머신러닝 파이프라인

리서치 코드 품질 관리 자동화

송호연

목차

리서치 코드 품질 관리 자동화

학습목표

리서치 코드 품질 관리 자동화

01. 리서치 코드 품질 문제에 대해 이해한다.

일반적으로 리서치 조직에서 생기는 코드 품질의 문제에 대해 이해한다.

02. 린트와 유닛테스트를 이해한다.

데이터 제품 패턴을 발견하기 위해 가져야할 접근법에 대해 공부한다.

03. 지속적 통합에 대해 이해한다.

코드 품질 관리를 자동화하는 기법인 지속적 통합을 공부한다.

리서치 코드 품질 문제

01

리서치 코드 품질 문제

복사 붙여넣기

Ctrl + C, Ctrl + V

*출처 : 출처 작성

리서치 코드 품질 문제

코드 품질 문제 유형

1. 리서치 코드는 각자의 개인 컴퓨터에 저장되어 있다.
2. 코드는 매번 복사 붙여넣기로 개발하여 코드 중복이 많다.
3. 연구 결과는 재연이 불가능하다.
4. 수 많은 코드 악취가 남아있다.

*출처 : 출처 작성

리서치 코드 품질 문제

깨진 유리창의 법칙

깨진 유리창의 법칙
Broken Windows Theory

**"만일 한 건물의 유리창이 깨어진 채로 방치되어 있다면,
곧 다른 유리창들도 깨어질 것"**

여러분이 품질이 낮은 코드를 쌓아올리기 시작하는 순간부터,
곧 다른 협업자들의 코드 품질도 떨어지기 시작한다는 것

*출처 : 출처 작성

리서치 코드 품질 문제

문제 #1 코드 중복

소프트웨어 취약점이 있는 코드가 복사될 때
개발자가 이러한 사본을 알지 못하는 경우 취약점은 계속해서 복사된 코드에 남아있게 된다.[4]

코드 중복을 사용하면 다음 이점들을 달성할 수 있다.

- 컴파일 시간의 단축
- 인지 부하의 감소
- 인적 오류의 감소
- 코드를 잊거나 간과하는 일의 감소

*출처 : 출처 작성

리서치 코드 품질 문제

코드 재사용성

재사용가능한 소프트웨어나 소프트웨어 지식은 재사용가능한 자산이다.
자산에는 설계, 요구명세, 검사, 아키텍처 등도 포함된다.

아마 가장 잘 알려져 있는 재사용 가능 자산은 코드이다. 코드 재사용은 어떤 시점에 쓰여진 프로그램의 일부 또는 전부를 이후의 다른 프로그램을 만들 때 사용하는 것이다. 코드의 재사용은 장황한 작업에 소비하는 시간과 에너지를 절약하는 전형적 기법이다.

라이브러리는 추상화가 좋은 예이다.

*출처 : 출처 작성

리서치 코드 품질 문제

문제 #2 너무 많은 전역 변수

전역 변수는 당장 쓰기에는 편할 수 있지만, 항상 사이드이펙트를 가져온다.

가능하면, 환경 값들은 환경 변수를 활용하고-
함수에 명시적으로 파라미터를 전달하고 받아오는 방식으로 고쳐야 한다.

*출처 : 출처 작성

리서치 코드 품질 문제

문제 #3 너무 긴 코드

너무 긴 코드는 디버깅하기 불편하고, 각 함수와 클래스의 구분이 명확하지 않게되는 경향이 있다.

각 python 파일은 500 라인 이하로 유지해보도록 하자.

*출처 : 출처 작성

리서치 코드 품질 문제

문제 #4 이상하게 꼬여있는 import

python의 relative import를 여기저기서 사용하게 되면,
서로 참조 관계가 얽혀서 나중에는 디버깅이 어려운 수준까지 가게 된다.

PYTHONPATH 환경변수를 활용하여 현재 시작 지점을 명확하게 하고,
absolute import를 사용하자.

*출처 : 출처 작성

리서치 코드 품질 문제

문제 #4 이상하게 꼬여있는 import

Imports are always put at the top of the file, just after any module comments and docstrings, and before module globals and constants.

Imports should be grouped in the following order:

1. Standard library imports.
2. Related third party imports.
3. Local application/library specific imports.

You should put a blank line between each group of imports.

*출처 : 출처 작성

리서치 코드 품질 문제

문제 #4 이상하게 꼬여있는 import

Absolute imports are recommended, as they are usually more readable and tend to be better behaved (or at least give better error messages) if the import system is incorrectly configured (such as when a directory inside a package ends up on sys.path):

```
import mypkg.sibling
from mypkg import sibling
from mypkg.sibling import example
```

*출처 : 출처 작성

리서치 코드 품질 문제

문제 #5 명확하지 않은 변수명

너무 축약어를 쓰다보면, 나밖에 못알아보는 코드가 된다.

혼자 개발할 때는 문제가 없지만, 팀웍을 위해서는 변수명을 이해하기 쉽게 만드는 게 중요하다.

*출처 : 출처 작성

린트와 유닛 테스트

02

린트와 유닛 테스트

린트

python black

*출처 : 출처 작성

린트와 유닛 테스트

Python의 인덴트

파이썬의 대표적인 특징이기도 한 인덴트는 공식 가이드인 PEP 8에 따라 공백 4칸을 원칙으로 한다.
구글의 파이썬 가이드라인 또한 공백 4칸 들여쓰기가 원칙이다.
물론 이 또한 파이썬답게 강제는 아니며 얼마든지 선택적으로 적용할 수 있다.

*colab에서 직접 실행해보세요.

린트와 유닛 테스트

Python의 인덴트

이외에도 PEP 8에는 다음과 같은 기준들이 포함되어 있다.

이와 같이 첫 번째 줄에 파라미터가 있다면,
파라미터가 시작되는 부분에 보기 좋게 맞춘다.

```
foo = long_function_name(var_one, var_two,
                         var_three, var_four)
```

*colab에서 직접 실행해보세요.

린트와 유닛 테스트

Python의 인덴트

이 코드에서처럼 첫 번째 줄에 파라미터가 없다면, 공백 4칸 인덴트를 한번 더 추가하여 다른 행과 구분되게 한다.

```
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)
```

*colab에서 직접 실행해보세요.

린트와 유닛 테스트

네이밍 컨벤션

파이썬의 변수명 네이밍 컨벤션은 자바와 달리

각 단어를 밑줄로 구분하여 표기하는 스네이크 케이스를 따른다.

함수명도 마찬가지다.

```
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)
```

파이썬 개발자들에게는 **파이썬 다운 방식(Pythonic Way)**에 굉장한 자부심이 있어서, 카멜 케이스뿐만 아니라 자바 스타일로 코딩하는 것을 지양한다.

(파이썬과 달리 자바에서는 단어 별로 대소문자를 구별하여 표기하는 카멜케이스를 따른다)

린트와 유닛 테스트

카멜 케이스, 스네이크 케이스

카멜 케이스는 **낙타(Camel)처럼 생긴 것**에 유래해 부른 이름이며 단어를 대소문자로 구분하여 섞어서 작명하는 방식이다.

카멜 케이스는 모든 단어의 첫 문자는 대문자로 시작하지만, 첫 단어의 첫 문자는 소문자로 시작한다.

스네이크 케이스는 **뱀(Snake)과 같은 모양**에서 유래했으며 각 단어를 언더스코어(_)로 구분한다.

```
# 카멜 케이스

camelCase: int = 1

# 스네이크 케이스

snake_case: int = 1
```

린트와 유닛 테스트

린트

flake8

*출처 : 출처 작성

린트와 유닛 테스트

Python 타입 체크

mypy

*출처 : 출처 작성

린트와 유닛 테스트

Python data type

프로그래밍에 있어서 자료형(data type)은 매우 중요하다.
변수들(variables)은 다른 자료형의 다른 데이터를 담을 수 있고, 다른 타입의 데이터들은 다른 일을 할 수 있다.

파이썬에서 지원하는 기본 자료형은 다음과 같다.

Text Type : str
Numeric Types : int, float, complex
Sequence Types : list, tuple, range
Mapping Type : dict
Set Types : set, frozenset
Boolean Type : bool
Binary Types : bytes, bytearray, memoryview

*출처 : w3schools

린트와 유닛 테스트

수치형

숫자(number) 형태로 이루어진 자료형
int(정수), float(실수), complex(복소수)가 있다.

```
# 정수
type(10)
```

```
int
```

```
# 실수
type(3.14)
```

```
float
```

```
# 복소수
type(1+2j)
```

```
complex
```

*출처 : 출처 작성

린트와 유닛 테스트

순서형

for문에서 사용할 수 있는 자료형
string(문자열), list(리스트), tuple(튜플)이 있다.

```
# 문자열
type('cat')
```

```
str
```

```
# 리스트
type(['러시안 블루', '샴', '먼치킨', '페르시안'])
```

```
list
```

```
# 튜플
type(('러시안 블루', '샴', '먼치킨', '페르시안'))
```

```
tuple
```

린트와 유닛 테스트

매핑형

key-value로 매핑(mapping)할 수 있는 자료형
dict(딕셔너리)가 있다.

```
dic = {'이름':'마스', '전화번호':'01012345678', '생일': '0101'}
type(dic)
```

```
dict
```

```
# value에 리스트를 넣을 수도 있다.

dic = {}
dic['cats'] = ['러시안 블루', '샴', '먼치킨', '페르시안']
dic['dogs'] = ['말티즈', '푸들', '슈나우저', '포메라니안']

dic
```

```
{'cats': ['러시안 블루', '샴', '먼치킨', '페르시안'],
 'dogs': ['말티즈', '푸들', '슈나우저', '포메라니안']}
```

린트와 유닛 테스트

불(bool)

True(참)/False(거짓) 로 표현하는 자료형

```
a = 1
b = 2

print(a<b)
```

```
True
```

```
print(b<a)
```

```
False
```

*출처 : 출처 작성

린트와 유닛 테스트

타입 힌트 (1)

파이썬은 대표적인 동적 타이핑 언어임에도, 타입을 지정할 수 있는 타입 힌트가 PEP 484 문서에 추가됐다.
다음과 같은 형태로 타입을 선언할 수 있다.

```
a: str = "1"
b: int = 1
```

*출처 : 출처 작성

린트와 유닛 테스트

타입 힌트 (2)

기존에 타입 힌트를 사용하지 않는 파이썬 함수는 다음과 같이 함수를 정의해 사용해왔다

```
def fn(a):
  …
```

위 함수의 경우엔 파라미터 a가 정수형인지 실수형인지,
함수의 리턴값이 무엇인지 명확하지가 않다.

그래서 함수의 가독성을 떨어뜨리며,
프로젝트의 규모가 커지게 되면, 버그 유발의 주범이 된다.

*출처 : 출처 작성

린트와 유닛 테스트

타입 힌트 (3)

그렇다면 다음 코드를 살펴보자.

```
def fn(a: int) -> bool:
  …
```

이처럼 타입 힌트를 사용하게 되면 fn() 함수의 파라미터 a가 정수형임을 분명하게 알 수 있으며
리턴 값으로 True 또는 False를 리턴할 것이라는 것도 확실히 알 수 있다.

*출처 : 출처 작성

린트와 유닛 테스트

타입 힌트 (4)

온라인 코딩 테스트 시에는 mypy를 사용하면 타입 힌트에 오류가 없는지 자동으로 확인할 수 있으므로 이를 통해 수정 후 코드를 제출할 수 있다. mypy는 다음과 같이 설치할 수 있다.

```
$ pip install mypy
```

타입 힌트가 잘못 지정된 코드는 다음과 같이 오류가 발생하므로 확인 후 직접 코드를 수정할 수 있다.

```
$ mypy solution.py
solution.py:9: error: Incompatible return value type (got "str", expected "int")
```

*출처 : 출처 작성

지속적 통합Continuous Integration

지속적 통합

지속적 통합

소프트웨어 공학에서, 지속적 통합(continuous integration, CI)은 지속적으로 퀄리티 컨트롤을 적용하는 프로세스를 실행하는 것이다.

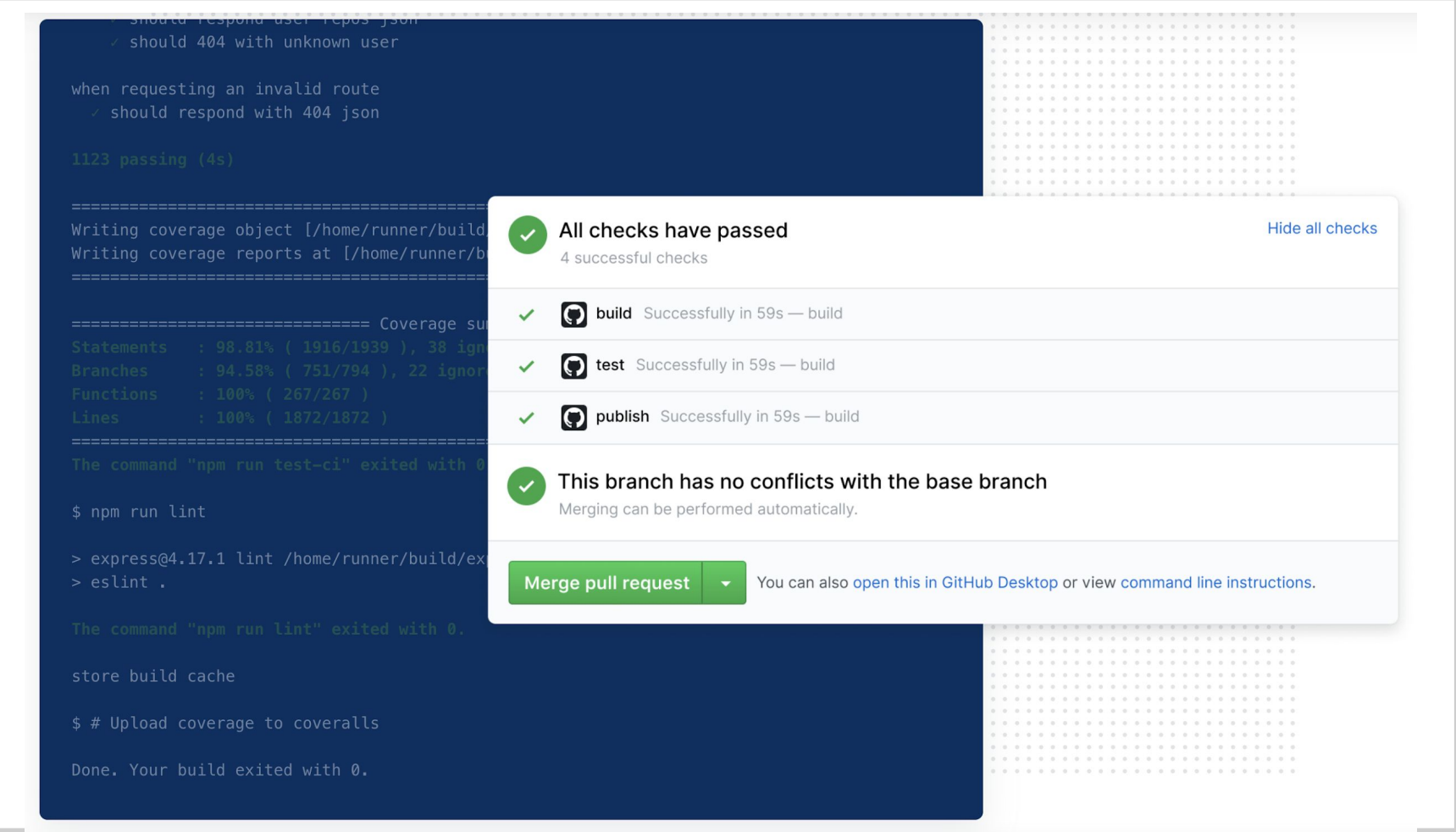
- 작은 단위의 작업, 빈번한 통합.

지속적인 통합은 모든 개발을 완료한 뒤에 퀄리티 컨트롤을 적용하는 고전적인 방법을 대체하는 방법으로서 소프트웨어의 질적 향상과 소프트웨어를 배포하는데 걸리는 시간을 줄이는데 초점이 맞추어져 있다.

*출처 : 출처 작성

지속적 통합

Github Actions



*출저 : 출저 작성

지속적 통합

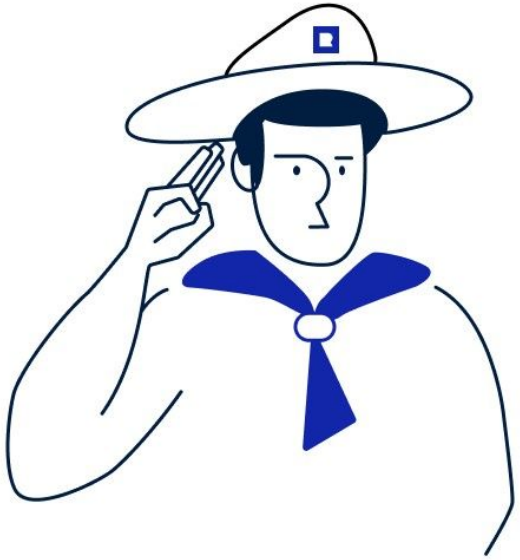
실습 시작

실습

*출처 : 출처 작성

지속적 통합

보이 스카우트 규칙



보이 스카우트 규칙
The boy scout rule

"떠날 때는 찾을 때보다 캠프장을 더욱 깨끗이 할 것"

여러분의 손을 거친 코드는 항상 원래 있었던 상태보다
조금 더 낫게 만들고 떠나라.
- 로버트 C. 마틴

*출처 : 출처 작성

짚어보기

리서치 코드 품질 관리

01. 리서치 코드 품질 문제에 대해 이해한다.

일반적으로 리서치 조직에서 생기는 코드 품질의 문제에 대해 이해한다.

02. 린트와 유닛테스트를 이해한다.

데이터 제품 패턴을 발견하기 위해 가져야할 접근법에 대해 공부한다.

03. 지속적 통합에 대해 이해한다.

코드 품질 관리를 자동화하는 기법인 지속적 통합을 공부한다.

머신러닝 파이프라인

리서치 코드 품질 관리

송호연

감사합니다.

THANKS FOR WATCHING