvReader)
      for row in csvReader:
        unitName = (row[0])
        unitCode = (row[1])
        inoutName = (row[2])
        trafficAmout = (row[3])
        timestamp = (row[4])
        sql = "insert into traffic (unitName, unitCode, inoutName,  trafficAmout , timestamp) values (%s, %s,  %s,  %s, %s)"
        curs.execute(sql, (unitName, unitCode, inoutName,  trafficAmout, timestamp))
      conn.commit()
      f.close()
      conn.close()
```

```
    print(df5)
    time.sleep(900)
```

```
2021-02-03 09:06:45.817785
   unitName unitCode inoutName  trafficAmout
0      가락      246       입구            50
1     가락(개)     029        입구           151
2     가락(개)     029        출구           339
3      가락2     596        출구            78
4      가산      203       입구            79
..     ...      ...       ...           ...
718     황전     557        출구            16
719     회인     532        입구            28
720     회인     532        출구             9
721     횡성     216        입구            31
722     횡성     216        출구            33

[723 rows x 4 columns]
```

▲ CSV 파일

unitName	unitCode	inoutName	trafficAmout
가락	246	입구	126
가락(개)	029	입구	231
가락(개)	029	출구	197
가락2	596	출구	144
가산	203	입구	99
가산	203	출구	106
가조	274	입구	23
가조	274	출구	20
감곡	560	입구	48
감곡	560	출구	72
강릉	189	입구	130
강릉	189	출구	133
강진무위사	982	입구	39
강진무위사	982	출구	43
거창	280	입구	52
거창	280	출구	63
건천	133	입구	42
건천	133	출구	50

▲ DB 적재

⚬ 데이터 파이프라인 구축

1) Pipeline 원본은 ElasticSearch에 저장
2) "unitName","unitCode","trafficAmout","inoutName" 는 kafka로 발행

```
# traffic.py으로 새롭게 쓰여진 traffic.csv를 Logstash에 입력
input {
       file {
         path => "/root/project01/traffic.csv"
         start_position => "beginning"
       }
}

# 데이터 필터링을 거쳐 각 줄마다 "unitName","unitCode","trafficAmout","inoutName"를 추출
filter {
        csv {
            separator => ","
            columns => ["unitName","unitCode","trafficAmout","inoutName"]
            }

        mutate{convert=>["unitName","string"]}
        mutate{convert=>["unitCode","integer"]}
        mutate{convert=>["trafficAmout","integer"]}
        mutate{convert=>["inoutName","string"]}
}

# Pipeline 원본 Elasticsearch에 저장
output {
      elasticsearch {
              action => "index"
              hosts => ["localhost:9200"]
              index => "traffic"
              }
      stdout { codec => rubydebug }
}

   kafka {
        bootstrap_servers => "localhost:9092"
        topic_id => "project"
        codec => json
         }
```

Time	_source
Feb 3, 2021 @ 09:17:20.005	unitcode: 068 trafficamout: 701 @version: 1 unitname: 골 @timestamp: Feb 3, 2021 @ 09:17:20.005 inoutname: 입 _id: q5VAZXcBkVd-6ThrK21Y _type: _doc _index: project _score: -
Feb 3, 2021 @ 09:17:20.005	unitcode: 068 trafficamout: 741 @version: 1 unitname: 골 @timestamp: Feb 3, 2021 @ 09:17:20.005 inoutname: 출 _id: rJVAZXcBkVd-6ThrK21Y _type: _doc _index: project _score: -
Feb 3, 2021 @ 09:17:20.005	unitcode: 754 trafficamout: 125 @version: 1 unitname: 평 @timestamp: Feb 3, 2021 @ 09:17:20.005 inoutname: 입 _id: rZVAZXcBkVd-6ThrK21Y _type: _doc _index: project _score: -
Feb 3, 2021 @ 09:17:20.005	unitcode: 754 trafficamout: 82 @version: 1 unitname: 평 @timestamp: Feb 3, 2021 @ 09:17:20.005 inoutname: 출 _id: rpVAZXcBkVd-6ThrK21Y _type: _doc _index: project _score: -

▲ **Logstash 확인**


```
{"unitcode":"545 ","trafficamout":56,"@version":"1","unitname":"소양","@timestamp":"2021-02-03T00:13:58.600Z","inoutname":"입구"}
{"unitcode":"545 ","trafficamout":30,"@version":"1","unitname":"소양","@timestamp":"2021-02-03T00:13:58.600Z","inoutname":"출구"}
{"unitcode":"534 ","trafficamout":28,"@version":"1","unitname":"속리산","@timestamp":"2021-02-03T00:13:58.600Z","inoutname":"입구"}
{"unitcode":"534 ","trafficamout":17,"@version":"1","unitname":"속리산","@timestamp":"2021-02-03T00:13:58.600Z","inoutname":"출구"}
{"unitcode":"186 ","trafficamout":60,"@version":"1","unitname":"속사","@timestamp":"2021-02-03T00:13:58.600Z","inoutname":"입구"}
{"unitcode":"186 ","trafficamout":6,"@version":"1","unitname":"속사","@timestamp":"2021-02-03T00:13:58.600Z","inoutname":"출구"}
{"unitcode":"702 ","trafficamout":72,"@version":"1","unitname":"속초","@timestamp":"2021-02-03T00:13:58.600Z","inoutname":"입구"}
```


▲ **Kafka 확인**

2. 이상탐지 모델(K-means)

가. 개요

- Kafka로부터 실시간으로 수신한 데이터로 Spark를 활용하여 K-means알고리즘 기반 이상탐지 모델을 생성한다. 이후 해당 모델을 기반으로 교통량 이상 결과를 시각화한다.

나. 프로그램

- Python 3.7.4
- Elasticsearch, Logstash, Kibana 7.10.2
- Zookeeper 3.5.6
- Kafka 2.7.0
- MariaDB 10.4
- Spark 2.4.7

다. 소스코드

```
# Spark 실행코드 (spark-shell 이용)
spark-shell --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.7
import org.apache.spark.sql.functions.udf
val df = spark.readStream.format("kafka").option("kafka.bootstrap.servers", "localhost:9092").option("subscribe", "project").option("startingOffsets", "earliest").load()
val toStr = udf((payload: Array[Byte]) => new String(payload))
val df2 = df.withColumn("value", toStr(kd("value")))
df.printSchema()
val df3 = df2.selectExpr("CAST(value AS STRING)")
import org.apache.spark.sql.types.{StructType, StructField, StringType, IntegerType};
val schema = new StructType().add("@version", StringType).add("unitcode",StringType).add("trafficamout",IntegerType).add("unitname",StringType).add("@timestamp",StringType).add("inoutname",StringType)
val df4 = df3.select(from_json(col("value"), schema).as("data")).select("data.*")
df4.writeStream.format("console").outputMode("append").start().awaitTermination()
```