Creating and Evaluating Solutions

In this notebook, you will train several models using Amazon Personalize, specifically:

1. User Personalization - what items are most relevant to a specific user.
2. Similar Items - given an item, what items are similar to it.
3. Personalized Ranking - given a user and a collection of items, in what order are they most releveant.

Outline

Introduction

To recap, for the most part, the algorithms in Amazon Personalize (called recipes) look to solve different tasks, explained here:

1. **User Personalization** - New release that supports ALL HRNN workflows / user personalization needs, it will be what we use here.
2. **HRNN & HRNN-Metadata** - Recommends items based on previous user interactions with items.
3. **HRNN-Coldstart** - Recommends new items for which interaction data is not yet available.
4. **Personalized-Ranking** - Takes a collection of items and then orders them in probable order of interest using an HRNN-like approach.
5. **SIMS (Similar Items)** - Given one item, recommends other items also interacted with by users.
6. **Popularity-Count** - Recommends the most popular items, if HRNN or HRNN-Metadata do not have an answer - this is returned by default.

No matter the use case, the algorithms all share a base of learning on user-item-interaction data which is defined by 3 core attributes:

1. **UserID** - The user who interacted
2. **ItemID** - The item the user interacted with
3. **Timestamp** - The time at which the interaction occurred

We also support event types and event values defined by:

1. **Event Type** - Categorical label of an event (browse, purchased, rated, etc).

2. **Event Value** - A value corresponding to the event type that occurred. Generally speaking, we look for normalized values between 0 and 1 over the event types. For example, if there are three phases to complete a transaction (clicked, added-to-cart, and purchased), then there would be an event_value for each phase as 0.33, 0.66, and 1.0 respectfully.

The event type and event value fields are additional data which can be used to filter the data sent for training the personalization model. In this particular exercise we will not have an event type or event value.

To run this notebook, you need to have run the previous notebooks, `01_Data_Layer.ipynb` , where you created a dataset and imported interaction and item metadata data into Amazon Personalize. At the end of that notebook, you saved some of the variable values, which you now need to load into this

In [2]:

```
%store -r
```

In [3]:

```
import time
from time import sleep
import json

import boto3
```

In [4]:

```
# Configure the SDK to Personalize:
personalize = boto3.client('personalize')
personalize_runtime = boto3.client('personalize-runtime')
```

Create Solutions

With our three datasets imported into our dataset group, we can now turn to training models. As a reminder, we will be training three models in this workshop to support four different personalization use-cases wth three different models. One model will be used to make related product recommendations on the product detail view/page, another model will be used to make personalized product recommendations to users on the homepage, and the last model will be used to rerank product lists on the category and featured products page. This last model will be repurposed to offer discounts to users. In Amazon Personalize, training a model involves creating a Solution and Solution Version. So when we are finished we will have three solutions and a solution version for each solution.

When creating a solution, you provide your dataset group and the recipe for training. Let's declare the recipes that we will need for our solutions.

List Recipes

First, let's list all available recipes.

In [6]:

```
list_recipes_response = personalize.list_recipes()
list_recipes_response
```

Out[6]:


```
{'recipes': [{'name': 'aws-hrnn',
   'recipeArn': 'arn:aws:personalize:::recipe/aws-hrnn',
   'status': 'ACTIVE',
   'creationDateTime': datetime.datetime(2019, 6, 10, 0, 0, tzinfo=tzlocal()),
   'lastUpdatedDateTime': datetime.datetime(2021, 2, 6, 19, 6, 40, 447000, tzinfo=tzlocal())},
  {'name': 'aws-hrnn-coldstart',
   'recipeArn': 'arn:aws:personalize:::recipe/aws-hrnn-coldstart',
   'status': 'ACTIVE',
   'creationDateTime': datetime.datetime(2019, 6, 10, 0, 0, tzinfo=tzlocal()),
   'lastUpdatedDateTime': datetime.datetime(2021, 2, 6, 19, 6, 40, 447000, tzinfo=tzlocal())},
  {'name': 'aws-hrnn-metadata',
   'recipeArn': 'arn:aws:personalize:::recipe/aws-hrnn-metadata',
   'status': 'ACTIVE',
   'creationDateTime': datetime.datetime(2019, 6, 10, 0, 0, tzinfo=tzlocal()),
   'lastUpdatedDateTime': datetime.datetime(2021, 2, 6, 19, 6, 40, 447000, tzinfo=tzlocal())},
  {'name': 'aws-personalized-ranking',
   'recipeArn': 'arn:aws:personalize:::recipe/aws-personalized-ranking',
   'status': 'ACTIVE',
   'creationDateTime': datetime.datetime(2019, 6, 10, 0, 0, tzinfo=tzlocal()),
   'lastUpdatedDateTime': datetime.datetime(2021, 2, 6, 19, 6, 40, 447000, tzinfo=tzlocal())},
  {'name': 'aws-popularity-count',
   'recipeArn': 'arn:aws:personalize:::recipe/aws-popularity-count',
   'status': 'ACTIVE',
   'creationDateTime': datetime.datetime(2019, 6, 10, 0, 0, tzinfo=tzlocal()),
   'lastUpdatedDateTime': datetime.datetime(2021, 2, 6, 19, 6, 40, 447000, tzinfo=tzlocal())},
  {'name': 'aws-sims',
   'recipeArn': 'arn:aws:personalize:::recipe/aws-sims',
   'status': 'ACTIVE',
   'creationDateTime': datetime.datetime(2019, 6, 10, 0, 0, tzinfo=tzlocal()),
   'lastUpdatedDateTime': datetime.datetime(2021, 2, 6, 19, 6, 40, 447000, tzinfo=tzlocal())},
  {'name': 'aws-user-personalization',
   'recipeArn': 'arn:aws:personalize:::recipe/aws-user-personalization',
   'status': 'ACTIVE',
   'creationDateTime': datetime.datetime(2019, 6, 10, 0, 0, tzinfo=tzlocal()),
   'lastUpdatedDateTime': datetime.datetime(2021, 2, 6, 19, 6, 40, 447000, tzinfo=tzlocal())}],
 'ResponseMetadata': {'RequestId': '0cc205b9-4c2b-4dbb-804e-b83248cfc452',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'content-type': 'application/x-amz-json-1.1',
   'date': 'Sun, 18 Apr 2021 00:09:56 GMT',
   'x-amzn-requestid': '0cc205b9-4c2b-4dbb-804e-b83248cfc452',
   'content-length': '1259',
```

```
  'connection': 'keep-alive'},
 'RetryAttempts': 0}}
```

As you can see above, there are several recipes to choose from. Let's declare the recipes for each Solution.

Declare Personalize Recipe for Related Products

On the product detail page we want to display related products so we'll create a campaign using the SIMS (https://docs.aws.amazon.com/personalize/latest/dg/native-recipe-sims.html) recipe.

> The Item-to-item similarities (SIMS) recipe is based on the concept of collaborative filtering. A SIMS model leverages user-item interaction data to recommend items similar to a given item. In the absence of sufficient user behavior data for an item, this recipe recommends popular items.

In [7]:

```
related_recipe_arn = "arn:aws:personalize:::recipe/aws-sims"
```

Declare Personalize Recipe for Product Recommendations

Since we are providing metadata for users and items, we will be using the User-Personalization (https://docs.aws.amazon.com/personalize/latest/dg/native-recipe-new-item-USER_PERSONALIZATION.html) recipe for our product recommendations solution.

> The User-Personalization (aws-user-personalization) recipe is optimized for all personalized recommendation scenarios. It predicts the items that a user will interact with based on Interactions, Items, and Users datasets. When recommending items, it uses automatic item exploration.

In [8]:

```
recommend_recipe_arn = "arn:aws:personalize:::recipe/aws-user-personalization"
```

Declare Personalize Recipe for Personalized Ranking

In use-cases where we have a curated list of products, we can use the Personalized-Ranking (https://docs.aws.amazon.com/personalize/latest/dg/native-recipe-search.html) recipe to reorder the products for the current user.

> The Personalized-Ranking recipe generates personalized rankings. A personalized ranking is a list of recommended items that are re-ranked for a specific user.

In [9]:

```
ranking_recipe_arn = "arn:aws:personalize:::recipe/aws-personalized-ranking"
```

Create Solutions and Solution Versions

With our recipes defined, we can now create our solutions and solution versions.

SIMS

SIMS is one of the oldest algorithms used within Amazon for recommendation systems. A core use case for it is when you have one item and you want to recommend items that have been interacted with in similar ways over your entire user base. This means the result is not personalized per user. Sometimes this leads to recommending mostly popular items, so there is a hyperparameter that can be tweaked which will reduce the popular items in your results.

For our use case, let's assume we pick a particular item. We can then use SIMS to recommend other items based on the interaction behavior of the entire user base. The results are not personalized per user, but instead, differ depending on the item we chose as our input.

Just like last time, we start by selecting the recipe.

Create Related Products Solution

In [10]:

```
create_solution_response = personalize.create_solution(
    name = "amazon-personalize-immersionday-related-products",
    datasetGroupArn = dataset_group_arn,
    recipeArn = related_recipe_arn
)

related_solution_arn = create_solution_response['solutionArn']
print(json.dumps(create_solution_response, indent=2))
```


```
{
  "solutionArn": "arn:aws:personalize:us-east-1:968824662561:solution/amazon-personalize-immersionday-related-
products",
  "ResponseMetadata": {
    "RequestId": "34da59b1-d491-417e-b827-7be94bb45de3",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "content-type": "application/x-amz-json-1.1",
      "date": "Sun, 18 Apr 2021 00:10:05 GMT",
      "x-amzn-requestid": "34da59b1-d491-417e-b827-7be94bb45de3",
      "content-length": "118",
      "connection": "keep-alive"
    },
    "RetryAttempts": 0
  }
}
```


Create Related Products Solution Version

In [11]:

```
create_solution_version_response = personalize.create_solution_version(
    solutionArn = related_solution_arn
)

related_solution_version_arn = create_solution_version_response['solutionVersionArn']
print(json.dumps(create_solution_version_response, indent=2))
```


```
{
  "solutionVersionArn": "arn:aws:personalize:us-east-1:968824662561:solution/amazon-personalize-immersionday-r
elated-products/0f3b3c74",
  "ResponseMetadata": {
    "RequestId": "dbeb7a92-fb60-452f-9d2a-75e2685e9e2f",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "content-type": "application/x-amz-json-1.1",
      "date": "Sun, 18 Apr 2021 00:10:07 GMT",
      "x-amzn-requestid": "dbeb7a92-fb60-452f-9d2a-75e2685e9e2f",
      "content-length": "134",
      "connection": "keep-alive"
    },
    "RetryAttempts": 0
  }
}
```


User Personalization

The User-Personalization (aws-user-personalization) recipe is optimized for all USER_PERSONALIZATION recommendation scenarios. When recommending items, it uses automatic item exploration.

With automatic exploration, Amazon Personalize automatically tests different item recommendations, learns from how users interact with these recommended items, and boosts recommendations for items that drive better engagement and conversion. This improves item discovery and engagement when you have a fast-changing catalog, or when new items, such as news articles or promotions, are more relevant to users when fresh.

You can balance how much to explore (where items with less interactions data or relevance are recommended more frequently) against how much to exploit (where recommendations are based on what we know or relevance). Amazon Personalize automatically adjusts future recommendations based on implicit user feedback.

First, select the recipe by finding the ARN in the list of recipes above.

In [12]:

```
create_solution_response = personalize.create_solution(
    name = "amazon-personalize-immersionday-product-personalization",
    datasetGroupArn = dataset_group_arn,
    recipeArn = recommend_recipe_arn
)

recommend_solution_arn = create_solution_response['solutionArn']
print(json.dumps(create_solution_response, indent=2))
```


```
{
  "solutionArn": "arn:aws:personalize:us-east-1:968824662561:solution/amazon-personalize-immersionday-product-
personalization",
  "ResponseMetadata": {
    "RequestId": "09b0f72f-b32f-4003-8fd3-24940d377ab1",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "content-type": "application/x-amz-json-1.1",
      "date": "Sun, 18 Apr 2021 00:10:09 GMT",
      "x-amzn-requestid": "09b0f72f-b32f-4003-8fd3-24940d377ab1",
      "content-length": "125",
      "connection": "keep-alive"
    },
    "RetryAttempts": 0
  }
}
```

In [13]:

```
create_solution_version_response = personalize.create_solution_version(
    solutionArn = recommend_solution_arn
)

recommend_solution_version_arn = create_solution_version_response['solutionVersionArn']
print(json.dumps(create_solution_version_response, indent=2))
```


```
{
  "solutionVersionArn": "arn:aws:personalize:us-east-1:968824662561:solution/amazon-personalize-immersionday-p
roduct-personalization/dc9e29c9",
  "ResponseMetadata": {
    "RequestId": "38a5e351-11a6-4370-86e6-42b5eb9fee8f",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "content-type": "application/x-amz-json-1.1",
      "date": "Sun, 18 Apr 2021 00:10:10 GMT",
      "x-amzn-requestid": "38a5e351-11a6-4370-86e6-42b5eb9fee8f",
      "content-length": "141",
      "connection": "keep-alive"
    },
    "RetryAttempts": 0
  }
}
```


Personalized Ranking

Personalized Ranking is an interesting application of HRNN. Instead of just recommending what is most probable for the user in question, this algorithm takes in a user and a list of items as well. The items are then rendered back in the order of most probable relevance for the user. The use case here is for filtering on unique categories that you do not have item metadata to create a filter, or when you have a broad collection that you would like better ordered for a particular user.

For our use case, we could imagine that an ecommerce website may want to create a shelf of seasonal items, or items by a specific designer. We most likely have these lists based title metadata we have. We would use personalized ranking to re-order the list of items for each user, based on their previous tagging history.

Just like last time, we start by selecting the recipe.

In [14]:

```
create_solution_response = personalize.create_solution(
    name = "amazon-personalize-immersionday-personalized-ranking",
    datasetGroupArn = dataset_group_arn,
    recipeArn = ranking_recipe_arn
)

ranking_solution_arn = create_solution_response['solutionArn']
print(json.dumps(create_solution_response, indent=2))
```


```
{
  "solutionArn": "arn:aws:personalize:us-east-1:968824662561:solution/amazon-personalize-immersionday-personal
ized-ranking",
  "ResponseMetadata": {
    "RequestId": "c3be50b5-bb0f-4105-b107-613e57619770",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "content-type": "application/x-amz-json-1.1",
      "date": "Sun, 18 Apr 2021 00:10:14 GMT",
      "x-amzn-requestid": "c3be50b5-bb0f-4105-b107-613e57619770",
      "content-length": "122",
      "connection": "keep-alive"
    },
    "RetryAttempts": 0
  }
}
```


Create Personalized Ranking Solution Version

In [15]:

```
create_solution_version_response = personalize.create_solution_version(
    solutionArn = ranking_solution_arn
)

ranking_solution_version_arn = create_solution_version_response['solutionVersionArn']
print(json.dumps(create_solution_version_response, indent=2))
```


```
{
  "solutionVersionArn": "arn:aws:personalize:us-east-1:968824662561:solution/amazon-personalize-immersionday-p
ersonalized-ranking/7db3eae3",
  "ResponseMetadata": {
    "RequestId": "6dc243ab-f38c-4248-9893-071895820a20",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "content-type": "application/x-amz-json-1.1",
      "date": "Sun, 18 Apr 2021 00:10:15 GMT",
      "x-amzn-requestid": "6dc243ab-f38c-4248-9893-071895820a20",
      "content-length": "138",
      "connection": "keep-alive"
    },
    "RetryAttempts": 0
  }
}
```


Wait for Solution Versions to Complete

It can take 40-60 minutes for all solution versions to be created. During this process a model is being trained and tested with the data contained within your datasets. The duration of training jobs can increase based on the size of the dataset, training parameters and a selected recipe. We submitted requests for all three solutions and versions at once so they are trained in parallel and then below we will wait for all three to finish.

While you are waiting for this process to complete you can learn more about solutions in [the documentation (https://docs.aws.amazon.com/personalize/latest/dg/training-deploying-solutions.html)](https://docs.aws.amazon.com/personalize/latest/dg/training-deploying-solutions.html)

Wait for Related Products Solution Version to Have ACTIVE Status

In [16]:

```
%%time

soln_ver_arns = [ related_solution_version_arn, recommend_solution_version_arn, ranking_solution_version_arn ]

max_time = time.time() + 3*60*60 # 3 hours
while time.time() < max_time:
    for soln_ver_arn in reversed(soln_ver_arns):
        soln_ver_response = personalize.describe_solution_version(
            solutionVersionArn = soln_ver_arn
        )
        status = soln_ver_response["solutionVersion"]["status"]

        if status == "ACTIVE":
            print(f'Solution version {soln_ver_arn} successfully completed')
            soln_ver_arns.remove(soln_ver_arn)
        elif status == "CREATE FAILED":
            print(f'Solution version {soln_ver_arn} failed')
            if soln_ver_response.get('failureReason'):
                print('   Reason: ' + soln_ver_response['failureReason'])
            soln_ver_arns.remove(soln_ver_arn)

    if len(soln_ver_arns) > 0:
        print('At least one solution version is still in progress')
        time.sleep(60)
    else:
        print("All solution versions have completed")
        break
```

At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress

```
At least one solution version is still in progress
Solution version arn:aws:personalize:us-east-1:968824662561:solution/amazon-personalize-immersionday-related-p
roducts/0f3b3c74 successfully completed
At least one solution version is still in progress
At least one solution version is still in progress
At least one solution version is still in progress
Solution version arn:aws:personalize:us-east-1:968824662561:solution/amazon-personalize-immersionday-personali
zed-ranking/7db3eae3 successfully completed
At least one solution version is still in progress
At least one solution version is still in progress
Solution version arn:aws:personalize:us-east-1:968824662561:solution/amazon-personalize-immersionday-product-p
ersonalization/dc9e29c9 successfully completed
All solution versions have completed
CPU times: user 625 ms, sys: 0 ns, total: 625 ms
Wall time: 49min 5s
```

View solution creation status in the console

You can view the status updates in the console:

- In another browser tab you should already have the AWS Console up from opening this notebook instance.
- Switch to that tab and search at the top for the service `Personalize`, then go to that service page.
- Click `View dataset groups`.
- Click the name of your dataset group, most likely something with POC in the name.
- Click `Solutions and recipes`.
- You will now see a list of all of the solutions you created above, including a column with the status of the solution versions. Once it is `Active`, your solution is ready to be reviewed. It is also capable of being deployed.

Hyperparameter tuning

Personalize offers the option of running hyperparameter tuning when creating a solution. Because of the additional computation required to perform hyperparameter tuning, this feature is turned off by default. Therefore, the solutions we created above, will simply use the default values of the hyperparameters for each recipe. For more information about hyperparameter tuning, see the documentation (https://docs.aws.amazon.com/personalize/latest/dg/customizing-solution-config-hpo.html).

If you have settled on the correct recipe to use, and are ready to run hyperparameter tuning, the following code shows how you would do so, using SIMS as an example.

```
sims_create_solution_response = personalize.create_solution(
    name = "personalize-poc-sims-hpo",
    datasetGroupArn = dataset_group_arn,
    recipeArn = SIMS_recipe_arn,
    performHPO=True
)

sims_solution_arn = sims_create_solution_response['solutionArn']
print(json.dumps(sims_create_solution_response, indent=2))
```

If you already know the values you want to use for a specific hyperparameter, you can also set this value when you create the solution. The code below shows how you could set the value for the `popularity_discount_factor` for the SIMS recipe.

```
sims_create_solution_response = personalize.create_solution(
    name = "personalize-poc-sims-set-hp",
    datasetGroupArn = dataset_group_arn,
    recipeArn = SIMS_recipe_arn,
    solutionConfig = {
        'algorithmHyperParameters': {
            'popularity_discount_factor': '0.7'
        }
    }
)

sims_solution_arn = sims_create_solution_response['solutionArn']
print(json.dumps(sims_create_solution_response, indent=2))
```

Evaluate solution versions

Back to top

It should not take more than an hour to train all the solutions from this notebook. While training is in progress, we recommend taking the time to read up on the various algorithms (recipes) and their behavior in detail. This is also a good time to consider alternatives to how the data was fed into the system and what kind of results you expect to see.

When the solutions finish creating, the next step is to obtain the evaluation metrics. Personalize calculates these metrics based on a subset of the training data. The image below illustrates how Personalize splits the data. Given 10 users, with 10 interactions each (a circle represents an interaction), the interactions are ordered from oldest to newest based on the timestamp. Personalize uses all of the interaction data from 90% of the users (blue circles) to train the solution version, and the remaining 10% for evaluation. For each of the users in the remaining 10%, 90% of their interaction data (green circles) is used as input for the call to the trained model. The remaining 10% of their data (orange circle) is compared to the output produced by the model and used to calculate the evaluation metrics.

We recommend reading the documentation (https://docs.aws.amazon.com/personalize/latest/dg/working-with-training-metrics.html) to understand the metrics, but we have also copied parts of the documentation below for convenience.

You need to understand the following terms regarding evaluation in Personalize:

- *Relevant recommendation* refers to a recommendation that matches a value in the testing data for the particular user.
- *Rank* refers to the position of a recommended item in the list of recommendations. Position 1 (the top of the list) is presumed to be the most relevant to the user.
- *Query* refers to the internal equivalent of a GetRecommendations call.

The metrics produced by Personalize are:

- **coverage**: The proportion of unique recommended items from all queries out of the total number of unique items in the training data (includes both the Items and Interactions datasets).
- **mean_reciprocal_rank_at_25**: The mean of the reciprocal ranks (https://en.wikipedia.org/wiki/Mean_reciprocal_rank) of the first relevant recommendation out of the top 25 recommendations over all queries. This metric is appropriate if you're interested in the single highest ranked recommendation.
- **normalized_discounted_cumulative_gain_at_K**: Discounted gain assumes that recommendations lower on a list of recommendations are less relevant than higher recommendations. Therefore, each recommendation is discounted (given a lower weight) by a factor dependent on its position. To produce the cumulative discounted gain (https://en.wikipedia.org/wiki/Discounted_cumulative_gain) (DCG) at K, each relevant discounted recommendation in the top K recommendations is summed together. The normalized discounted cumulative gain (NDCG) is the DCG divided by the ideal DCG such that NDCG is between 0 - 1. (The ideal DCG is where the top K recommendations are sorted by relevance.) Amazon Personalize

uses a weighting factor of 1/log(1 + position), where the top of the list is position 1. This metric rewards relevant items that appear near the top of the list, because the top of a list usually draws more attention.

- **precision_at_K**: The number of relevant recommendations out of the top K recommendations divided by K. This metric rewards precise recommendation of the relevant items.

Related Products Metrics

In [17]:

```
get_solution_metrics_response = personalize.get_solution_metrics(
    solutionVersionArn = related_solution_version_arn
)

print(json.dumps(get_solution_metrics_response, indent=2))
```


```
{
  "solutionVersionArn": "arn:aws:personalize:us-east-1:968824662561:solution/amazon-personalize-immersionday-related-products/0f3b3c74",
  "metrics": {
    "coverage": 0.9955,
    "mean_reciprocal_rank_at_25": 0.3417,
    "normalized_discounted_cumulative_gain_at_10": 0.3346,
    "normalized_discounted_cumulative_gain_at_25": 0.4198,
    "normalized_discounted_cumulative_gain_at_5": 0.27,
    "precision_at_10": 0.1051,
    "precision_at_25": 0.0681,
    "precision_at_5": 0.137
  },
  "ResponseMetadata": {
    "RequestId": "f9bb229a-08b6-4d58-8811-e9e0d2571121",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "content-type": "application/x-amz-json-1.1",
      "date": "Sun, 18 Apr 2021 04:19:29 GMT",
      "x-amzn-requestid": "f9bb229a-08b6-4d58-8811-e9e0d2571121",
      "content-length": "429",
      "connection": "keep-alive"
    },
    "RetryAttempts": 0
  }
}
```


Product Recommendations Metrics

In [18]:

```
get_solution_metrics_response = personalize.get_solution_metrics(
    solutionVersionArn = recommend_solution_version_arn
)

print(json.dumps(get_solution_metrics_response, indent=2))
```


```
{
  "solutionVersionArn": "arn:aws:personalize:us-east-1:968824662561:solution/amazon-personalize-immersionday-p
roduct-personalization/dc9e29c9",
  "metrics": {
    "coverage": 0.9486,
    "mean_reciprocal_rank_at_25": 0.5639,
    "normalized_discounted_cumulative_gain_at_10": 0.5325,
    "normalized_discounted_cumulative_gain_at_25": 0.5898,
    "normalized_discounted_cumulative_gain_at_5": 0.464,
    "precision_at_10": 0.1662,
    "precision_at_25": 0.0859,
    "precision_at_5": 0.2493
  },
  "ResponseMetadata": {
    "RequestId": "552c7cc3-fc43-4449-93c7-cff607aa6ded",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "content-type": "application/x-amz-json-1.1",
      "date": "Sun, 18 Apr 2021 04:19:30 GMT",
      "x-amzn-requestid": "552c7cc3-fc43-4449-93c7-cff607aa6ded",
      "content-length": "438",
      "connection": "keep-alive"
    },
    "RetryAttempts": 0
  }
}
```


Personalized Ranking Metrics

In [19]:

```
get_solution_metrics_response = personalize.get_solution_metrics(
    solutionVersionArn = ranking_solution_version_arn
)

print(json.dumps(get_solution_metrics_response, indent=2))
```


```
{
  "solutionVersionArn": "arn:aws:personalize:us-east-1:968824662561:solution/amazon-personalize-immersionday-p
ersonalized-ranking/7db3eae3",
  "metrics": {
    "coverage": 0.9653,
    "mean_reciprocal_rank_at_25": 0.6129,
    "normalized_discounted_cumulative_gain_at_10": 0.5502,
    "normalized_discounted_cumulative_gain_at_25": 0.6118,
    "normalized_discounted_cumulative_gain_at_5": 0.493,
    "precision_at_10": 0.17,
    "precision_at_25": 0.089,
    "precision_at_5": 0.2675
  },
  "ResponseMetadata": {
    "RequestId": "2240b54e-d7a5-4072-b22d-e5c4eee56fa3",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "content-type": "application/x-amz-json-1.1",
      "date": "Sun, 18 Apr 2021 04:19:32 GMT",
      "x-amzn-requestid": "2240b54e-d7a5-4072-b22d-e5c4eee56fa3",
      "content-length": "432",
      "connection": "keep-alive"
    },
    "RetryAttempts": 0
  }
}
```

Just a quick comment on this one, here we see again a precision of near (2.7% for full, 2.2% for small), as this is based on User Personalization, that is to be expected. However the sample items are not the same for validaiton, thus the low scoring.

Using evaluation metrics

Back to top

It is important to use evaluation metrics carefully. There are a number of factors to keep in mind.

- If there is an existing recommendation system in place, this will have influenced the user's interaction history which you use to train your new solutions. This means the evaluation metrics are biased to favor the existing solution. If you work to push the evaluation metrics to match or exceed the existing solution, you may just be pushing the User Personalization to behave like the existing solution and might not end up with something better.
- The HRNN Coldstart recipe is difficult to evaluate using the metrics produced by Amazon Personalize. The aim of the recipe is to recommend items which are new to your business. Therefore, these items will not appear in the existing user transaction data which is used to compute the evaluation metrics. As a result, HRNN Coldstart will never appear to perform better than the other recipes, when compared on the evaluation metrics alone. Note: The User Personalization recipe also includes improved cold start functionality

Keeping in mind these factors, the evaluation metrics produced by Personalize are generally useful for two cases:

1. Comparing the performance of solution versions trained on the same recipe, but with different values for the hyperparameters and features (impression data etc)
2. Comparing the performance of solution versions trained on different recipes (except HRNN Coldstart).

Properly evaluating a recommendation system is always best done through A/B testing while measuring actual business outcomes. Since recommendations generated by a system usually influence the user behavior which it is based on, it is better to run small experiments and apply A/B testing for longer periods of time. Over time, the bias from the existing model will fade.

Create Campaigns

Once we're satisfied with our solution versions, we need to create Campaigns for each solution version. When creating a campaign you specify the minimum transactions per second (`minProvisionedTPS`) that you expect to make against the service for this campaign. Personalize will automatically scale the inference endpoint up and down for the campaign to match demand but will never scale below `minProvisionedTPS` .

Let's create campaigns for our three solution versions with each set at `minProvisionedTPS` of 1.

Create Related Products Campaign

In [20]:

```
create_campaign_response = personalize.create_campaign(
    name = "amazon-personalize-immersionday-related-products",
    solutionVersionArn = related_solution_version_arn,
    minProvisionedTPS = 1
)

related_campaign_arn = create_campaign_response['campaignArn']
print(json.dumps(create_campaign_response, indent=2))
```


```
{
  "campaignArn": "arn:aws:personalize:us-east-1:968824662561:campaign/amazon-personalize-immersionday-related-
products",
  "ResponseMetadata": {
    "RequestId": "cd815451-7b01-48e0-ab83-e39210ada8c9",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "content-type": "application/x-amz-json-1.1",
      "date": "Sun, 18 Apr 2021 04:19:35 GMT",
      "x-amzn-requestid": "cd815451-7b01-48e0-ab83-e39210ada8c9",
      "content-length": "118",
      "connection": "keep-alive"
    },
    "RetryAttempts": 0
  }
}
```


Create Product Recommendation Campaign

In [21]:

```
create_campaign_response = personalize.create_campaign(
    name = "amazon-personalize-immersionday-product-personalization",
    solutionVersionArn = recommend_solution_version_arn,
    minProvisionedTPS = 1
)

recommend_campaign_arn = create_campaign_response['campaignArn']
print(json.dumps(create_campaign_response, indent=2))
```


```
{
  "campaignArn": "arn:aws:personalize:us-east-1:968824662561:campaign/amazon-personalize-immersionday-product-
personalization",
  "ResponseMetadata": {
    "RequestId": "5a7faab8-0d26-4e23-8b24-2dc84b30a812",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "content-type": "application/x-amz-json-1.1",
      "date": "Sun, 18 Apr 2021 04:19:36 GMT",
      "x-amzn-requestid": "5a7faab8-0d26-4e23-8b24-2dc84b30a812",
      "content-length": "125",
      "connection": "keep-alive"
    },
    "RetryAttempts": 0
  }
}
```


Create Personalized Ranking Campaign

In [22]:

```
create_campaign_response = personalize.create_campaign(
    name = "amazon-personalize-immersionday-personalized-ranking",
    solutionVersionArn = ranking_solution_version_arn,
    minProvisionedTPS = 1
)

ranking_campaign_arn = create_campaign_response['campaignArn']
print(json.dumps(create_campaign_response, indent=2))
```


```
{
  "campaignArn": "arn:aws:personalize:us-east-1:968824662561:campaign/amazon-personalize-immersionday-personal
ized-ranking",
  "ResponseMetadata": {
    "RequestId": "14f3f9f9-e859-43fe-87b0-9ca23f8d1b71",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "content-type": "application/x-amz-json-1.1",
      "date": "Sun, 18 Apr 2021 04:19:38 GMT",
      "x-amzn-requestid": "14f3f9f9-e859-43fe-87b0-9ca23f8d1b71",
      "content-length": "122",
      "connection": "keep-alive"
    },
    "RetryAttempts": 0
  }
}
```


Wait for Related Products Campaign to Have ACTIVE Status

It can take 20-30 minutes for the campaigns to be fully created.

While you are waiting for this to complete you can learn more about campaigns in [the documentation (https://docs.aws.amazon.com/personalize/latest/dg/campaigns.html)](https://docs.aws.amazon.com/personalize/latest/dg/campaigns.html)

In [23]:

```
%%time

campaign_arns = [ related_campaign_arn, recommend_campaign_arn, ranking_campaign_arn ]

max_time = time.time() + 3*60*60 # 3 hours
while time.time() < max_time:
    for campaign_arn in reversed(campaign_arns):
        campaign_response = personalize.describe_campaign(
            campaignArn = campaign_arn
        )
        status = campaign_response["campaign"]["status"]

        if status == "ACTIVE":
            print(f'Campaign {campaign_arn} successfully completed')
            campaign_arns.remove(campaign_arn)
        elif status == "CREATE FAILED":
            print(f'Campaign {campaign_arn} failed')
            if campaign_response.get('failureReason'):
                print('   Reason: ' + campaign_response['failureReason'])
            campaign_arns.remove(campaign_arn)

    if len(campaign_arns) > 0:
        print('At least one campaign is still in progress')
        time.sleep(60)
    else:
        print("All campaigns have completed")
        break
```

```
At least one campaign is still in progress
At least one campaign is still in progress
At least one campaign is still in progress
At least one campaign is still in progress
At least one campaign is still in progress
At least one campaign is still in progress
At least one campaign is still in progress
At least one campaign is still in progress
At least one campaign is still in progress
At least one campaign is still in progress
Campaign arn:aws:personalize:us-east-1:968824662561:campaign/amazon-personalize-immersionday-product-personali
zation successfully completed
Campaign arn:aws:personalize:us-east-1:968824662561:campaign/amazon-personalize-immersionday-related-products
successfully completed
At least one campaign is still in progress
Campaign arn:aws:personalize:us-east-1:968824662561:campaign/amazon-personalize-immersionday-personalized-rank
ing successfully completed
All campaigns have completed
CPU times: user 147 ms, sys: 0 ns, total: 147 ms
Wall time: 11min 1s
```

Create Purchased Products Filter

Amazon Personalize supports the ability to create filters (https://docs.aws.amazon.com/personalize/latest/dg/filter.html) that can be used to exclude items from being recommended that meet a filter expression. Since it's a poor user experience to recommend products that a user has already purchased, we will create a filter that excludes recently purchased products. We'll do this by creating a filter expression that excludes items that have an interaction with an event type of `OrderCompleted` for the user.

> As noted above, the Retail Demo Store web application streams clickstream events to Personalize when the user performs various actions such as viewing and purchasing products. The filter created below allows us to use those events as exclusion criteria. See the AnalyticsHandler.js (https://github.com/aws-samples/retail-demo-store/blob/master/src/web-ui/src/analytics/AnalyticsHandler.js) file for the code that sends clickstream events.

In [24]:

```
response = personalize.create_filter(
    name = 'immersion-day-filter-purchased-products',
    datasetGroupArn = dataset_group_arn,
    filterExpression = 'EXCLUDE itemId WHERE INTERACTIONS.event_type in ("OrderCompleted")'
)

filter_arn = response['filterArn']
print(f'Filter ARN: {filter_arn}')
```

```
Filter ARN: arn:aws:personalize:us-east-1:968824662561:filter/immersion-day-filter-purchased-products
```

Wait for Filter Status to Become ACTIVE

The filter should take a minute or so to become active.

In [25]:

```
status = None
max_time = time.time() + 60*60 # 1 hours
while time.time() < max_time:
    describe_filter_response = personalize.describe_filter(
        filterArn = filter_arn
    )
    status = describe_filter_response["filter"]["status"]
    print("Filter: {}".format(status))

    if status == "ACTIVE" or status == "CREATE FAILED":
        break

    time.sleep(15)
```

```
Filter: CREATE PENDING
Filter: CREATE IN_PROGRESS
Filter: CREATE IN_PROGRESS
Filter: CREATE IN_PROGRESS
Filter: CREATE IN_PROGRESS
Filter: CREATE IN_PROGRESS
Filter: CREATE IN_PROGRESS
Filter: CREATE IN_PROGRESS
Filter: CREATE IN_PROGRESS
Filter: CREATE IN_PROGRESS
Filter: CREATE IN_PROGRESS
Filter: CREATE IN_PROGRESS
Filter: CREATE IN_PROGRESS
Filter: CREATE IN_PROGRESS
Filter: CREATE IN_PROGRESS
Filter: CREATE IN_PROGRESS
Filter: CREATE IN_PROGRESS
Filter: ACTIVE
```

Storing useful variables

Back to top

Before exiting this notebook, run the following cells to save the version ARNs for use in the next notebook.

In [26]:

```
%store related_solution_arn
%store recommend_solution_arn
%store related_solution_version_arn
%store recommend_solution_version_arn
%store ranking_solution_arn
%store ranking_solution_version_arn
%store related_campaign_arn
%store recommend_campaign_arn
%store ranking_campaign_arn
%store filter_arn
```

```
Stored 'related_solution_arn' (str)
Stored 'recommend_solution_arn' (str)
Stored 'related_solution_version_arn' (str)
Stored 'recommend_solution_version_arn' (str)
Stored 'ranking_solution_arn' (str)
Stored 'ranking_solution_version_arn' (str)
Stored 'related_campaign_arn' (str)
Stored 'recommend_campaign_arn' (str)
Stored 'ranking_campaign_arn' (str)
Stored 'filter_arn' (str)
```

In [ ]:

```
```

In [ ]:

```
```