1.1 강의 진행 방식

수평 학습하고 왜 프레임워크가 등장했는지 이해하자.

1.2 MLOps 개론

Research와 Production의 차이

	Research ML	Production ML
데이터	고정(Static)	계속 변함(Dynamic - Shifting)
중요 요소	모델 성능(Accuracy, RMSE 등)	모델 성능, 빠른 Inference 속도, 해석 가능함
도전 과제	더 좋은 성능을 내는 모델(SOTA), 새로운 구조의 모델	안정적인 운영, 전체 시스템 구조
학습	(데이터는 고정) 모델 구조, 파라미터 기반 재학습	시간의 흐름에 따라 데이터가 변경되어 재학습
목적	논문 출판	서비스에서 문제 해결
표현	Offline	Online

Serving

- Batch Serving : 많은 양(=많은 데이터)을 일정 주기(1일, 1주, 1달 등)로 한꺼번에 음식 서빙(=예측) Jupyter Notebook에서 실행하는 방식은 대부분 Batch Serving으로 쉽게 변경 가능(Dataframe의 데이터를 한번에 예측)
- Online Serving : 한번에 하나씩(=실시간으로) 포장해서 배송(=예측) 동시에 여러 주문이 들어올 경우 병목이 없어야 하고, 확장 가능하도록 준비해야 함

tutorial code

[ ] *숨겨진 셀 4개*

Special Mission

1. MLOps가 필요한 이유 이해하기
    - MLOps는 머신러닝 모델을 운영하면서 반복적으로 필요한 업무를 자동화시키는 과정이고 모델링에 집중할 수 있도록 관련된 인프라를 만들고, 자동으로 운영되도록 만든다.
2. MLOps의 각 Component에 대해 이해하기(왜 이런 Component가 생겼는가?)
    1. Infra : 클라우드, 온프레미스(전산실 서버)
    2. Serving : 모델 서빙
    3. Experiment, Model Management : 모델링을 위한 기록 / mlflow
    4. Feature Store : 머신러닝 Feature를 집계 / feast
    5. Data Validation : 데이터셋 별 데이터 분포 및 퀄리티 파악 / TensorFlow Data Validation, AWS Deequ
    6. Continuous Training : CI/CD
    7. Monitoring
    8. AutoML
3. MLOps 관련된 자료, 논문 읽어보며 강의 내용 외에 어떤 부분이 있는지 파악해보기
    - Continuous Monitoring
4. MLOps Component 중 내가 매력적으로 생각하는 TOP3을 정해보고 왜 그렇게 생각했는지 작성해보기
    - Serving : 서비스에 필수
    - Continuous Training : 학습 파이프라이닝이 편해져서
    - AutoML : 모든 엔지니어의 꿈, 자동화

1.3 Model Serving

Serving vs Inference

- Serving : 모델을 웹/앱 서비스에 배포하는 과정, 모델을 활용하는 방식, 모델을 서비스화하는 관점
- Inference : 모델에 데이터가 제공되어 예측하는 경우, 사용하는 관점

Server

- Web Server는 Client의 다양한 요청을 처리해주는 역할
- Machine Learning Server는 Client의 다양한 요청을 처리해주는 역할 (데이터 전처리, 모델을 기반으로 예측 등)

API(Application Programming Interface)

- 운영체제나 프로그래밍 언어가 제공하는 기능을 제어할 수 있게 만든 인터페이스

Online Serving

Online Serving 정의

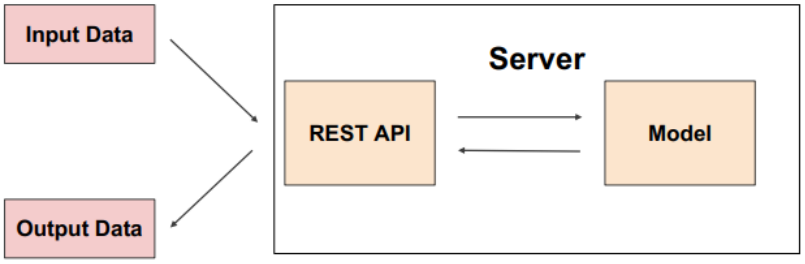
- 요청(Request)이 올 때마다 실시간으로 예측
- 클라이언트(애플리케이션)에서 ML 모델 서버에 HTTP 요청(Request)하고, 머신러닝 모델 서버에서 예측한 후, 예측 값(응답)을 반환(Response)

Online Serving 구현 방식

1. 직접 API 웹 서버 개발 : Flask, FastAPI 등을 사용해 서버 구축
2. 클라우드 서비스 활용 : AWS의 SageMaker, GCP의 Vertex AI 등
3. Serving 라이브러리 활용 : Tensorflow Serving, Torch Serve, MLFlow, BentoML 등

Online Serving 고려할 점

- 실시간 예측을 하기 때문에 예측할 때 지연 시간(Latency)를 최소화해야 함
    - Latency : 하나의 예측을 요청하고 반환값을 받는데까지 걸리는 시간
- Serving 할 때 Python 버전, 패키지 버전 등 Dependency가 굉장히 중요
- Input 데이터를 기반으로 Database에 있는 데이터를 추출해서 모델 예측해야 하는 경우
- 모델이 수행하는 연산
- 결과 값에 대한 보정이 필요한 경우



Batch Serving

Batch Serving 정의

- Batch Serving은 주기적으로 학습을 하거나 예측을 하는 경우
- 실시간이 필요 없는 대부분의 방식에서 활용 가능

Batch Serving 특징

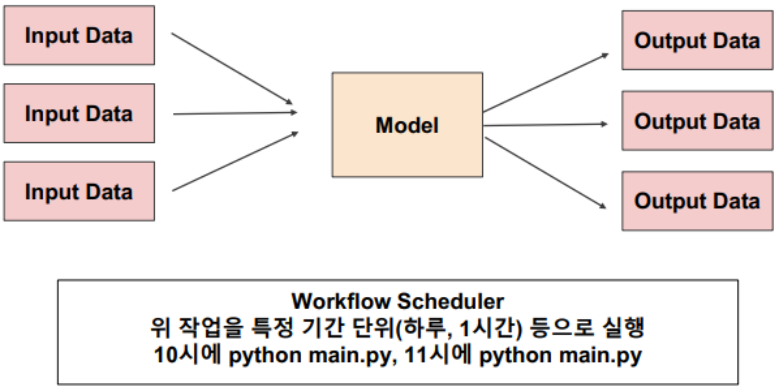
- 학습 / 예측을 별도의 작업으로 설정
- Airflow, Cron Job 등으로 스케쥴링 작업(Workflow Scheduler)

Batch Serving 장점

- Jupyter Notebook에 작성한 코드를 함수화한 후, 주기적으로 실행하는 간단한 구조!
- Online Serving보다 구현이 수월하며, 간단함
- 한번에 많은 데이터를 처리하므로 Latency가 문제되지 않음

Batch Serving 장점

- 실시간으로 활용할 수 없음
- Cold Start 문제 : 오늘 새로 생긴 컨텐츠는 추천할 수 없음



Special Mission

1. Rules of Machine Learning: Best Practices for ML Engineering 문서 읽고 정리하기!
    - 원문 : https://developers.google.com/machine-learning/guides/rules-of-ml?hl=ko
    - 한글 요약 : https://zzsza.github.io/data/2019/12/15/rules-of-ml/
2. Online Serving / Batch Serving 기업들의 Use Case 찾아서 정리하기 (어떤 방식으로 되어 있는지 지금은 이해가 되지 않아도 문서를 천천히 읽고 정리하기)
    - Online Serving : 유튜브 추천 알고리즘
    - Batch Serving : 스포티파이의 예측 알고리즘(Discover Weekly)

1.4 머신러닝 프로젝트 라이프 사이클

머신러닝 프로젝트

프로젝트 설계

현실적인 머신러닝 프로젝트 과정

- 문제 정의
- 최적화할 Metric 선택
- 데이터 수집, 레이블 확인
- 모델 개발
- 모델 예측 결과를 토대로 Error Analysis. 잘못된 라벨이 왜 생기는지 확인
- 다시 모델 학습
- 더 많은 데이터 수집
- 다시 모델 학습
- 2달 전 테스트 데이터에선 성능이 좋지만 어제 데이터엔 성능이 좋지 않음
- 모델을 다시 학습함
- 모델 배포
- 최적화할 Metric이 실제로 잘 동작하지 않아 Metric을 수정
- 다시 시작 ㅠ

문제 정의에 기반해서 프로젝트 설계

- 해결하려고 하는 문제 구체화
- 머신러닝 문제 타당성 확인
- 목표 설정, 지표 결정
- 제약 조건(Constraint & Risk)
- 베이스라인, 프로토타입
- 평가(Evaluation) 방법 설계

머신러닝 문제 타당성 확인

- 패턴 : 학습할 수 있는 패턴이 있는가?
- 목적 함수 : 학습을 위한 목적 함수를 만들 수 있어야 함
- 복잡성 : 패턴이 복잡해야 함

- 데이터 존재 여부 : 데이터가 존재하거나 수집할 수 있어야 함
- 반복 : 사람이 반복적으로 실행하는 경우

머신러닝이 사용되면 좋지 않은 경우

- 비윤리적인 문제
- 간단히 해결할 수 있는 경우
- 좋은 데이터를 얻기 어려울 경우
- 한번의 예측 오류가 치명적인 결과를 발생할 경우
- 시스템이 내리는 모든 결정이 설명 가능해야 할 경우
- 비용 효율적이지 않은 경우

Objective가 여러개인 경우 분리하는 것이 좋음

- 학습하기 쉬워야 함
    - 하나의 objective를 최적화하는 것이 여러 objectives를 최적화하는 것보다 쉬움
- 모델을 재학습하지 않도록 모델을 분리
- Objectives는 수정해야 하는 유지보수 일정이 모두 다를 수 있음
    - 예) 스팸 필터링은 품질 순위 시스템보다 더 빠르게 업데이트해야 함

Metric Evaluation

- 이 부분은 작게는 모델의 성능 지표(RMSE)일 수 있고,
- 크게는 비즈니스의 지표일 수 있음(고객의 재방문율, 매출 등)

비즈니스 모델

BM 파악하기

1. 회사의 비즈니스 파악하기 : 회사가 어떤 서비스, 가치를 제공하고 있는가?
    - 추천 도서 : 한 장으로 끝내는 비즈니스 모델 100
2. 데이터를 활용할 수 있는 부분은 어디인가? (Input)
3. 모델을 활용한다고 하면 예측의 결과가 어떻게 활용되는가? (Output)
    - 더 좋은 가치 제공, 매출 증대, 업무 자동화

용어

- 대기 시간(ETA, Estimated time of Arrival)

기술블로그

- https://eng.uber.com/

2.1 프로토타이핑 - Notebook 베이스(Voila)

Voila

Voila

- 대시보드
- 2019년 12월에 Jupyter의 하위 프로젝트로 통합됨

Voila 장점

1. Jupyter Notebook 결과를 쉽게 웹 형태로 띄울 수 있음
2. Ipywidget, Ipyleaflet 등 사용 가능
3. Jupyter Notebook의 Extension 있음(=노트북에서 바로 대시보드로 변환 가능)
4. Python, Julia, C++ 코드 지원
5. 고유한 템플릿 생성 가능
6. 너무 쉬운 러닝커브

Voila 사용시 TIP

- voila voila_basic.ipynb --strip_sources=False
    - --strip_sources=False과 함께 실행하면 Voila에서 코드도 보임(직접 실행은 불가능)
- Voila는 유저별로 새로운 Notebook Kernel을 실행시키는 구조

- Voila 노트북을 사용하지 않을 때 자동으로 종료해야 함
- Jupyter Notebook의 Config에서 cull 옵션
  - idle 상태인 경우 cull(끄는 행위)
  - cull_interval : idle 커널을 확인할 간격(초)
  - cull_idle_timeout : 커널을 idle 상태로 판단할 기준(초). 이 시간동안 이벤트가 없으면 idle로 판단

```
voila voila_basic.ipynb --MappingKernelManager.cull_interval=60 --MappingKernelManager.cull_idle_timeout=300
```

- Voila 셀 타임아웃 제한, 기본 30초

```
voila --ExecutePreprocessor.timeout=180
jupyter notebook --ExecutePreprocessor.timeout=180
```

- Voila에서 nbextension을 사용하고 싶은 경우

```
voila your_notebook.ipynb --enable_nbextensions=True
```

- jupyter notebook 실행 시에도 인자를 넘길 수 있음

```
jupyter notebook --ExecutePreprocessor.timeout=180 --VoilaConfiguration.enable_nbextensions=True
```

Jupyter Notebook의 passwd를 사용

1. Jupyter Notebook의 설정 파일 생성하기(있다면 Skip)
   - `jupyter notebook --generate-config`
2. 터미널에서 python 실행 후 아래 코드 실행

```
from IPython.lib import passwd
passwd()
```

3. sha1로 된 암호가 나타남. 'sha1~~'값 복사
4. 아까 생성된 jupyter notebook config로 진입
   - `vi ~/.jupyter/jupyter_notebook_config.py`
5. c.NotebookApp.password를 찾아서 우측에 복사한 sha1값을 붙여넣기
6. ESC :wq로 저장하고 나오기

ipywidget

Tutorial code

숨겨진 셀 47개

Voila

Load github files

숨겨진 셀 4개

Tutorial code

숨겨진 셀 16개

2.2 프로토타이핑 - 웹 서비스 형태(Streamlit)

Voila의 특징

- 노트북에서 쉽게 프로토타입을 만들 수 있음
- 단, 대시보드처럼 레이아웃을 잡기 어려움 => 이런 경우 웹 개발 진행할 수 있음

Streamlit 장점

- 파이썬 스크립트 코드를 조금만 수정하면 웹을 띄울 수 있음
- 백엔드 개발이나 HTTP 요청을 구현하지 않아도 됨
- 다양한 Component 제공해 대시보드 UI 구성할 수 있음
- Streamlit Cloud도 존재해서 쉽게 배포할 수 있음(단, Community Plan은 Public Repo만 가능)
- 화면 녹화 기능(Record) 존재

Streamlit Gallery

- https://streamlit.io/gallery

Tutoral code

숨겨진 셀 32개

2.3 Linux & Shell Command

1. Linux

Linux를 알아야 하는 이유 Linux

- 서버에서 자주 사용하는 OS
  - Mac, Window도 서버로 활용은 가능하나 유료
- Free, 오픈소스
  - 여러 버전이 존재 => 여러분들의 버전을 만들 수도 있음
- 안정성, 신뢰성. 유닉스라 Stability, Reliability
- 쉘 커맨드, 쉘 스크립트

대표적인 Linux 배포판 Linux

Debian

- 온라인 커뮤니티에서 제작해 배포

Ubuntu

- 영국의 캐노니컬이라는 회사에서 만든 배포판으로 쉽고 편한 설치
- 초보자들이 쉽게 접근할 수 있도록 만듬

Redhat

- 레드햇이라는 회사에서 배포한 리눅스

CentOS

- Red Hat이 공개한 버전을 가져와서 브랜드와 로고를 제거하고 배포한 버전

Linux를 사용하는 방법

- VirtualBox에 Linux 설치, Docker로 설치
- WSL 사용(윈도우)
- Notebook에서 터미널 실행
- 2.4 Cloud에서 띄우는 인스턴스에서 연습

2. Shell Command

쉘

- 사용자가 문자를 입력해 컴퓨터에 명령할 수 있도록 하는 프로그램

터미널/콘솔

- 쉘을 실행하기 위해 문자 입력을 받아 컴퓨터에 전달
- 프로그램의 출력을 화면에 작성

sh

- 최초의 쉘

bash

- Linux 표준 쉘

zsh

- Mac 카탈리나 OS 기본 쉘

쉘 UX

- username@hostname:current_folder
  - hostname : 컴퓨터 네트워크에 접속된 장치에 할당된 이름
  - IP 대신 기억하기 쉬운 글자로 저장

쉘을 사용하는 경우

- 서버에서 접속해서 사용하는 경우
- crontab 등 Linux의 내장 기능을 활용하는 경우
- 데이터 전처리를 하기 위해 쉘 커맨드를 사용
- Docker를 사용하는 경우
- 수백대의 서버를 관리할 경우
- Jupyter Notebook의 Cell에서 앞에 !를 붙이면 쉘 커맨드가 사용됨
- 터미널에서 python3, jupyter notebook 도 쉘 커맨드
- Test Code 실행
- 배포 파이프라인 실행(Github Action 등에서 실행)

필수 쉘 커맨드

man

- 쉘 커맨드의 매뉴얼 문서를 보고 싶은 경우
- man python
- 종료 : `:q` 입력

mkdir

- 폴더 생성하기 : Make Directory
- mkdir linux-test

ls

- 현재 접근한 폴더의 폴더, 파일 확인 : List Segments
- ls 뒤에 아무것도 작성하지 않으면 현재 폴더 기준으로 실행 폴더를 작성하면 폴더 기준에서 실행
- 옵션
  - -a : .으로 시작하는 파일, 폴더를 포함해 전체 파일 출력
  - -l : 퍼미션, 소유자, 만든 날짜, 용량까지 출력
  - -h : 용량을 사람이 읽기 쉽도록 GB, MB 등 표현. `-l` 과 같이 사용
    - ls ~
    - ls
    - ls -al
    - ls -lh

pwd

- 현재 폴더 경로를 절대 경로로 보여줌 : Print Working Directory
- pwd

cd

- 폴더 변경하기, 폴더로 이동하기 : Change Directory
- cd linux-test

echo

- Python의 print처럼 터미널에 텍스트 출력
- echo “hi”
- echo `쉘 커맨드` 입력시 쉘 커맨드의 결과를 출력. ` : 1 왼쪽에 있는 backtick
- echo `pwd`

vi

- vim 편집기로 파일 생성
    - INSERT 모드에서만 수정할 수 있음
- vi vi-test.sh (새로운 창이 뜨면) i 를 눌러서 INSERT 모드로 변경
- 그 후 echo “hi” 작성
- ESC를 누른 후 :wq (저장하고 나가기, write and quit)
- ESC :wq! : 강제로 저장하고 나오기
- ESC :q : 그냥 나가기

vi 편집기의 Mode

Command Mode

- vi 실행시 기본 Mode
- 방향키를 통해 커서를 이동할 수 있음
- dd : 현재 위치한 한 줄 삭제
- i : INSERT 모드로 변경
- x : 커서가 위치한 곳의 글자 1개 삭제(5x : 문자 5개 삭제)
- yy : 현재 줄을 복사(1줄을 ctrl + c)
- p : 현재 커서가 있는 줄 바로 아래에 붙여넣기
- k : 커서 위로
- j : 커서 아래로
- l : 커서 오른쪽으로
- h : 커서 왼쪽으로

Insert Mode

- 파일을 수정할 수 있는 Mode
- 만약 Command Mode로 다시 이동하고 싶다면 ESC 입력

Last Line Mode

- ESC를 누른 후 콜론(:)을 누르면 나오는 Mode
- w : 현재 파일명으로 저장
- q : vi 종료(저장되지 않음)
- q! : vi 강제 종료(!는 강제를 의미)
- wq : 저장한 후 종료
- /문자 : 문자 탐색
    - 탐색한 후 n을 누르면 계속 탐색 실행
- set nu : vi 라인 번호 출력

bash

- bash로 쉘 스크립트 실행
- bash vi-test.sh
- 앞에서 작성한 “hi”가 출력
- 터미널에서 Tab을 누르면 자동완성(지원하지 않는 쉘도 존재)

sudo

- 관리자 권한으로 실행하고 싶은 경우 커맨드 앞에 sudo를 붙임
- sudo 명령어 : 최고 권한을 가진 슈퍼 유저로 프로그램을 실행
- “superuser do”에서 유래하고, 최근엔 “substitute user do”(다른 사용자의 권한으로 실행)의 줄임말로 해석

cp

- 파일 또는 폴더 복사하기 : Copy
- cp vi-test.sh vi-test2.sh
- -r : 디렉토리를 복사할 때 디렉토리 안에 파일이 있으면 recursive(재귀적)으로 모두 복사
- -f : 복사할 때 강제로 실행

mv

- 파일, 폴더 이동하기(또는 이름 바꿀 때도 활용) : Move
- mv vi-test.sh vi-test3.sh

cat

- 특정 파일 내용 출력 : concatenate
  - cat vi-test.sh
- 여러 파일을 인자로 주면 합쳐서(CONCAT) 출력
  - cat vi-test2.sh vi-test3.sh
- 파일에 저장하고(OVERWRITE) 싶은 경우
  - cat vi-test2.sh vi-test3.sh > new_test.sh
- 파일에 추가(APPEND) 싶은 경우
  - cat vi-test2.sh vi-test3.sh >> new_test.sh

clear

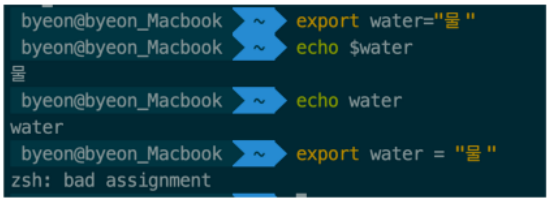
- 터미널 창을 깨끗하게 해줌

find

- 파일 및 디렉토리를 검색할 때 사용
- find . -name “File” : 현재 폴더에서 File이란 이름을 가지는 파일 및 디렉토리 검색

export

- export로 환경 변수 설정
- export water=”물"
- echo $water
- export로 환경 변수 설정한 경우, 터미널이 꺼지면 사라지게 됨
  - 매번 쉘을 실행할 때마다 환경변수를 저장하고 싶으면 .bashrc, .zshrc에 저장하면 됨
  - (Linux) vi ~/.bashrc 또는 vi ~/.zshrc (자신이 사용하는 쉘에 따라 다름)
  - 제일 하단에 export water=”물"을 저장하고 나옴(ESC :wq)
  - 그 후 source ~/.bashrc 또는 source ~/.zshrc
    - Linux 환경 설정을 재로그인하지 않고 즉시 적용하고 싶은 경우 source 사용



alias

- 터미널에서 alias라고 치면 현재 별칭으로 설정된 것을 볼 수 있음
- alias는 기본 명령어를 간단히 줄일 수 있는 것
  - 예) ll는 ls -l로 별칭이 지정되어 있음
- alias ll2=’ls -l’
- ll2를 입력하면 ls -l이 동작됨

기타 쉘 커맨드

head, tail

- 파일의 앞/뒤 n행 출력
- head -n 3 vi-test.sh

sort

- 행 단위 정렬
- -r : 정렬을 내림차순으로 정렬(Default 옵션 : 오름차순)
- -n : Numeric Sort

```
vi fruits.txt
banana
orange
apple
apple
```

```
orange
orange
apple
banana
```

- ESC :wq로 저장

```
cat fruits.txt | sort
cat fruits.txt | sort -r
```

uniq

- 중복된 행이 연속으로 있는 경우 중복 제거
- sort와 함께 사용
- -c : 중복 행의 개수 출력

```
cat fruits.txt | uniq
cat fruits.txt | sort | uniq
cat fruits.txt | uniq | wc -l
cat fruits.txt | sort | uniq | wc -l
```

grep

- 파일에 주어진 패턴 목록과 매칭되는 라인 검색
- grep 옵션 패턴 파일명
- 옵션
  - -i : Insensitively하게, 대소문자 구분 없이 찾기
  - -w : 정확히 그 단어만 찾기
  - -v: 특정 패턴 제외한 결과 출력
  - -E : 정규 표현식 사용
- 정규 표현식 패턴
  - ^단어 : 단어로 시작하는 것 찾기
  - 단어$: 단어로 끝나는 것 찾기
  - . : 하나의 문자 매칭

cut

- 파일에서 특정 필드 추출
- -f : 잘라낼 필드 지정
- -d : 필드를 구분하는 구분자. Default는 \t

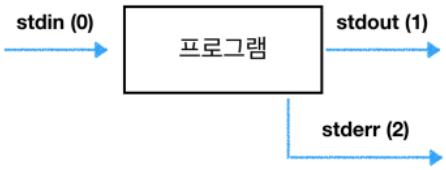
```
vi cut_file
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

ESC :wq

```
cat cut_file | cut -d : -f 1,7
# 1번째, 7번째 값을 가져옴
```

표준 스트림(Stream)

- Unix에서 동작하는 프로그램은 커맨드 실행시 3개의 Stream이 생성
  - stdin : 0으로 표현, 입력(비밀번호, 커맨드 등)
  - stdout : 1로 표현, 출력 값(터미널에 나오는 값)
  - stderr : 2로 표현, 디버깅 정보나 에러 출력



Redirection & Pipe

- Redirection : 프로그램의 출력(stdout)을 다른 파일이나 스트림으로 전달

```
# > : 덮어쓰기(Overwrite) 파일이 없으면 생성하고 저장
# >> : 맨 아래에 추가하기(Append)
echo “hi” > vi-test3.sh
echo “hello” >> vi-test3.sh
cat vi-test3.sh
```

- Pipe : 프로그램의 출력(stdout)을 다른 프로그램의 입력으로 사용하고 싶은 경우
- A의 Output을 B의 Input으로 사용(다양한 커맨드를 조합)
- 현재 폴더에 있는 파일명 중 vi가 들어간 단어를 찾고 싶은 경우
  - `ls | grep “vi”`
    - grep “vi” : 특정 단어 찾기
- 위 결과를 다시 output.txt에 저장하고 싶은 경우 `ls | grep “vi” > output.txt`
- 최근 입력한 커맨드 중 echo가 들어간 명령어를 찾고 싶은 경우 `history | grep “echo”`

Redirection & Pipe 연습 문제

- test.txt 파일에 "Hi!!!!"를 입력해주세요(vi 사용 금지)
- test.txt 파일 맨 아래에 "kkkkk"를 입력해주세요(vi 사용 금지)
- test.txt의 라인 수를 구해주세요(힌트 : wc -l를 쓰면 라인 수를 구할 수 있음)

서버에서 자주 사용하는 쉘 커맨드

ps

- 현재 실행되고 있는 프로세스 출력하기 : Process Status
- -e : 모든 프로세스
- -f : Full Format으로 자세히 보여줌

curl

- Command Line 기반의 Data Transfer 커맨드 : Client URL
- Request를 테스트할 수 있는 명령어
- 웹 서버를 작성한 후 요청이 제대로 실행되는지 확인할 수 있음
- curl -X localhost:5000/ {data}
- curl 외에 httpie 등도 있음(더 가독성있게 출력)

df

- 현재 사용 중인 디스크 용량 확인 : Disk Free
- -h : 사람이 읽기 쉬운 형태로 출력

scp

- SSH을 이용해 네트워크로 연결된 호스트 간 파일을 주고 받는 명령어 : Secure Copy(Remote file copy program)
- -r : 재귀적으로 복사
- -P : ssh 포트 지정
- -i : SSH 설정을 활용해 실행

```
#  local => remote 전달
scp local_path user@ip:remote_directory
# remote => local 전달
scp user@ip:remote_directory local_path
```

```
# remote => remote 전달
scp user@ip:remote_directory user2@ip2:target_remote_directory
```

nohup

- 터미널 종료 후에도 계속 작업이 유지하도록 실행(백그라운드 실행)
- `nohup python3 app.py &`
- nohup으로 실행될 파일은 Permission이 755여야 함
- nohup 끄려면 ps써서 pid kill
- 종료는 ps ef | grep app.py 한 후, pid(Process ID) 찾은 후 kill -9 pid 로
  - 프로세스를 Kill
  - 로그는 nohup.out에 저장됨
- nohup 외에도 screen이란 도구도 있음

chmod

- 파일의 권한을 변경하는 경우 사용 : Change Mode
- 유닉스에서 파일이나 디렉토리의 시스템 모드를 변경함
- ls -al(혹은 ll)을 입력하면 다음과 같이 나옴

Permission

- r = Read(읽기), 4
- w = Write(쓰기), 2
- x = eXecute(실행하기), 1
-
  - = Denied
- r-x : 읽거나 실행할 수는 있지만 수정은 불가능

755, 644로 퍼미션을 주세요! 라고 하는 경우가 존재

- rwx를 더하면 4+2+1 = 7

쉘 스크립트

- .sh 파일을 생성하고, 그 안에 쉘 커맨드를 추가
- 파이썬처럼 if, while, case 문이 존재하며 작성시 bash name.sh로 실행 가능

쉘 스크립트 예시

- 쉘 스크립트 = 쉘 커맨드의 조합
- 쉘 커맨드에 익숙해진 후, 스크립트로 생성(Python과 유사)
- 지금은 쉘 커맨드에 많이 익숙해진 후, 추후에 하나씩 만들어보기(쉘 커맨드가 항상 베이스)
- #!/bin/bash : Shebang, 이 스크립트를 Bash 쉘로 해석
- $(date +%s) : date를 %s(unix timestamp)로 변형
  - START=$(date +%s) : 변수 저장

쉘 스크립트 참고자료

- https://github.com/zzsza/shell-scripts
- https://github.com/denysdovhan/bash-handbook
- https://github.com/epety/100-shell-script-examples

13 lines (9 sloc) | 193 Bytes

```
#!/bin/bash
START=$(date +%s)

echo "Calculate run-time"

# put your script
sleep 3

END=$(date +%s)
DIFF_SECOND=$(( $END - $START ))
DIFF_MINUTE=$(( $DIFF_SECOND / 60 ))

echo ${DIFF_SECOND}
```

Special Mission

1. 학습한 쉘 커맨드 정리하기
2. 카카오톡 그룹 채팅방에서 옵션 - 대화 내보내기로 csv로 저장 후, 쉘 커맨드 1줄로 카카오톡 대화방에서 2021년에 제일 메세지를 많이 보낸 TOP 3명 추출하기!

2.4 Cloud

1. Cloud

1.1 Cloud 서비스를 사용하는 이유

전통적인 배포 방법

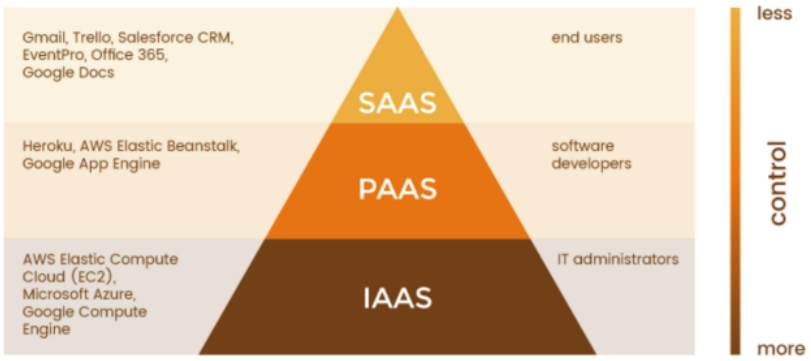
- 물리적 공간, 확장성을 고려한 서버실을 만들고 운영
- IDC(Internet Data Center)
  - 서버 컴퓨터를 넣을 공간 + 추후 서버를 추가할 때 즉각적으로 확장할 수 있는지
  - 전기, 에어컨 등 서버가 급작스럽게 종료되지 않도록 준비가 필요함

갑자기 트래픽이 몰릴 경우, 컴퓨터 10대를 설치하기 어려움(자재 수급 이슈 등) 반대로 트래픽이 적어서 컴퓨터 10대를 없애기가 수월하지 않음

앞선 내용보다 자유롭게 활용할 수 있는 개념으로 클라우드 서비스가 점점 발전 그 이후엔 개발자가 직접 설정해야 했던 작업 등을 클라우드에서 쉽게 할 수 있는 방향으로 발전 (Managed 서비스)

- Apache Spark를 쉽게 운영할 수 있도록 AWS EMR, GCP Dataproc 등을 활용 => 직접 하둡을 설치할 필요 없이 이미 설치되어 있음
- Jupyter Notebook 환경도 미리 설치해두고 사용할 수 있음(Tensorflow, 쿠다 등)

다양한 Service



1.2 Cloud 서비스의 다양한 제품

Computing Service(Server)

- 연산을 수행하는(Computing) 서비스
- 가상 컴퓨터, 서버, VM, Instance라고 불림
- CPU, Memory, GPU 등을 선택할 수 있음
- 가장 많이 사용할 제품
- 인스턴스 생성 후, 인스턴스에 들어가서 사용 가능
- 회사별로 월에 무료 사용량이 존재(성능은 약 cpu 1 core, memory 2G)

Serverless Computing

- 앞에 나온 Computing Service와 유사하지만, 서버 관리를 클라우드쪽에서 진행
- 코드를 클라우드에 제출하면, 그 코드를 가지고 서버를 실행해주는 형태
- 요청 부하에 따라 자동으로 확장(Auto Scaling)
- Micro Service로 많이 활용
- 예 : AWS Lambda

Stateless Container

- Docker를 사용한 Container 기반으로 서버를 실행하는 구조
- Docker Image를 업로드하면 해당 이미지 기반으로 서버를 실행해주는 형태
- 요청 부하에 따라 자동으로 확장(Auto Scaling)

Object Storage

- 다양한 Object를 저장할 수 있는 저장소
- 다양한 형태의 데이터를 저장할 수 있으며, API를 사용해 데이터에 접근할 수 있음
- 점점 데이터 저장 비용이 저렴해지고 있음
- 머신러닝 모델 pkl 파일, csv 파일, 실험 log 등을 Object Storage에 저장할 수 있음

Database(RDB)

- Database가 필요한 경우 클라우드에서 제공하는 Database를 활용할 수 있음
- 웹, 앱서비스와 데이터베이스가 연결되어 있는 경우가 많으며, 대표적으로 MySQL, PosgreSQL 등을 사용할 수 있음
- 사용자 로그 데이터를 Database에 저장할 수도 있고, Object Storage에 저장할 수도 있음. 저장된 데이터를 어떻게 사용하냐에 따라 어디에 저장할지를 결정

Data Warehouse

- Database에 저장된 데이터는 데이터 분석을 메인으로 하는 저장소가 아닌 서비스에서 활용할 Database
- Database에 있는 데이터, Object Storage 에 있는 데이터 등을 모두 모아서 Data Warehouse에 저장
- 데이터 분석에 특화된 Database

AI Platform

- AI Research, AI Develop 과정을 더 편리하게 해주는 제품
- MLOps 관련 서비스 제공
- Google Cloud Platform : TPU

	AWS	GCP	Azure
Computing Service (Server)	Elastic Compute (EC2)	Compute Engine	Virtual Machine
Serverless Computing	Lambda	Cloud Function	Azure function
Stateless Container	ECS	Cloud Run	Container Instance
Object Storage	S3	Cloud Storage	Blob Storage
Database(RDB)	Amazon RDS	Cloud SQL	Azure SQL
Data Warehouse	Redshift	BigQuery	Synapse Analytics
AI Platform	SageMaker	Vertex AI	Azure Machine Learning
Kubernetes	EKS (Elastic Kubernetes Service)	GKE (Google Kubernetes Engine)	AKS (Azure Kubernetes Service)

2. Google Cloud Platform

GCP를 선택한 이유

- 첫 가입시 $300 크레딧 제공

2.1 Google Cloud Platform 프로젝트 생성하기

Free Tier : 무료 사용이 가능한 범위

- https://cloud.google.com/free

Region을 서울로 지정 머신 유형을 e2-micro로 지정

- (무료로 사용 가능한 성능)

2.2 Google Cloud Platform Compute Engine

- 사용하지 않는 경우엔 중지 또는 삭제!
- 중지되는 경우에도 비용이 부과되는 경우가 존재할 수 있음

2.3 Google Cloud Platform Cloud Storage

Python Cloud Storage API를 사용해 Cloud Storage에 업로드한 파일을 파이썬에서 사용하는 코드 참고자료

- https://googleapis.dev/python/storage/latest/index.html

2.5 Github Action을 활용한 CI/CD

1. CI/CD

1.1 현업 개발 프로세스

개발 환경

Local

- 각자의 컴퓨터에서 개발
- 각자의 환경을 통일시키기 위해 Docker 등을 사용

Dev

- Local에서 개발한 기능을 테스트할 수 있는 환경
- Test 서버

Staging

- Production 환경에 배포하기 전에 운영하거나 보안, 성능 측정하는 환경
- Staging 서버

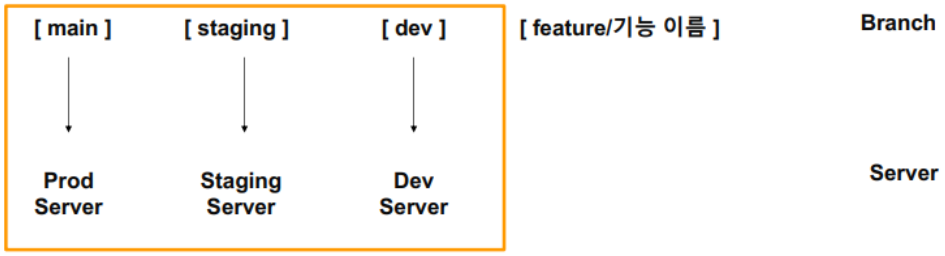
Production

- 실제 서비스를 운영하는 환경
- 운영 서버

개발 환경을 나누는 이유

- 실제 운영중인 서비스에 장애가 발생하면 안됨

- Git Flow



- 1.2 CI/CD 개념

Continuous Integration, 지속적 통합

- 새롭게 작성한 코드 변경 사항이 Build, Test 진행한 후 Test Case에 통과했는지 확인
- 지속적으로 코드 품질 관리
- 10명의 개발자가 코드를 수정했다면 모두 CI 프로세스 진행

Continuous Deploy/Delivery, 지속적 배포

- 작성한 코드가 항상 신뢰 가능한 상태가 되면 자동으로 배포될 수 있도록 하는 과정
- CI 이후 CD를 진행
- dev / staging / main 브랜치에 Merge가 될 경우 코드가 자동으로 서버에 배포

CI vs CD

- CI : 빌드, 테스트 자동화
- CD : 배포 자동화

2. Github Action

2.1 Github Action 소개

Workflow 예시

1. Test Code
    - 특정 함수의 return 값이 어떻게 나오는지 확인하는 Test Code
    - 특정 변수의 타입이 int가 맞는가?
    - Unit Test, End to End Test
2. 배포
    - Prod, Staging, Dev 서버에 코드 배포
    - FTP로 파일 전송할 수도 있고, Docker Image를 Push하는 방법 등
    - Node.js 등 다양한 언어 배포도 지원
3. 파이썬, 쉘 스크립트 실행
    - Github Repo에 저장된 스크립트를 일정 주기를 가지고 실행
    - crontab의 대용
    - 데이터 수집을 주기적으로 해야할 경우 활용할 수도 있음
    - 참고자료 : https://github.com/actions/setup-python
4. Github Tag, Release 자동으로 설정
    - Main 브랜치에 Merge 될 경우에 특정 작업 실행
    - 기존 버전에서 버전 Up하기
    - 새로운 브랜치 생성시 특정 작업 실행도 가능

Workflow 템플릿을 공유 가능, 원하는 기능이 있는 경우 <기능> github action 등으로 검색!

- Action Marketplace : https://github.com/marketplace?type=actions
- Awesome Github Action : https://github.com/sdras/awesome-actions

- 하나의 Github Repository 당 Workflow는 최대 20개까지 등록할 수 있음
- Workflow에 존재하는 Job(실행)은 최대 6시간 실행할 수 있으며, 초과시 자동으로 중지됨
- 동시에 실행할 수 있는 Job 제한 존재

제약 조건

- 하나의 Github Repository 당 Workflow는 최대 20개까지 등록할 수 있음
- Workflow에 존재하는 Job(실행)은 최대 6시간 실행할 수 있으며, 초과시 자동으로 중지됨
- 동시에 실행할 수 있는 Job 제한 존재

사용 방식

1. 코드 작업
2. 코드 작업 후, Github Action으로 무엇을 할 것인지 생각
3. 사용할 Workflow 정의
4. Workflow 정의 후 정상 작동하는지 확인

Github Action Core

- 핵심 개념 : Workflow, Event, Job, Step, Action, Runner

1. Workflow
    - 여러 Job으로 구성되고 Event로 Trigger(실행)되는 자동화된 Process
    - 최상위 개념
    - Workflow 파일은 YAML으로 작성되고, Github Repository의 .github/workflows 폴더에 저장
2. Event
    - Workflow를 Trigger하는 특정 활동, 규칙
    - 특정 Branch로 Push하는 경우
    - 특정 Branch로 Pull Request하는 경우
    - 특정 시간대에 반복(Cron)
3. Jobs
    - Runner(서버)에서 실행되는 Steps의 조합
    - 여러 Job이 있는 경우 병렬로 실행하며, 순차적으로 실행할 수도 있음
        - 다른 Job에 의존 관계를 가질 수 있음(A Job Success 후 B Job 실행)
4. Steps
    - Step은 Job에서 실행되는 개별 작업
    - Action을 실행하거나 쉘 커맨드 실행
    - 하나의 Job에선 데이터를 공유할 수 있음
5. Actions
    - Workflow의 제일 작은 단위
    - Job을 생성하기 위해 여러 Step을 묶은 개념
    - 재사용이 가능한 Component
    - 개인적으로 Action을 만들 수도 있고, Marketplace의 Action을 사용할 수도 있음
6. Runner
    - Github Action도 일종의 서버에서 실행되는 개념
    - Workflow가 실행될 서버
    - Github-hosted Runner : Github Action의 서버를 사용하는 방법
        - 성능 : vCPU 2, Memory 7GB, Storage 14GB
    - Self-hosted Runner : 직접 서버를 호스팅해서 사용하는 방법

2.2 Github Action을 사용한 배포

Python application 찾고 Set up this workflow

YAML 파일 분석

- on : Event, 언제 Workflow가 실행될 것인가?
- jobs : jobs 정의, build는 job의 이름!
- runs-on : ubuntu 환경에서 실행
- uses : 사용할 Github Action
- name : Step의 이름
- uses 없는 경우 : run에 작성된 쉘 커맨드 실행

3. Mask Classification Streamlit 배포하기

3.1 Compute Engine에서 Streamlit 실행하기

SSH Key 생성

1. cd ~/.ssh/
2. ssh-keygen -t rsa -b 4096 -C “이메일”
3. passphrase는 그냥 엔터

SSH Key 복사

1. cat id_rsa.pub >> authorized_keys
    - 기본적으로 authorized_keys에 퍼블릭 키가 등록되면 외부에서 접근 가능
    - 단, GCP는 주기적으로 authorized_keys 파일을 삭제해서 외부에서 키파일 등록하는 과정이 필요
2. cat id_rsa.pub을 실행한 후, 복사

SSH Key 추가

1. GCP Console로 이동한 후, 메타데이터 클릭
2. SSH 키로 이동한 후, 수정 클릭
3. 복사한 퍼블릭키를 추가한 후 저장
    - 정상적으로 붙여넣기가 되었으면 왼쪽에 이름이 나타남. 이름이 나타나지 않으면 줄바꿈이나 띄어쓰기로 이슈가 없는지 확인

Github repo에 Key 저장

1. Product serving repo fork후 Settings - Secrtes - New repository secret 클릭
2. 인스턴스 외부 IP 복사
3. Github Secret에 HOST 추가, Value에 방금 복사한 인스턴스 IP 복붙
4. Secret에 USERNAME, Value 추가
5. Private Key 복사 후 Secret에 SSH KEY 추가, 방금 복사한 값 Value에 추가
    - Private Key는 길기 때문에 vi id_rsa으로 전체 복사
6. git config --global credential.helper store
    - Github Action에서 추가 인증없이 사용하도록 설정
7. Personal access tokens 클릭 - Generate new token 클릭

Streamlit 실행

- nohup streamlit run app.py --server.runOnSave true &
    - runOnSave : 파일이 변경될 경우 자동으로 다시 실행(Re Run)
    - nohup으로 Background 실행

3.2 Github Action을 사용한 배포 자동화

push 아래에 paths의 의미 : 이 경로에 있는 파일이 변경된 경우에 실행!

- Github Action Document : https://docs.github.com/en/actions/learn-github-actions/workflow-syntax-for-github-actions

Example: Ignoring paths

When all the path names match patterns in `paths-ignore`, the workflow will not run. GitHub evaluates patterns defined in `paths-ignore` against the path name. A workflow with the following path filter will only run on `push` events that include at least one file outside the `docs` directory at the root of the repository.

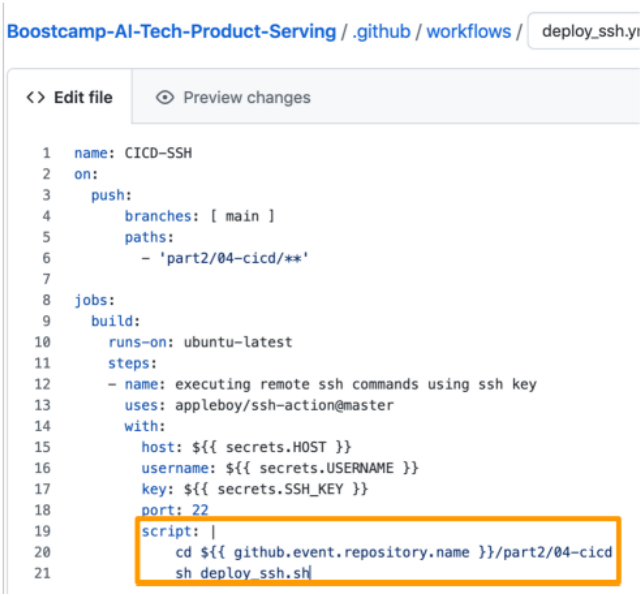
```
on:
  push:
    paths-ignore:
      - 'docs/**'
```

Example: Including paths

If at least one path matches a pattern in the `paths` filter, the workflow runs. To trigger a build anytime you push a JavaScript file, you can use a wildcard pattern.

```
on:
  push:
    paths:
      - '**.js'
```

deploy_ssh.sh 쉘 스크립트 실행



인스턴스를 중지했다가 다시 실행할 경우 IP가 변경됨 현재 IP는 임시로 할당했기 때문

- 고정 IP로 설정하면 인스턴스 중지 후 다시 실행해도 그대로 유지!

고정 IP 사용법

1. VPC 네트워크 - 외부 IP 주소로 이동
2. IP 우측의 예약 클릭
3. 고정된 외부 IP를 사용

다음에서 사용 중: 없음일 경우 고정 주소 해제(삭제) 실행!