

데 이 터 사 이 언 스 없 이

추천시스템 만들기

디큐

dq.data.world@gmail.com

CONTENTS

01

AWS Personalize 소개

02

데이터 전처리

03

Personalize 트레이닝, 모델 준비하기

04

실제 서비스에 서빙하기 (feat. 가성비)

Q & A

C H A P
T E R



AWS PERSONALIZE 소개

AWS Personalize 란?

AWS 의 공식 문서에 따르면, Personalize 서비스는 "머신러닝 전문성" 없이 커스텀 개인화 추천 모델을 만들 수 있는 서비스입니다.

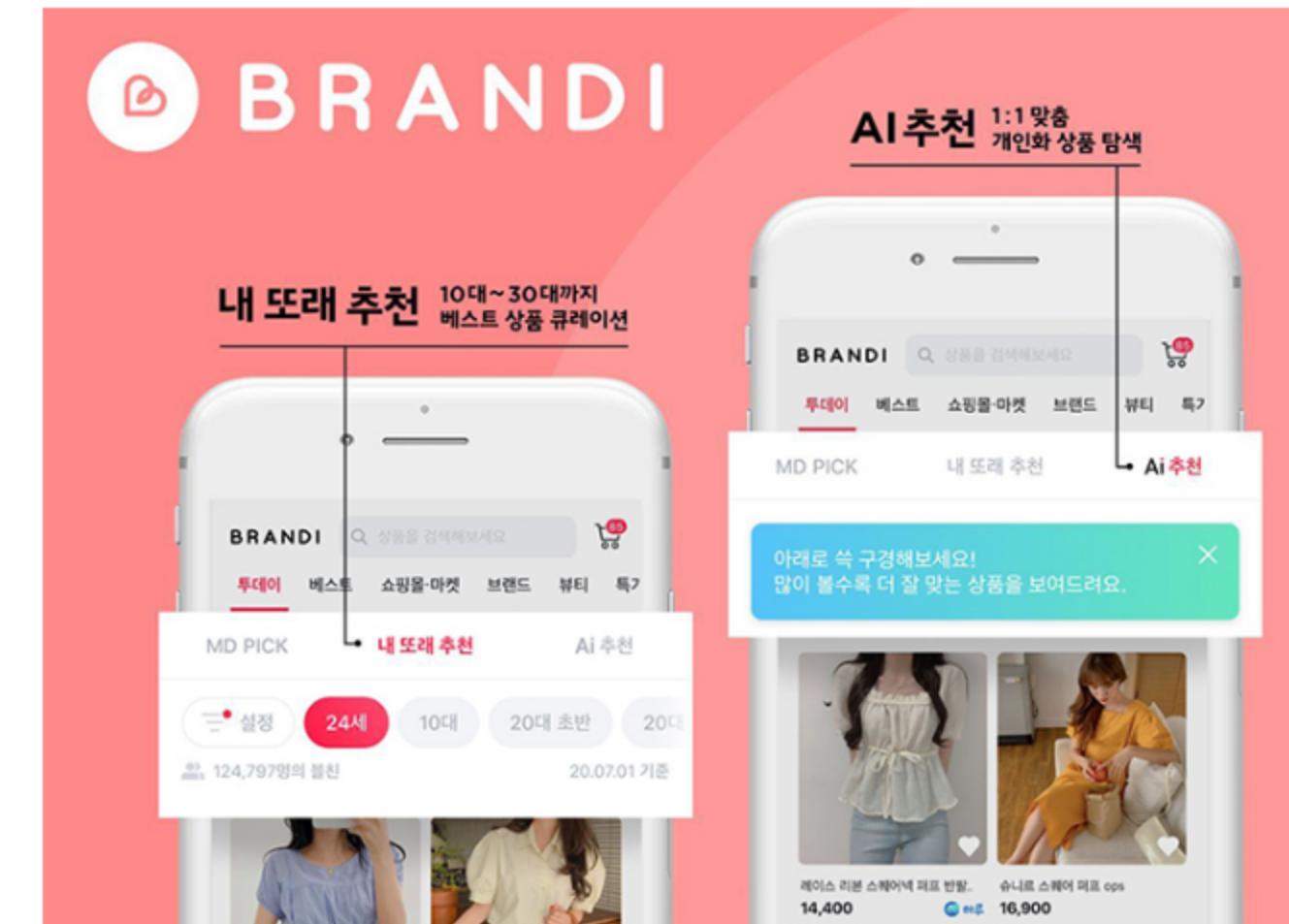
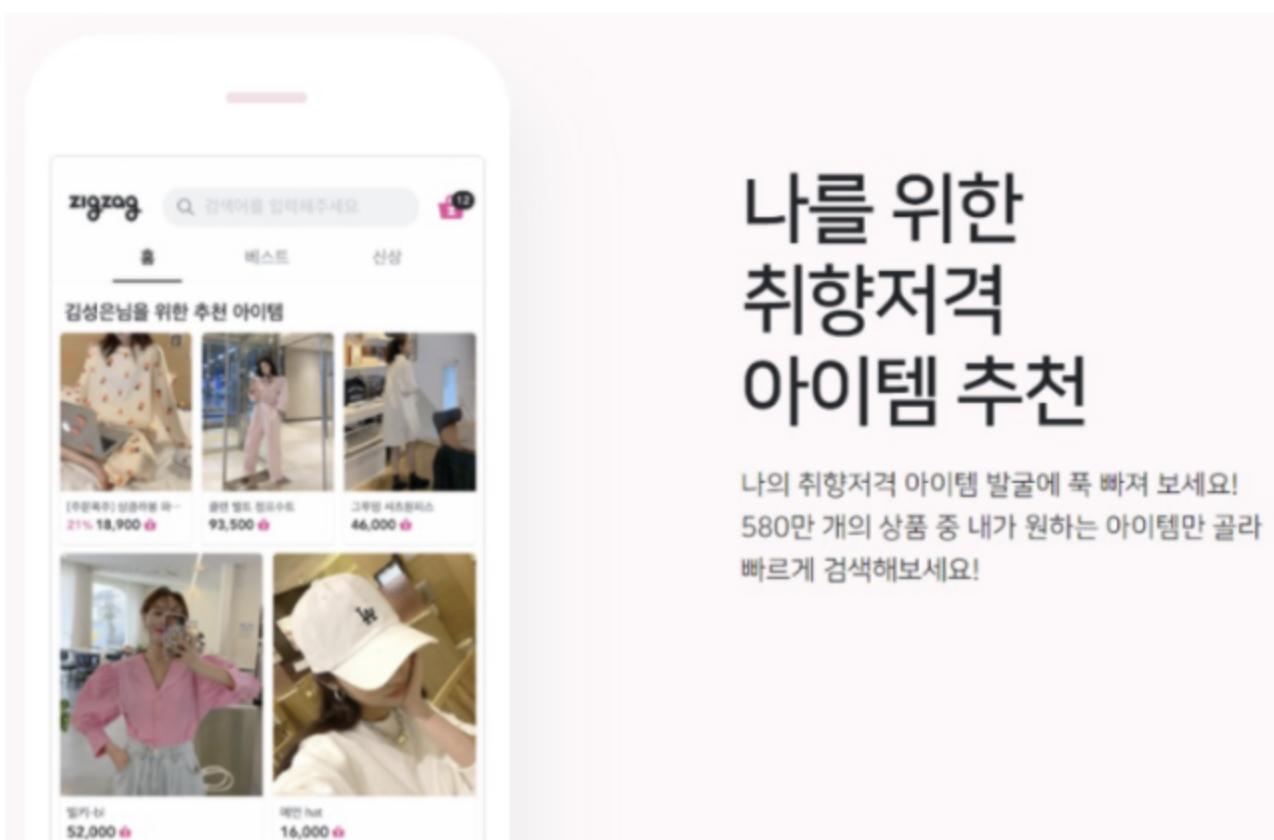
그 말은 데이터 과학자나, 데이터 팀을 꾸릴 여유가 없는 소규모 팀에서도 추천 시스템을 구축, 실험해볼 수 있다는 거죠.

추천시스템의 중요성은 시간이 갈수록 커지고 있고, 그에 따라 데이터 과학자, 데이터 팀의 시장 가치는 금값을 넘어 천상계로 가고 있죠. 그렇기 때문에 데이터 팀을 본격적으로 꾸리기 전에 '개인화 추천'이 얼마나 제품에 효과가 있을지 테스트를 해보는 과정이 필요합니다.

또한 소규모 스타트업의 경우에는 추천 그 자체가 제품의 메인 기능이 아닌 이상은 데이터 전문가를 채용하기에 부담되죠. 그런 경우에 적은 비용으로 추천시스템을 구축할 수 있다면 비즈니스에 도움이 될겁니다.

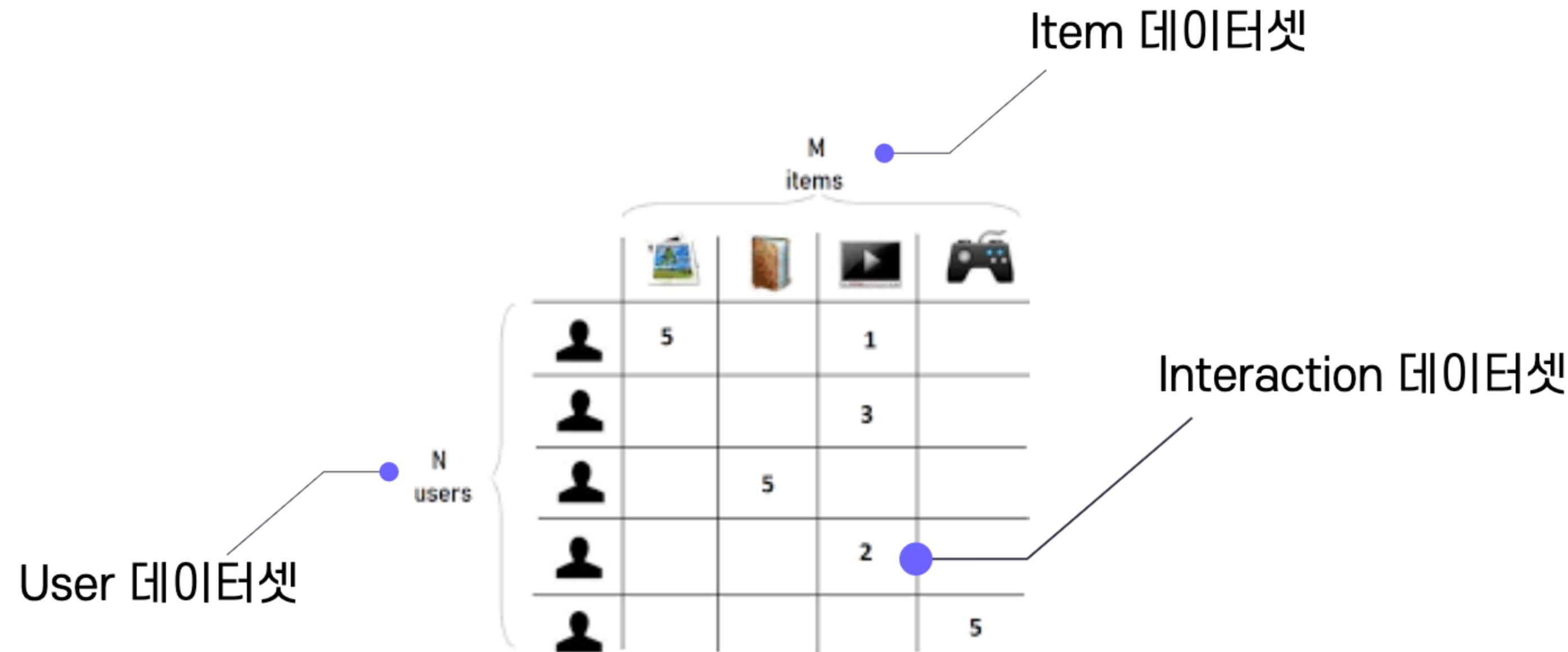
AWS Personalize 는 이에 대한 솔루션이 될 수 있습니다. 완벽하게 추천 팀을 꾸리는 것보다는 부족한 점이 많겠지만, PoC (Proof of Concept) 으로서 잘 작동하는 (그리고 1달만 실험해볼 수도 있는!) 추천시스템이니까요.

개인화 추천시스템 적용 사례



C H A P
T E
U2
데이터 전처리

데이터셋(Dataset)의 이해



Collaborative Filtering

이런 내용이 추천시스템에서는 자주 나오는데,
데이터 사이언스적인 관점에서 관심이 생기시는
분들은 찾아보시면 되고..

AWS Personalize 를 쓰는데에는 이런 지식이
필요가 없습니다.

	Book 1	Book 2	Book 3	Book 4	Book 5
User A	👍	👎	👍	👍	👍
User B		👍		👎	👎
User C	👍	👍	👎		
User D		👍	?		👎

Schema 정의

— user_schema.json

```
{  
    "type": "record",  
    "name": "Users",  
    "namespace": "your.namespace.schema",  
    "fields": [  
        {  
            "name": "user_id",  
            "type": "string"  
        },  
        {  
            "name": "os_device",  
            "type": "string"  
        },  
        {  
            "name": "age",  
            "type": "int"  
        }  
    "version": "1.0"  
}
```

ID값, 필수

Features, 원하는대로 추가 가능

Schema 정의

— item_schema.json

```
{  
    "type": "record",  
    "name": "Items",  
    "namespace": "your.namespace.schema",  
    "fields": [  
        {  
            "name": "item_id",  
            "type": "string"  
        },  
        {  
            "name": "category_id",  
            "type": "string"  
        },  
        {  
            "name": "price",  
            "type": "float"  
        }  
    ],  
    "version": "1.0"  
}
```

ID값, 필수

Features, 원하는대로 추가 가능

Schema 정의

— event_schema.json

```
{  
    "type": "record",  
    "name": "Interactions",  
    "namespace": "your.namespace.schema",  
    "fields": [  
        {  
            "name": "user_id",  
            "type": "string"  
        },  
        {  
            "name": "item_id",  
            "type": "string"  
        },  
        {  
            "name": "point",  
            "type": "float"  
        }  
    ]  
}
```

The code snippet shows a JSON schema for a record named 'Interactions'. It includes three fields: 'user_id' (string type), 'item_id' (string type), and 'point' (float type). Three annotations are present: a green bracket and pointer pointing to 'user_id' and 'item_id' with the text 'ID값, 필수 (user, item)'; a grey circle on the 'item_id' entry; and a green bracket and pointer pointing to 'point' with the text 'Features, 원하는대로 추가 가능'.

ID값, 필수 (user, item)

Features, 원하는대로 추가 가능

feature engineering (데이터 사이언스?)

	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none">• High training error• Training error close to test error• High bias	<ul style="list-style-type: none">• Training error slightly lower than test error	<ul style="list-style-type: none">• Very low training error• Training error much lower than test error• High variance
Regression illustration			
Classification illustration			
Deep learning illustration			
Possible remedies	<ul style="list-style-type: none">• Complexify model• Add more features• Train longer		<ul style="list-style-type: none">• Perform regularization• Get more data

모든 데이터를 다 넣는 것이
최고의 성능을 보장하지는 않습니다.

반복되는 실험을 통해
최적값을 찾아내는게 중요합니다.

데이터 전처리



```
def etl_user() -> None:
    df = pd.read_sql_table(대충 데이터베이스 연결하는 내용)

    df_selected = df.reset_index()[['user_id', 'age', 'os_device']]

    # 당연하게도, 이 부분에서 user_schema.json에 정의한 내용대로
    # dataframe의 column 이름, type을 맞춰줘야한다.

    # S3에 csv 형태로 저장한다.
    df.to_csv(USER_S3_PATH)

def etl_item():
    df = pd.read_sql_table(대충 데이터베이스 연결하는 내용)

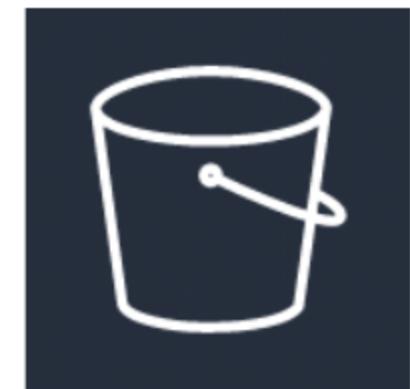
    # 마찬가지, 데이터 전처리 내용이 들어간다.

    df.to_csv(ITEM_S3_PATH)
    return item_df['item_id'].tolist()

def etl_event(common_list) -> None:
    df = pd.read_sql_table(대충 데이터베이스 연결하는 내용)

    # 마찬가지..

    pip_df.to_csv(EVENT_S3_PATH)
```



Amazon S3

AWS ARN ?

Review

Review the following role information. To edit the role, click an edit link, or click **Create Role** to finish.

Role Name	AccessToProdDynamoDB	Edit Role Name
Role ARN	arn:aws:iam::[REDACTED]:role/AccessToProdDynamoDB	
Trusted Entities	The account [REDACTED]	
Permissions	Amazon DynamoDB Full Access	Edit Permissions
Give this link to users who can switch roles in the console	https://signin.aws.amazon.com/switchrole? account=[REDACTED]&roleName=AccessToProdDynamoDB	Copy Link

Dataset Import Job (feat.AWS python SDK)

```
def create_dataset() -> None:
    personalize = boto3.client('personalize', region_name='your region')

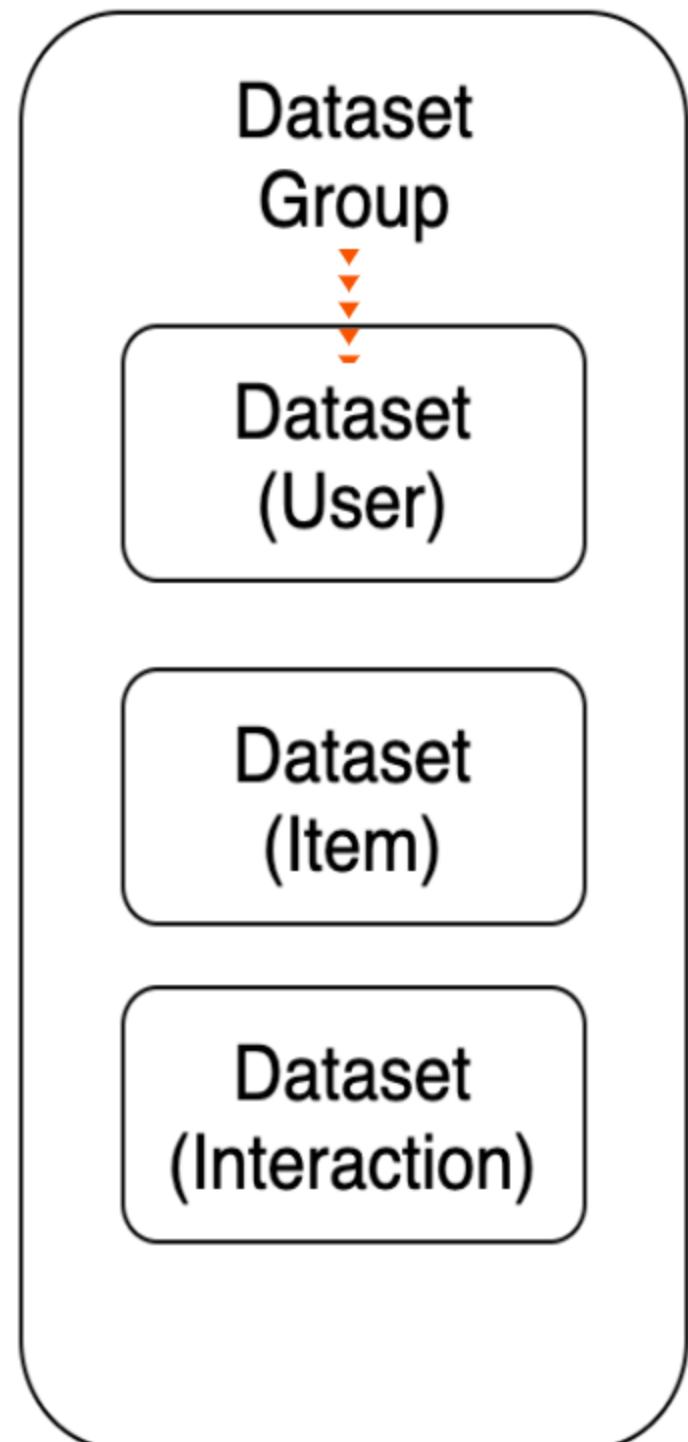
    # dataset group 의 이름이 중복되면 안되므로 적당히 time 함수를 써서 중복을 방지해준다.
    response = personalize.create_dataset_group(name=f'dsg-{int(time.time())}')
    dsg_arn = response['datasetGroupArn']

    description = personalize.describe_dataset_group(datasetGroupArn=dsg_arn)['datasetGroup']

    print('1. Name: ' + description['name'])
    print('1. ARN: ' + description['datasetGroupArn'])
    print('1. Status: ' + description['status'])

    # dataset 생성과 코드실행간의 타이밍 오류를 방지하기 위함.
    time.sleep(5)
```

Dataset Import Job (feat.AWS python SDK)



순서가 중요합니다.

Dataset Group 생성
(생성 완료까지 대기)

Dataset 생성
(생성완료까지 대기)

Dataset Import Job 생성
(생성완료까지 대기)

C H A P
T E
U 3

Personalize 트레이닝

모델 준비하기

AWS Personalize solution(모델) 만들기

```
solution_obj = personalize.create_solution(  
    name=f"solution-{int(time.time())}",  
    datasetGroupArn=DSG_ARN,  
    performAutoML=True,  
    performHPO=True  
)  
solution_arn = solution_obj['solutionArn']  
print("1. SOLUTION :: ", solution_arn)  
time.sleep(10)  
  
sv_obj = personalize.create_solution_version(  
    solutionArn=solution_arn,  
    trainingMode='FULL'  
)  
sv_arn = sv_obj['solutionVersionArn']  
print("2. Solution is Training :: ", sv_arn)
```

이 코드에서 수정할 수 있는 부분은 `performAutoML`, `performHPO` 인데, 개인적인 경험으로는 두 옵션 모두 `True`로 하는 것을 추천합니다.

AutoML: 최적의 모델 찾기

HPO: 하이퍼파라미터 최적화

두 옵션을 켜두었을 때, 결과적으로는 더 좋은 결과를 보여주었기 때문인데, 단점은 `training`에 드는 시간과 비용이 조금 더 증가할 수 있다.

AWS Personalize solution(모델) 만들기

```
solution_obj = personalize.create_solution(  
    name=f"solution-{int(time.time())}",  
    datasetGroupArn=DSG_ARN,  
    performAutoML=True,  
    performHPO=True  
)  
solution_arn = solution_obj['solutionArn']  
print("1. SOLUTION :: ", solution_arn)  
time.sleep(10)  
  
sv_obj = personalize.create_solution_version(  
    solutionArn=solution_arn,  
    trainingMode='FULL'  
)  
sv_arn = sv_obj['solutionVersionArn']  
print("2. Solution is Training :: ", sv_arn)
```

solution 생성 옵션

(<https://docs.aws.amazon.com/personalize/latest/dg/working-with-predefined-recipes.html>)

목적에따라

User_personalize
Personalized_ranking

Related_items
로 크게 나뉜다.

CHAPTER

04

실제 서비스에 서빙하기 (feat. 가성비)

Batch VS Campaign

	Batch	Campaign
추천 주기	1일	실시간
구현 난이도	★★★	★★★★★
가격	\$	\$\$\$\$\$

Solution & Filter 구현

```
def batch_recommendation() -> str:
    DSG_ARN = personalize.list_dataset_groups()['datasetGroups'][0]['datasetGroupArn']

    SOLUTION_ARN = personalize.list_solutions(datasetGroupArn=DSG_ARN)['solutions'][0]['sol
utionArn']
    SV_ARN = personalize.list_solution_versions(
        solutionArn=SOLUTION_ARN
    )['solutionVersions'][0]['solutionVersionArn']
    filter_list = personalize.list_filters(
        datasetGroupArn=DSG_ARN
    )['Filters']
    if len(filter_list) == 0:
        filter_obj = personalize.create_filter(
            name=f"filter-{int(time.time())}",
            datasetGroupArn=DSG_ARN,
            filterExpression=f'EXCLUDE ItemID WHERE 조건1 AND 조건 2'
        )
        filter_arn = filter_obj['filterArn']
    else:
        filter_arn = filter_list[0]['filterArn']

    print("1. FILTER :: ", filter_arn)
```

Solution과 결합되어 만들어지는 형태
Filter를 만들때 **filter Expression**이라는
것을 이용 (SQL과 비슷한 문법)

Solution Version 구현

```
# filter 와 solution version 가 생성될 때까지 대기하는 while loop.
FILTER_STATUS = ''
SV_STATUS = ''
while SV_STATUS != 'ACTIVE' or FILTER_STATUS != 'ACTIVE':
    print("waiting SV.... ", SV_STATUS, "\nfilter.... ::", FILTER_STATUS)
    time.sleep(30)
    SV_STATUS = personalize.describe_solution_version(
        solutionVersionArn=SV_ARN
    )['solutionVersion']['status']
    FILTER_STATUS = personalize.describe_filter(
        filterArn=filter_arn
    )['filter']['status']

response = personalize.create_batch_inference_job(
    solutionVersionArn=SV_ARN,
    jobName=f"recommendation-batch-{int(time.time())}",
    roleArn='your role arn',
    jobInput=
    {"s3DataSource": {"path": "s3://path/to/batch_input.json"}},
    jobOutput=
    {"s3DataDestination": {"path": "s3://path/to/batch_result/"}},
    numResults=500, #(maximum is 500)
    filterArn=filter_arn
)
BATCH_ARN = response['batchInferenceJobArn']
print("Batch job created ", BATCH_ARN)
return BATCH_ARN
```

**Solution 과 Solution Version
다른 개념이다**

**사실상 추천 결과는 여기서 이미 얻어짐
(S3에 아웃풋을 내놓는 코드)**

서비스 DB에 ETL 서빙



Amazon S3



```
# DB에 추천 결과를 ETL
def etl_recommendation(batch_arn: str) -> None:
    BATCH_STATUS = ''
    while BATCH_STATUS != 'ACTIVE':
        print('WAITING BATCH TO DONE... ::', BATCH_STATUS)
        time.sleep(10)
        BATCH_STATUS = personalize.describe_batch_inference_job(
            batchInferenceJobArn=batch_arn
        )['batchInferenceJob']['status']

    s3 = boto3.resource('s3')
    output = s3.Object('bucket name', 'path/to/batch_result/batch_input.json.out')
    tmp = output.get()['Body'].read().decode('utf-8').split('\n')
    result = pd.DataFrame(tmp)[:-1]

    # 제품에서 필요한 형태로 재가공해서 데이터베이스에 결과를 serving 할 수 있다.

    result.to_sql('result', 데이터베이스 내용)
```



Clean up (과금 폭탄을 피하자)

```
# AWS 위에 Personalize 관련 인스턴스를 모두 지워준다.
def clean_up() -> None:
    DSG_ARN = personalize.list_dataset_groups()['datasetGroups'][0]['datasetGroupArn']

    FILTER_LIST = personalize.list_filters(
        datasetGroupArn=DSG_ARN
    )['Filters']
    SOLUTION_LIST = personalize.list_solutions(
        datasetGroupArn=DSG_ARN
    )['solutions']
    DS_LIST = personalize.list_datasets(
        datasetGroupArn=DSG_ARN
    )['datasets']

    for idx, filter in enumerate(FILTER_LIST):
        response = personalize.delete_filter(
            filterArn=filter['filterArn']
        )
        print("number ", idx, " FILTER CLEAN UP")

    for idx, solution in enumerate(SOLUTION_LIST):
        response = personalize.delete_solution(
            solutionArn=solution['solutionArn']
        )
        print("number ", idx, " SOLUTION CLEAN UP")

    FILTER_LENGTH = 1
    while FILTER_LENGTH > 0:
        print("WAIT FILTER TO DELETE... :: ", FILTER_LENGTH)
        time.sleep(10)
        FILTER_LENGTH = len(personalize.list_filters(
            datasetGroupArn=DSG_ARN
        )['Filters'])
```

순서가 중요

Filter 삭제

solution 삭제

dataset 삭제

dataset group 삭제

코드와 설명

<https://dq-dreamsearch.com/121>

Q & A



궁금한 점을 물어보세요
Personalize 관련이 아닌
추천시스템 전반에 대해서도 환영합니다.

THANK
Y O U E V E R Y O N E

S E E Y O U A G A I N