Microsoft Recommenders

Using Jupyter notebooks, Python and the https://www.github.com/Microsoft/Recommenders repo, I'll demonstrate how you can make your own Recommendation system using AzureML. I'll note the benefits of Azure ML and its ability to use all of the usual Python tools and libraries that are popular today. In addition, various models used for recommendation systems will be compared using predictions developed on the MovieLens dataset.

https://github.com/Microsoft/Recommenders

https://notebooks.azure.com/

Microsoft

Microsoft

Learn to make your own Recommendation System with AzureML

Beth Zeranski
Principal Experimentation Engineer

http://github.com/bethz/2019GlobalAzureBootcamp

https://github.com/Microsoft/Recommenders

https://twitter.com/bethzeranski

Linkedin: aka.ms/bethz

April 27, 2019

Agenda

- Goal
- Why Recommendation Systems?
- What is a Recommendation System?
  - Powerpoint's *AWESOME* Recommendation System
- http://github.com/Microsoft/Recommenders
- Solution Architecture
- Data – public movielens data
- Walk through Repo
- Azure Machine Learning, AzureML
- Workflow for SAR, Smart Adaptive Recommendation solution
  - https://github.com/Microsoft/Product-Recommendations/blob/master/doc/sar.md
- Code Walk through

Goal

Provide sufficient context and guidance so you can use the Recommenders github repo to make your own Recommendation System

Highlight notebooks demonstrate multiple algorithms, automated parameter tuning and even algorithm selection to assist.

https://github.com/Microsoft/Recommenders

Why Recommendation Systems?

Huge Revenue Impact!

“35% of what consumers purchase on Amazon and 75% of what they watch on Netflix come from recommendations algorithms”

McKinsey & Co

What is a Recommendation System?

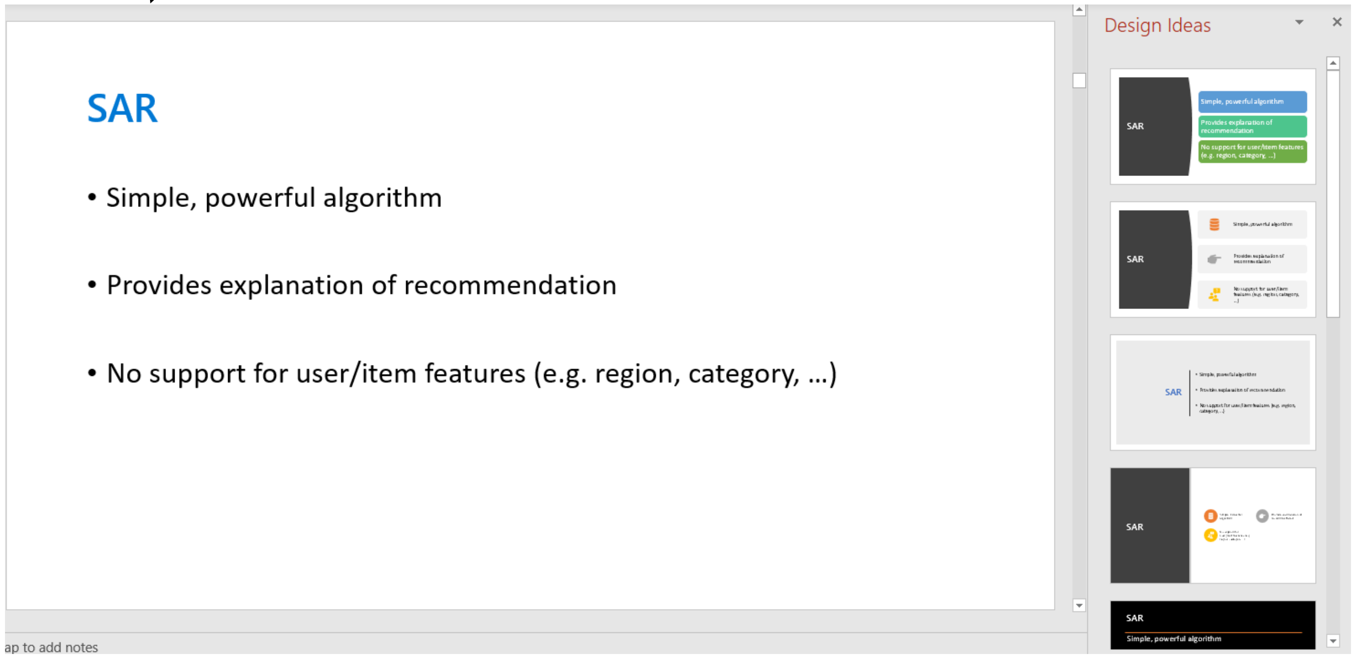
…is a subclass of information filtering systems that seek to predict the ‘rating’ or ‘preference’ that a user would give to an item

https://en.wikipedia.org/wiki/Recommender_system

Powerpoint has an *AWESOME* Recommendation System

My simple slide

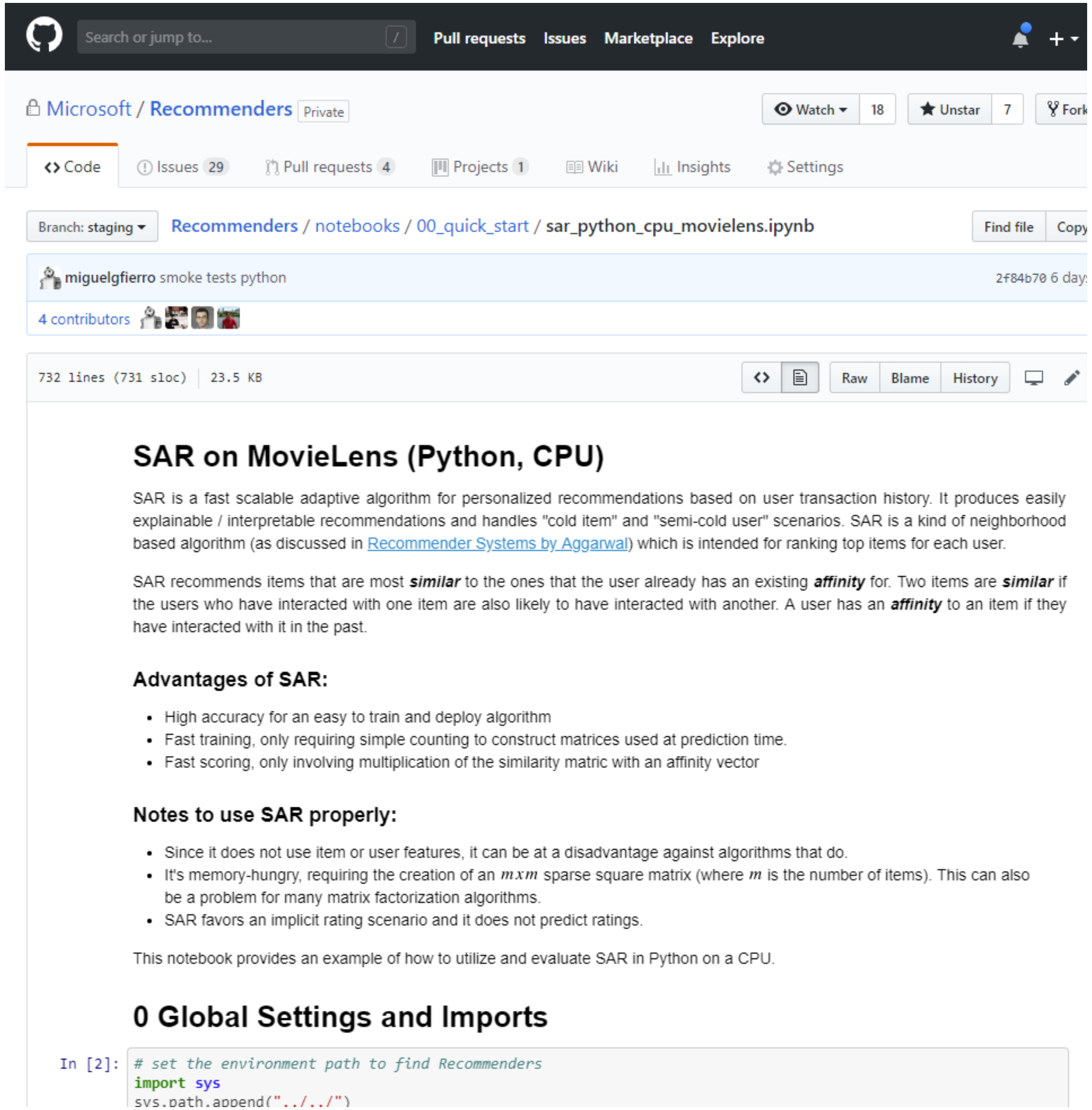
Design Ideas



- Powerpoint's "Design Ideas" Recommendation system created a number of improved layouts which are more compelling for my slides
- Fast and Easy

Key Themes of our *Recommenders repo*

- Reduce time and effort required for Proof of Concept
- Lower barrier of entry for implementation
- Easily scale out on Azure
- Widely usable and open-source
- Provide advanced algorithms


Search or jump to... Pull requests Issues Marketplace Explore

Microsoft / Recommenders Private

Watch 18 Unstar 7 Fork

Code Issues 29 Pull requests 4 Projects 1 Wiki Insights Settings

Branch: staging Recommenders / notebooks / 00_quick_start / sar_python_cpu_movielens.ipynb Find file Copy

miguelgfierro smoke tests python 2f84b70 6 day

4 contributors

732 lines (731 sloc) 23.5 KB Raw Blame History

SAR on MovieLens (Python, CPU)

SAR is a fast scalable adaptive algorithm for personalized recommendations based on user transaction history. It produces easily explainable / interpretable recommendations and handles "cold item" and "semi-cold user" scenarios. SAR is a kind of neighborhood based algorithm (as discussed in Recommender Systems by Aggarwal) which is intended for ranking top items for each user.

SAR recommends items that are most ***similar*** to the ones that the user already has an existing ***affinity*** for. Two items are ***similar*** if the users who have interacted with one item are also likely to have interacted with another. A user has an ***affinity*** to an item if they have interacted with it in the past.

Advantages of SAR:

- High accuracy for an easy to train and deploy algorithm
- Fast training, only requiring simple counting to construct matrices used at prediction time.
- Fast scoring, only involving multiplication of the similarity matric with an affinity vector

Notes to use SAR properly:

- Since it does not use item or user features, it can be at a disadvantage against algorithms that do.
- It's memory-hungry, requiring the creation of an mxm sparse square matrix (where m is the number of items). This can also be a problem for many matrix factorization algorithms.
- SAR favors an implicit rating scenario and it does not predict ratings.

This notebook provides an example of how to utilize and evaluate SAR in Python on a CPU.

0 Global Settings and Imports

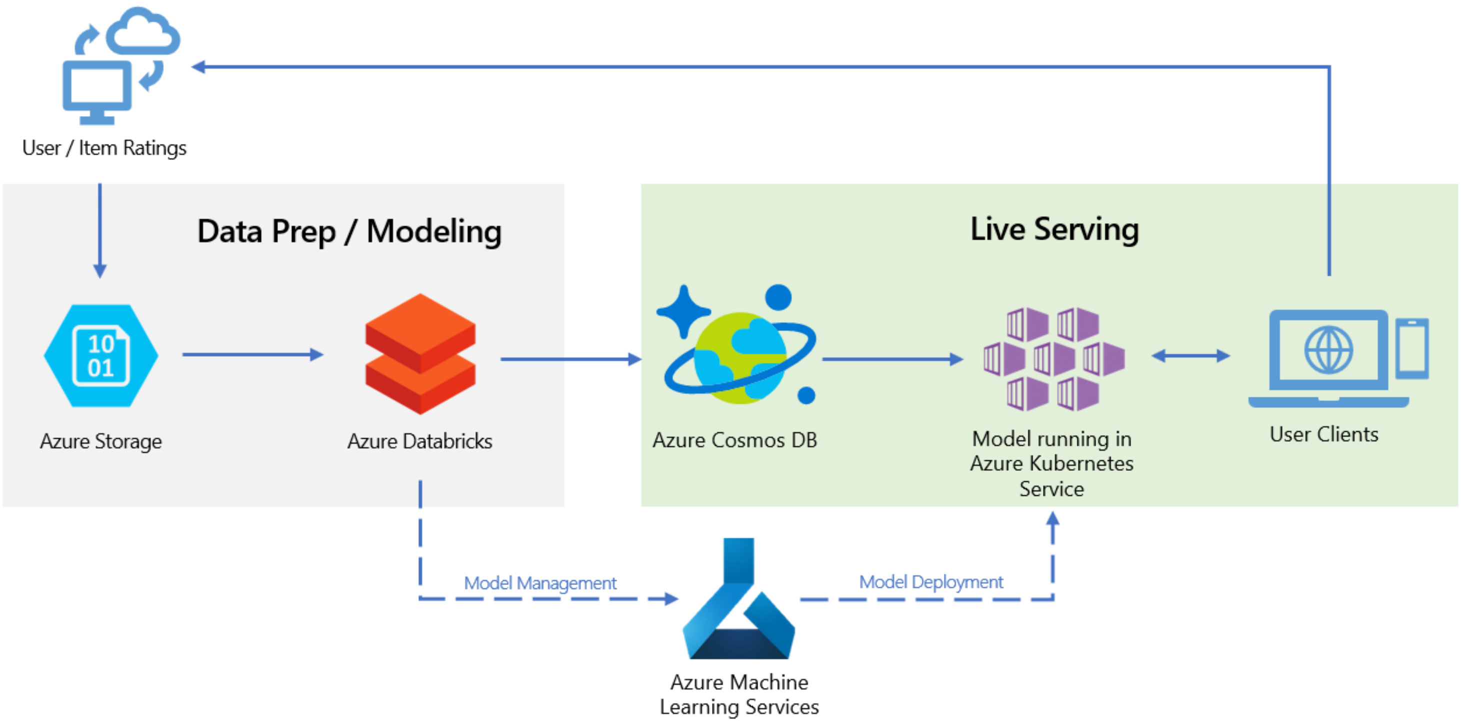
In [2]:
```
# set the environment path to find Recommenders
import sys
sys.path.append("../../")
```

https://github.com/Microsoft/Recommenders

Microsoft/Recommenders

- Microsoft/Recommenders
  - Collaborative development efforts of Microsoft Cloud & AI data scientists, Microsoft Research researchers, academia researchers, etc.
  - Github url: https://github.com/Microsoft/Recommenders
  - Contents
    - Utilities: modular functions for model creation, data manipulation, evaluation, etc.
      - Algorithms: SVD, SAR, ALS, NCF, Wide&Deep, xDeepFM, DKN, etc.
    - Notebooks: HOW-TO examples for end to end recommender building.
  - Highlights
    - 2800+ stars on GitHub
    - Featured in YC Hacker News, O'Reily Data Newsletter, GitHub weekly trending list, etc.
  - Any contribution to the repo will be highly appreciated!
    - Create issue/PR directly in the GitHub repo
    - Send email to RecoDevTeam@service.microsoft.com for any collaboration

Recommenders Architecture



UserId	MovieId	Rating	Timestamp
196	242	3.0	881250949
186	302	3.0	891717742
22	377	1.0	878887116
244	51	2.0	880606923
166	346	1.0	886397596
298	474	4.0	884182806
115	265	2.0	881171488

Data format

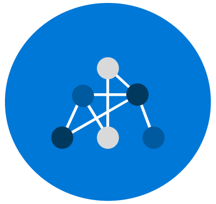
- User ID
- Item ID
- Time (Optional)
- Event Weight (Optional)
  - Rating
  - Number of item views
  - Number of purchases

Repository General Components

Reco Utilities

Dataset helpers
Splitters
Ranking Metrics
Rating Metrics
Diversity Metrics

Notebooks

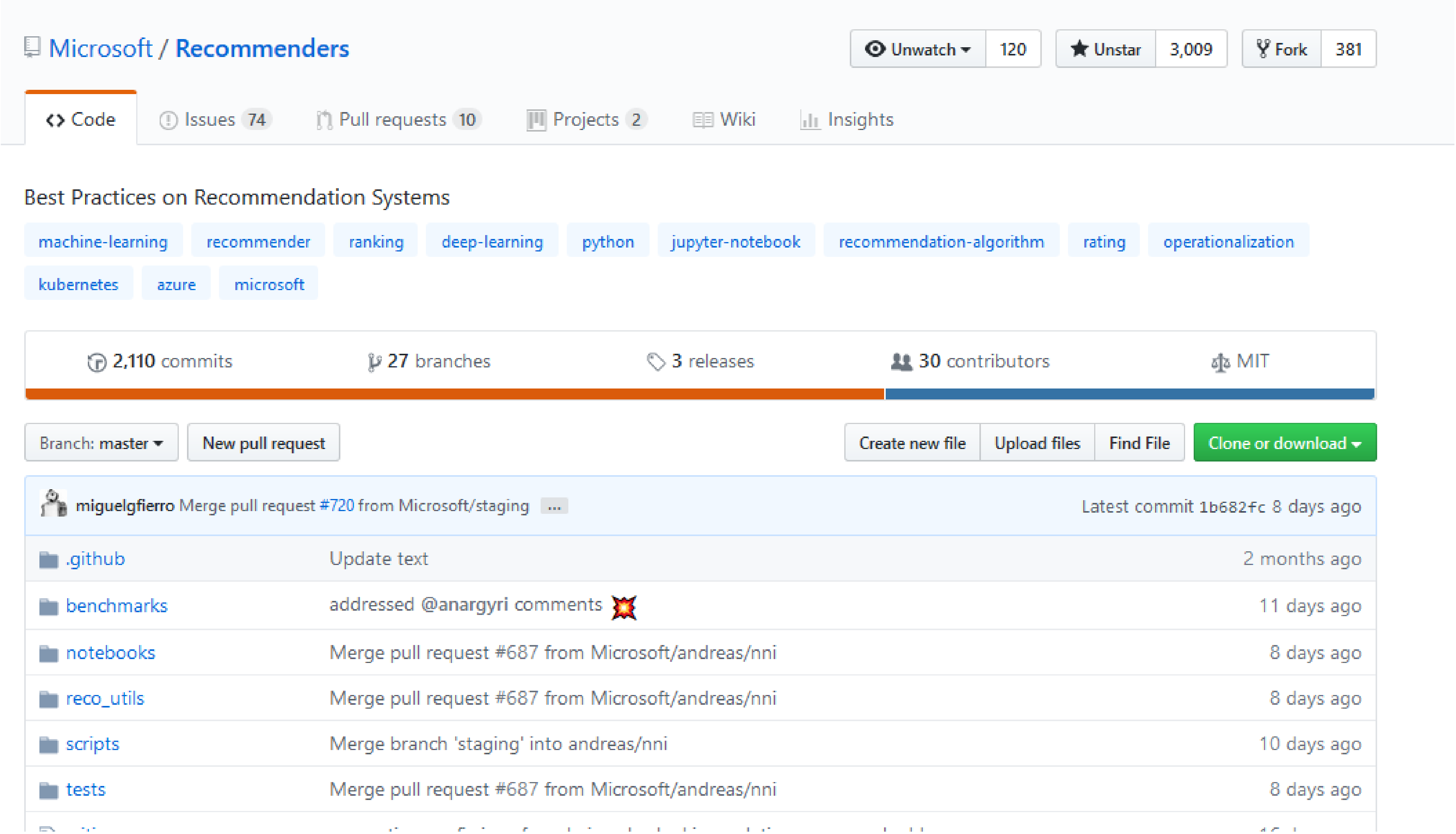


Spark ALS
SAR: Spark and Python implementation
Deep Learning algos from MSR (DeepRec)
Open source frameworks: Surprise

Tests

Unit tests
Integration tests
Nightly builds
Benchmarks updated every night

Repository Walk-thru



https://github.com/Microsoft/Recommenders

Why Azure ML Service?

Training in the cloud

Highly scalable and flexible model training

Rapid Iteration

Model Traceability

Automated Data, Output, Notebook and related Asset Retention

Azure ML's automated machine learning finds algorithms based on your data

Pipelines for repeatability and sharing

Workspace

- The workspace is the top-level resource for Azure Machine Learning service. It provides a centralized place to work with all the artifacts you create when you use Azure Machine Learning service.
- The workspace keeps a list of compute targets that you can use to train your model. It also keeps a history of the training runs, including logs, metrics, output, and a snapshot of your scripts. You use this information to determine which training run produces the best model.
- You register models with the workspace. You use a registered model and scoring scripts to create an image. You can then deploy the image to Azure Container Instances, Azure Kubernetes Service, or to a field-programmable gate array (FPGA) as a REST-based HTTP endpoint. You can also deploy the image to an Azure IoT Edge device as a module.
- You can create multiple workspaces, and each workspace can be shared by multiple people.
- Workspace is accessible in portal.azure.com or by Command Line Interface

For more info:

- https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-azure-machine-learning-architecture

Experiment

- An experiment is a grouping of many runs from a specified script. It always belongs to a workspace. When you submit a run, you provide an experiment name. Information for the run is stored under that experiment. If you submit a run and specify an experiment name that doesn't exist, a new experiment with that newly specified name is automatically created.

- For an example of using an experiment, see Quickstart: Get started with Azure Machine Learning service.

Model

- At its simplest, a model is a piece of code that takes an input and produces output. Creating a machine learning model involves selecting an algorithm, providing it with data, and tuning hyperparameters. Training is an iterative process that produces a trained model, which encapsulates what the model learned during the training process.
- A model is produced by a run in Azure Machine Learning. You can also use a model that's trained outside of Azure Machine Learning. You can register a model in an Azure Machine Learning service workspace.
- Azure Machine Learning service is framework agnostic. When you create a model, you can use any popular machine learning framework, such as Scikit-learn, XGBoost, PyTorch, TensorFlow, and Chainer.
- For an example of training a model, see Tutorial: Train an image classification model with Azure Machine Learning service.

Model registry

- The model registry keeps track of all the models in your Azure Machine Learning service workspace.
- Models are identified by name and version. Each time you register a model with the same name as an existing one, the registry assumes that it's a new version. The version is incremented, and the new model is registered under the same name.
- When you register the model, you can provide additional metadata tags and then use the tags when you search for models.
- You can't delete models that are being used by an image.
- For an example of registering a model, see Train an image classification model with Azure Machine Learning.

Run Configuration

An environment defined for each compute target so that you can run your training scripts on different compute targets *without* modification.

For example, to use a local target configuration:

```
from azureml.core import ScriptRunConfig
import os
script_folder = os.getcwd()
src = ScriptRunConfig(source_directory = script_folder,
script = 'train.py', run_config = run_local)
run = exp.submit(src)
run.wait_for_completion(show_output = True)
```

Switch the same experiment to run in a different compute target by using a different run configuration, such as the AzureML Compute target:

```
from azureml.core import ScriptRunConfig

src = ScriptRunConfig(source_directory = script_folder,
script = 'train.py', run_config = run_amlcompute)
run = exp.submit(src)
run.wait_for_completion(show_output = True)
```

Command Line Interface

Attach a run configuration before submitting an experiment	az ml project attach --experiment-name myhistory
Start a run of your experiment. The compute target uses the run configuration to create the training environment for the model.	az ml run submit -c myrunconfig train.py
View list of submitted experiments	az ml history list

During local training or cloud training, have you ever lost a run?

- Days or weeks go by and models+info go missing
- Azure ML service enables you to log artifacts from local experimentation or cloud training painlessly to the cloud. That includes inputs, notebooks, outputs and more!
- That means the work is easily repeatable (*and findable*) by you or anyone with access

Other info

- Exporting and deleting data
- https://docs.microsoft.com/en-us/azure/machine-learning/service/how-to-export-delete-data
- Start, monitor and cancel training runs
- https://docs.microsoft.com/en-us/azure/machine-learning/service/how-to-manage-runs
- Log Metrics
- https://docs.microsoft.com/en-us/azure/machine-learning/service/how-to-track-experiments
- Estimator – convenient object that wraps run configuration information to specify how a script is executed for model training
- https://github.com/Azure/MachineLearningNotebooks/blob/master/how-to-use-azureml/training-with-deep-learning/how-to-use-estimator/how-to-use-estimator.ipynb

Workflow

The machine learning workflow generally follows this sequence:

1. Develop machine learning training scripts in **Python**.
2. Create and configure a **compute target**.
3. Attach the compute target to your workspace.
4. Configure the compute target so it contains the Python environment and package dependencies required.
5. **Submit the scripts** to the configured compute target to run in that environment. During training, the scripts can read from or write to **datastore**. The records of execution are saved as **runs** in the **workspace** and grouped under **experiments**.
6. **Query the experiment** for logged metrics from the current and past runs. If the metrics don't indicate a desired outcome, loop back to step 1 and iterate on your scripts. All artifacts are stored together for revisiting prior work and outcomes.
7. After a satisfactory run is identified, register the persisted model in the **model registry**.
8. Develop a scoring script.
9. **Create an image** and register it in the **image registry**.
10. **Deploy the image** as a **web service** in Azure.

Start Here

- https://github.com/Azure/MachineLearningNotebooks/blob/master/configuration.ipynb

Demo: SAR Quickstart

https://github.com/Microsoft/Recommenders/blob/master/notebooks/00_quick_start/sar_movielens_with_azureml.ipynb
https://github.com/Microsoft/Recommenders/blob/master/notebooks/run_notebook_on_azureml.ipynb
https://github.com/Microsoft/Recommenders/blob/master/notebooks/00_quick_start/sar_movielens.ipynb

References

Great detail on Algorithm and deeper dive on creating a system
www.github.com/yueguoguo/pakdd2019tutorial

Train an image classification model with AzureML
https://github.com/Azure/MachineLearningNotebooks/blob/master/tutorials/img-classification-part1-training.ipynb

Deploy an image classification model in Azure Container Instance
https://github.com/Azure/MachineLearningNotebooks/blob/master/tutorials/img-classification-part2-deploy.ipynb

Textbook:
Recommender Systems: The Textbook by Charu C. Aggarwal
https://www.amazon.com/Recommender-Systems-Textbook-Charu-Aggarwal/dp/3319296574

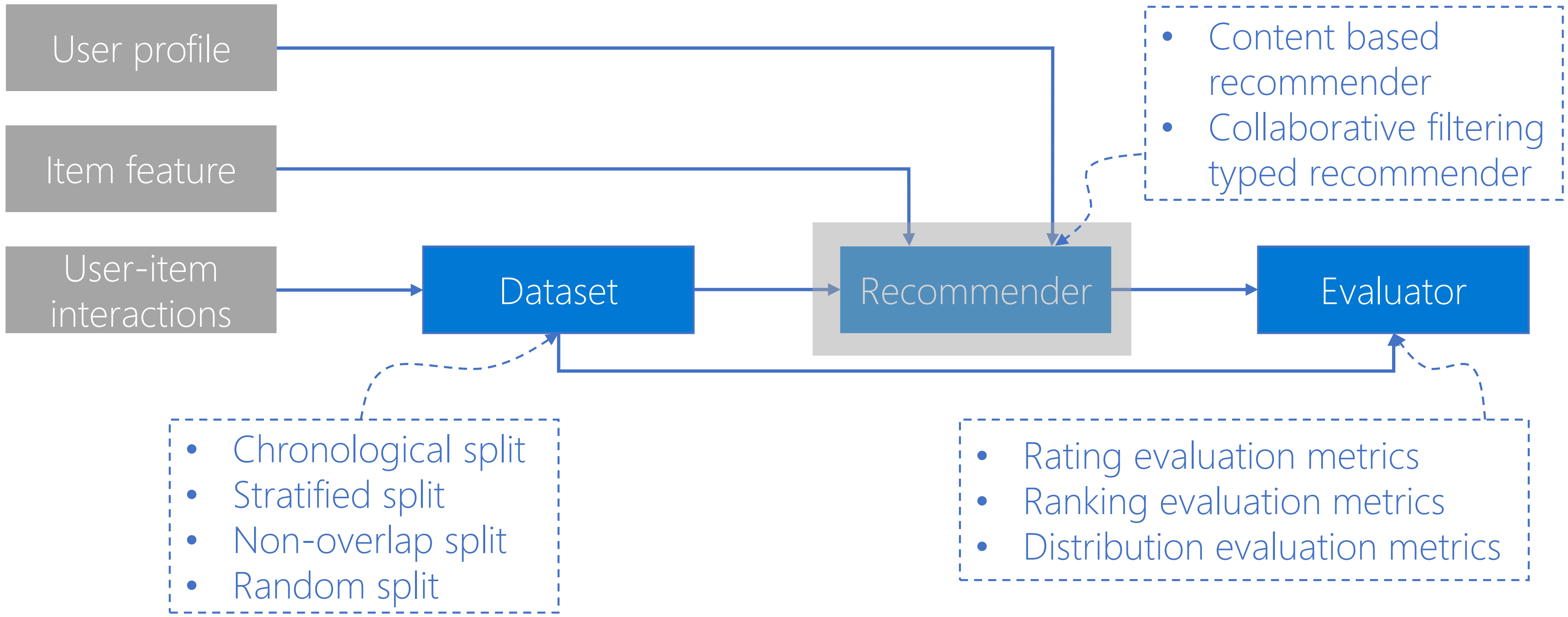
Blog:
https://azure.microsoft.com/en-us/blog/experimentation-using-azure-machine-learning/

Questions?

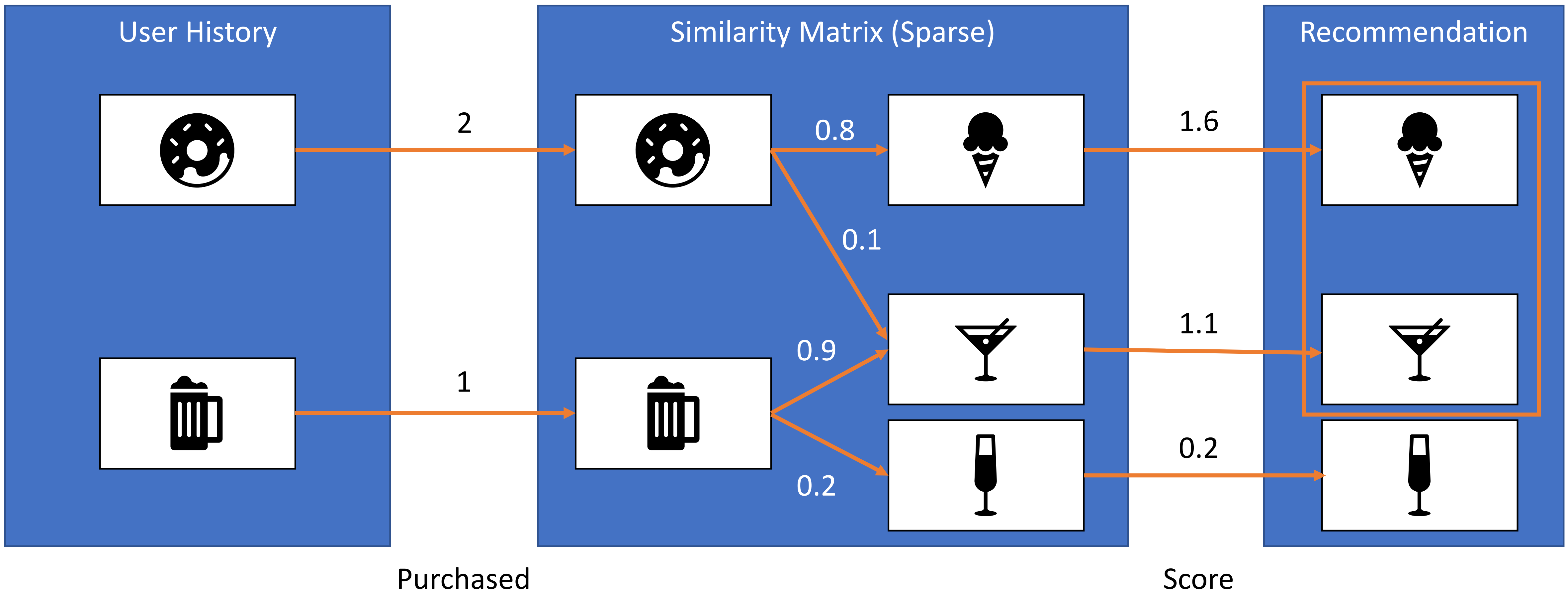
Microsoft Recommenders

https://github.com/Microsoft/Recommenders

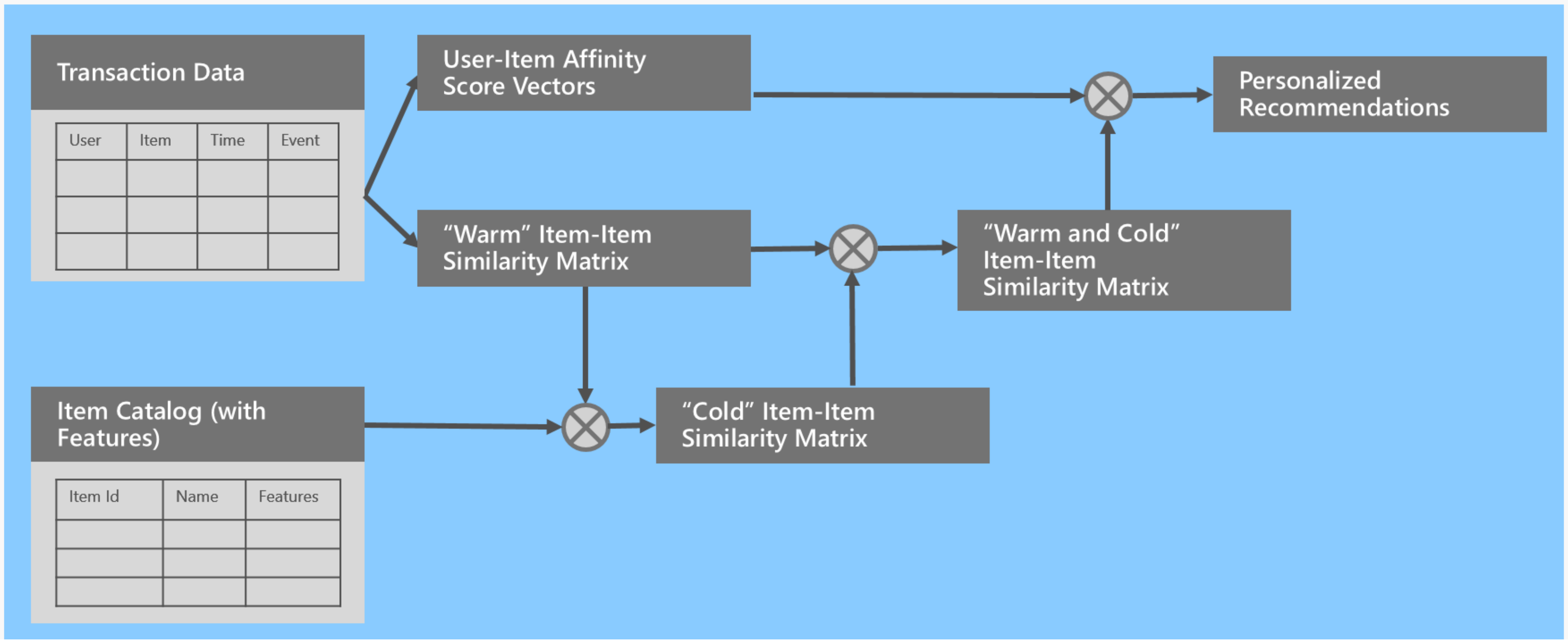
Example of an end-to-end pipeline



SAR


User History
Similarity Matrix (Sparse)
Recommendation
2
1
0.8
0.1
0.9
0.2
1.6
1.1
0.2
Purchased
Score

SAR overview



Data Splitting

- Stratified splitting
  - e.g. movie ratings
  - Avoid cold-users
  - Train users = test users
- Chronological splitting
  - e.g. fashion items, news, …
  - User taste changes over time
  - Obey time-dependency

SAR

- Simple, powerful algorithm
- Provides explanation of recommendation
- No support for user/item features (e.g. region, category, …)